

z/OS Connect Enterprise Edition
3.0

z/OS Connect Enterprise Edition



Note

Before using this information and the product it supports, read the information in “[Notices](#)” on page [831](#).

Last updated: 2021-04-01

This edition applies to the IBM Version 3 Release 0 (product number 5655-CE3) and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 2015, 2019.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Chapter 1. Why z/OS Connect Enterprise Edition?.....	1
Chapter 2. What is IBM z/OS Connect Enterprise Edition?.....	3
z/OS Connect EE concepts.....	6
Differences between products.....	7
z/OS assets as REST APIs.....	8
z/OS applications to call REST APIs.....	9
The z/OS Connect EE API toolkit.....	10
The z/OS Connect EE build toolkit.....	13
DevOps with z/OS Connect EE.....	14
Chapter 3. z/OS Connect EE change history.....	17
Removal notices.....	53
Chapter 4. Quick Start Scenarios.....	55
Prepare a sample SoR application.....	59
Prepare the sample CICS application.....	59
Prepare the sample IMS application.....	61
Prepare the sample IMS database.....	62
Prepare the sample IBM MQ stock query application.....	63
Prepare the sample Db2 native REST services.....	66
Create a server and connect to a SoR.....	69
Create a server and connect to CICS.....	69
Create a server and connect to IMS.....	72
Create a server and connect to an IMS database.....	74
Create a server to connect to IBM MQ.....	77
Create a server to connect to Db2.....	80
Create services.....	82
Create a CICS service.....	83
Create an IMS service.....	89
Create a two-way IBM MQ service.....	96
Create an IMS database service.....	103
Create a Db2 service.....	108
Create APIs.....	114
Create an API to invoke the CICS catalog manager services.....	115
Create an API to invoke the IMS phone book service.....	131
Create an API to invoke the IBM MQ stock query service.....	149
Create a Db2 API to invoke the Db2 employee services.....	154
Build service archives and API archives for DevOps.....	167
Build a service archive with the build toolkit for DevOps.....	167
Build an API archive with the build toolkit for DevOps.....	168
Configure HA.....	170
Configure HA for connections to z/OS Connect EE.....	170
Configure HA for connections to CICS.....	174
Calling an API from a System of Record (SoR).....	176
Call an API from a CICS application.....	176
Call an API from an IMS or z/OS application.....	184
Chapter 5. Installing.....	195
Requirements.....	195

Installing z/OS Connect EE.....	196
Creating the shared directory.....	196
Setting up the product extensions directory.....	197
Installing the build toolkit.....	199
Updating z/OS Connect Enterprise Edition.....	199
Converting to z/OS Connect Enterprise Edition Unlimited.....	199
Preferred method.....	200
Alternative method.....	200
Installing z/OS Explorer and the z/OS Connect EE API toolkit.....	201
Updating z/OS Explorer and the z/OS Connect EE API toolkit.....	202
Migrating from z/OS Connect V1.....	202
Upgrading from z/OS Connect EE V2.....	203
Chapter 6. Performance considerations.....	205
Performance best practices	205
How payload size and transformation affect performance.....	209
Chapter 7. Configuring.....	217
Creating a z/OS Connect EE server.....	217
Deleting or renaming a z/OS Connect EE Server.....	219
Configuring services.....	219
Using the CICS service provider.....	221
Configuring an IPIC connection to CICS.....	221
Transaction override precedence.....	225
Configuring security for an IPIC connection.....	225
Handling request timeouts.....	229
Configuring IPIC High Availability.....	229
Context containers.....	230
Using the WOLA service provider.....	231
Configuring CICS to use WOLA.....	233
Registering CICS with WOLA.....	236
Using the IMS service provider.....	238
Operational requirements.....	238
IMS service security process flow.....	238
Configuring PassTicket support for IMS services.....	241
Configuring for the IMS service provider.....	243
Verifying server communication with IMS.....	245
IMS mobile services and service registry	246
Security configuration for the IMS service provider.....	250
Transaction code override precedence.....	256
Using the IMS database service provider.....	257
Operational requirements.....	257
IMS database service security process flow.....	258
Configuring PassTicket support for IMS database services.....	260
Configuring user type converter support for IMS database services.....	262
Configuring for the IMS database service provider.....	263
Creating an IMS database host connection.....	265
Using the IBM MQ service provider.....	266
Service types.....	268
HTTP considerations for the IBM MQ service provider.....	270
Transactional considerations.....	272
HTTP headers that can be used with the IBM MQ service provider.....	272
Security requirements for the IBM MQ service provider.....	273
Data transformations with the IBM MQ service provider.....	274
Handling QRFH2 headers with the IBM MQ service provider.....	279
Migrating a service to the new IBM MQ service provider.....	280
Overriding IBM MQ service properties.....	281

Using Db2 services.....	281
Configuring a REST client connection to Db2.....	282
Configuring security for a REST client connection to Db2.....	282
Overriding Db2 service properties.....	286
Using the REST client service provider.....	286
Configuring a REST client connection in z/OS Connect EE.....	286
Configuring security for a REST client connection.....	287
Using other service providers.....	288
Configuring interceptors.....	288
Configuring the file system logger interceptor.....	290
Configuring the audit interceptor.....	291
Configuring the authorization interceptor.....	293
Configuring service-specific interceptors.....	293
Configuring API-specific interceptors.....	294
Configuring API requester-specific interceptors.....	295
Configuring Data Transformers.....	296
Creating bind and schema files.....	297
Configuring service archives.....	298
Configuring APIs.....	299
Configuring CORS.....	300
Configuration updates on demand.....	300
Using an MBean to trigger updates.....	301
Invoking the FileNotificationMBean from a Java program.....	302
Invoking FileNotificationMBean from the REST API.....	303
Using the FileTransferMBean.....	304
Configuring for API requesters.....	304
Configuring z/OS Connect EE to support API requesters.....	304
Configuring CICS to access z/OS Connect EE to call APIs.....	306
Configuring IMS to access z/OS Connect EE for API calls.....	309
Configuring other z/OS applications to access z/OS Connect EE for API calls.....	312
Configuring an HA environment for RESTful API calls from z/OS applications.....	314
Chapter 8. High availability.....	317
Connectivity for high availability.....	317
Sysplex distributor.....	317
TCP/IP port sharing.....	318
High availability of z/OS Connect EE servers.....	319
High availability of SoRs.....	320
Sharing server configuration.....	325
Splitting server configuration information.....	326
Server-specific configuration elements.....	327
Location of shared configuration files.....	327
Maintenance of API, service, and API requester deployments.....	327
Administration and operation tasks in an HA environment	328
Stopping and starting servers in an HA environment.....	328
Management of APIs and services in an HA environment.....	329
Chapter 9. Securing z/OS Connect EE resources.....	331
Overview of z/OS Connect EE security.....	331
API provider confidentiality and integrity.....	336
How to configure a TLS connection with RACF key rings.....	339
API provider authentication and identification.....	349
API provider basic authentication.....	354
API provider third-party authentication.....	360
API provider client certificate authentication.....	371
API provider authorization.....	373
Configuring the zosConnectAccess role.....	377

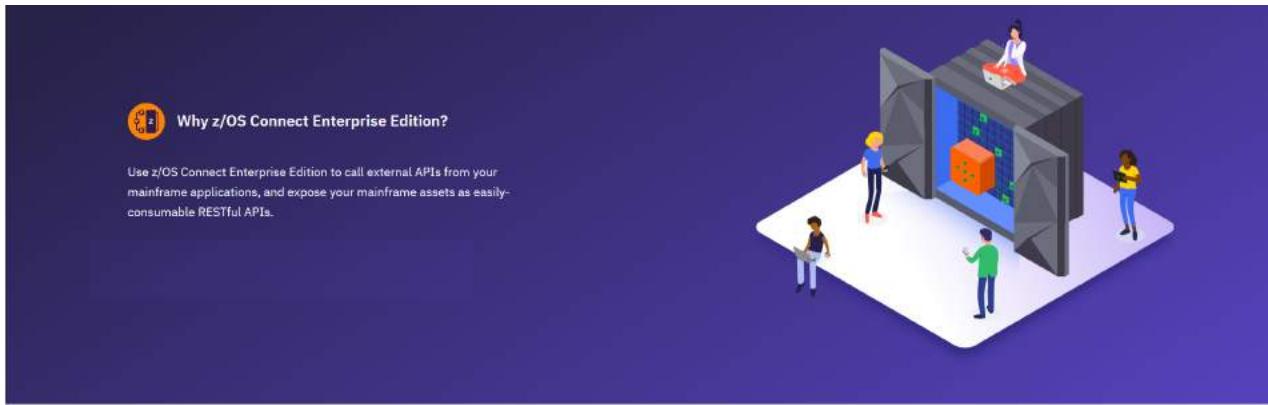
Configuring authorization levels.....	381
API provider audit.....	392
API provider security for service providers.....	393
API requester confidentiality and integrity.....	394
Configuring a TLS connection from CICS.....	397
Configuring an AT-TLS connection from IMS or a z/OS application.....	397
Configuring a TLS connection with RACF key rings.....	400
API requester authentication and identification.....	407
API requester authentication options.....	407
Identity assertion for API requesters.....	441
API requester authorization.....	446
zosConnectAccess role with a SAF user registry.....	449
Authorization levels with a SAF user registry.....	450
User registries.....	453
Configuring a basic user registry.....	453
Configuring an LDAP user registry.....	455
Activating and configuring the SAF user registry.....	456
Using a Trust Authentication Interceptor (TAI) to allow selected unauthenticated requests.....	459
Security capabilities using z/OS Connect EE.....	459
Transport Layer Security (TLS).....	459
Application Transparent Transport Layer Security (AT-TLS).....	460
Java Secure Sockets Extension (JSSE).....	460
Keystores and truststores.....	461
Cipher suites.....	462
Hardware cryptography.....	463
SP800-131a.....	464
Configuring the Liberty Angel process and z/OS authorized services.....	465
Configuring named angels.....	468
Chapter 10. Operating.....	471
Starting and stopping z/OS Connect EE.....	471
System automation.....	472
The MODIFY command.....	473
Chapter 11. Monitoring.....	475
Using SMF records to monitor requests.....	475
API provider data from SMF type 123.1 V2 records.....	476
Request data from SMF type 123.1 V1 records.....	481
Sample JCL to format SMF records.....	484
Chapter 12. Exposing z/OS assets as REST APIs.....	487
Creating services.....	487
Creating a CICS, IMS, or IBM MQ service.....	488
Creating an IMS database service.....	543
Db2 services.....	547
Creating a REST client service.....	555
Creating a WOLA service.....	556
Generating service archives for DevOps.....	560
How to move services between environments for DevOps.....	562
Invoking a service.....	563
Creating APIs.....	564
How to use the API editor.....	564
RESTful web services and API design.....	570
Designing RESTful APIs.....	572
Defining your APIs with the z/OS Connect EE API toolkit.....	579
Creating a REST API.....	589
Deploying an API in the API toolkit.....	595

Examining, testing, starting and stopping an API.....	596
Editing an existing API.....	599
Re-importing changed services.....	600
Generating an API archive in the API toolkit.....	600
Developing applications with Swagger documents.....	601
Documenting your API using Swagger.....	601
Generating API archives for DevOps.....	602
Configuring the API toolkit.....	603
Connecting to a z/OS Connect EE server.....	603
Configuring client certificates for server connections.....	604
Setting preferences for the API toolkit.....	606
Client application development.....	606
Retrieving service information.....	606
Conversion for data transformation.....	606
Error responses.....	615
Chapter 21. Calling RESTful APIs from z/OS applications.....	617
Generating artifacts for an API requester by using the build toolkit.....	617
Generating artifacts for an API requester from the command line.....	618
Generating artifacts for an API requester with the build toolkit SDK.....	620
Naming conventions for the API requester artifacts.....	621
The build toolkit properties file.....	621
JSON schemas supported by the build toolkit.....	627
Establishing a connection from z/OS Connect EE to the request endpoint.....	628
Deploying the API requester to z/OS Connect EE.....	631
Developing z/OS application to call APIs.....	632
Sample: Specifying values for arrays and strings.....	636
Securing your z/OS application calls to APIs.....	637
Overriding the URIMAP in a CICS application.....	643
Linking by dynamic calls.....	645
Error handling for API requester calls.....	648
Testing the z/OS application for API calls.....	653
Chapter 22. How to manage services.....	655
Administering services with the administration interface.....	655
Get a list of services.....	656
Get details of a service.....	657
Get the status of a service.....	659
Get the request schema of a service.....	660
Get the response schema of a service.....	661
Get the statistics of a service.....	661
Get the statistics for multiple services.....	662
Deploy a service.....	663
Set the initial status of a service.....	666
Change the status of a service.....	667
Update a service.....	668
Delete a service.....	669
Administering service archives.....	670
Automated service archive management.....	670
Overriding service archive properties.....	671
Chapter 23. How to manage APIs.....	673
Automated API management.....	673
Using the API Deployment utility.....	673
Deploying an API.....	674
Undeploying an API.....	674
Using the RESTful administration interface.....	675

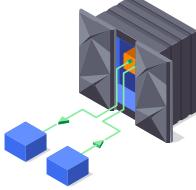
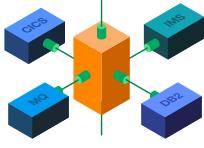
GET swagger documents.....	676
GET a list of APIs.....	677
GET details of an API.....	678
Deploying an API.....	679
Update an API.....	680
Change the status of an API.....	682
Delete an API.....	683
Administering z/OS Connect EE policies.....	683
Defining a rule set.....	684
Configuring z/OS Connect EE policies.....	690
Enabling z/OS Connect EE rule sets and policies.....	691
Chapter 24. How to manage API requesters.....	693
How to deploy an API requester automatically.....	693
How to use the RESTful administration interface to manage API requesters.....	694
How to get a list of API requesters.....	694
How to get details of an API requester.....	695
How to change the status of an API requester.....	696
How to deploy an API requester using the administration interface.....	697
How to update an API requester.....	698
How to delete an API requester.....	699
Chapter 25. Extending z/OS Connect EE.....	701
Creating a z/OS Connect EE service provider.....	701
Creating a z/OS Connect EE service at run time.....	701
Creating an interceptor.....	702
Intercepting API provider calls.....	702
Intercepting API requester calls.....	704
Creating a monitoring interceptor.....	706
Creating a data transformer.....	713
Using the DataFormExt interface.....	714
Chapter 26. Resolving problems.....	715
Enabling trace in z/OS Connect EE.....	715
Enabling trace in communications stubs.....	716
Contacting IBM Software Support.....	717
Prepare z/OS Connect EE for FFDC.....	718
Resolving startup problems.....	721
Resolving problems with security.....	722
Resolving performance problems.....	722
Resolving Java memory problems.....	722
Chapter 27. Messages.....	725
Communication stub (CICS) messages.....	725
Communication stub (IMS and z/OS) messages.....	732
GMOM messages - API toolkit data transformation diagnostic.....	738
GMOx messages - IMS service provider (IMS service provider).....	741
GMOBA messages.....	741
GMOCR messages.....	743
GMOIG messages.....	744
GMOPR messages.....	747
GMOSR messages.....	748
GMOTM messages.....	749
Chapter 28. Reference.....	751
Capabilities compatibility.....	751
z/OS Connect EE features.....	752

Configuration elements.....	753
Command reference.....	792
apideploy command syntax.....	792
MVS system MODIFY commands.....	793
MODIFY command syntax.....	794
zconbt command syntax.....	795
zosconnect command syntax.....	796
zconsetup command syntax.....	797
Creating a CICS service for C or top-down.....	798
Creating a channel description document.....	809
Sample for creating a CICS service with a properties file.....	810
Creating an IBM MQ service by using a properties file.....	813
Create a two-way MQ service with a properties file.....	819
Building service archive using the build toolkit SDK.....	822
Sample JCL.....	823
HTTP response code reference.....	824
Supplied server templates.....	825
Legal Notices.....	827
Notices.....	831
Glossary.....	835
Index.....	841

Chapter 1. Why z/OS Connect Enterprise Edition?



Truly RESTful APIs to and from your mainframe

 <p>APIs to and from the mainframe Expose your applications and data through RESTful APIs, or call external APIs from your mainframe applications.</p>	 <p>Comprehensive support z/OS® Connect Enterprise Edition supports CICS, IMS, DB2, MQ, as well as other WOLA subsystems.</p>
---	---

API enable your favorite z/OS subsystems

 <p>CICS</p>	 <p>Db2</p>
 <p>IMS</p>	 <p>IMS database</p>

Feature overview

Quick and easy API creation

Our API toolkit is designed to help you create APIs using RESTful best practices.

Avoid unnecessary changes to your applications

Transformation between native language structures and JSON is handled for you, so you don't need to modify existing code.

Visual Service-to-API mapping

Build APIs on top of your services using an intuitive, point-and-click interface.

Integrate with your existing z/OS security setup

Manage and secure your APIs on the world's most secure platform.

Rapid API design iteration

Make your API available in seconds, with right-click deployment from the developer environment.

Test APIs from within Eclipse

Use the built-in Swagger UI to try out your APIs.

DevOps-friendly artefacts

APIs and Services are represented as single files, so you can manage them as release artefacts in your pipeline.

OpenAPI (Swagger 2.0) Specification Support

OpenAPI specification support means that your APIs can be easily imported into API management solutions, such as IBM API Connect.

Try z/OS Connect EE now

<u>IBM Z software trials</u> No installation required Get a free trial environment, with tutorials showing how to rapidly expose CICS and IMS applications as RESTful APIs.	<u>Try out the latest features</u> Test out the newest features through our better product.
---	--

Chapter 2. What is z/OS Connect Enterprise Edition?

Cloud and mobile applications reshape the way enterprises and systems interact. RESTful APIs that use JSON message formats are the predominant standards for new application development. IBM z/OS Connect Enterprise Edition and IBM z/OS Connect Enterprise Edition Unlimited provide a framework that enables z/OS-based programs and data to participate fully in the new API economy for mobile and cloud applications.

Note: Where *z/OS Connect EE* is used in this documentation it is intended to mean either *z/OS Connect Enterprise Edition* or *z/OS Connect Enterprise Edition Unlimited*.

z/OS Connect EE provides RESTful API access to *z/OS* applications and data hosted in subsystems such as CICS, IMS, IBM MQ, and Db2. The framework provides concurrent access, through a common interface, to multiple *z/OS* subsystems.

z/OS Connect EE also provides the capability for CICS, IMS, and *z/OS* applications to access any RESTful endpoint, inside or outside *z/OS*, through RESTful APIs with JSON formatted messages.

z/OS Connect EE is built on the WebSphere® Application Server for *z/OS* Liberty profile and relies on many of its core capabilities for essential functions. For more information, see “[WebSphere Application Server Liberty profile](#)” on page 5.

z/OS Connect EE can help to deliver benefits for an enterprise in the following ways:

- Mobile and cloud application developers, inside or outside the enterprise, can incorporate *z/OS* data and transactions into their applications without the need to understand *z/OS* subsystems. The *z/OS* resources appear as any other RESTful API.
- *z/OS* application programmers can take advantage of published RESTful APIs to incorporate service and data into their business applications.

z/OS Connect Enterprise Edition Unlimited

z/OS Connect Enterprise Edition Unlimited provides an alternative pricing metric to *z/OS Connect Enterprise Edition* and includes the same features as *z/OS Connect Enterprise Edition*.

The number of server instances of *z/OS Connect Enterprise Edition* is restricted by the license. There is no limitation on the number of server instances of *z/OS Connect Enterprise Edition Unlimited*.

This documentation describes the features and capabilities of both *z/OS Connect Enterprise Edition Unlimited* and *z/OS Connect Enterprise Edition* unless stated otherwise. For more information, see “[Differences between *z/OS Connect Enterprise Edition* and *z/OS Connect Enterprise Edition Unlimited*](#)” on page 7.

Maintenance fixes for *z/OS Connect Enterprise Edition Unlimited* are delivered at the same time as fixes for *z/OS Connect Enterprise Edition* and can be applied by the same method as for any *z/OS* product.

z/OS Connect EE as API provider

When *z/OS* resources are exposed as RESTful APIs for mobile and cloud applications to consume, *z/OS Connect EE* acts as a RESTful API provider.

Using the *z/OS Connect EE* API toolkit, an intuitive, workstation-based tool, a developer, with or without *z/OS* skills, can easily create RESTful APIs from traditional *z/OS*-based assets. These RESTful APIs are then deployed in the *z/OS Connect EE* server, available for mobile and cloud applications to access.

The following diagram shows how *z/OS Connect EE* connects mobile and cloud applications with *z/OS* assets.

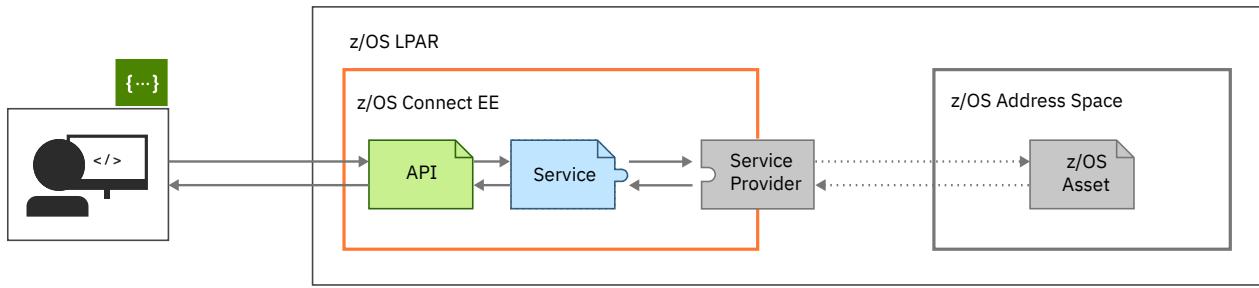


Figure 1. An illustration of z/OS Connect EE.

z/OS Connect EE as API requester

With z/OS Connect EE, z/OS applications can consume RESTful APIs, thus using the underlying services and data from external hosts (also called request endpoints). In this scenario, z/OS Connect EE acts as a RESTful API requester.

An intermediary named API requester is required to facilitate RESTful API calls by z/OS-based programs.

The following diagram shows how z/OS Connect EE connects z/OS applications with services and data available from a request endpoint.

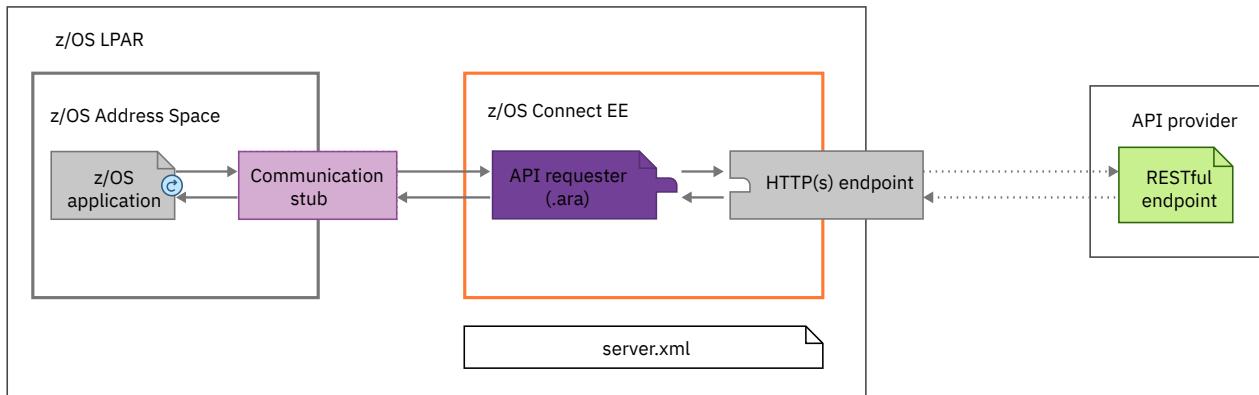


Figure 2. An illustration of z/OS Connect EE.

Features and benefits of using z/OS Connect EE

- Provides a centralized framework to enable z/OS assets to be exposed as RESTful APIs and to call RESTful APIs. The framework provides you a quick and easy way to deploy, manage and audit services, APIs and API requesters.
 - Provides functions that are based on Liberty server technology. It is lightweight and easily configurable.
- Note:** Do not attempt to run your own applications in the same Liberty server where z/OS Connect EE is running. Such usage is not supported.
- Provides z/OS differentiation with System Authorization Facility (SAF) security, and z/OS Workload Manager (WLM) integration. WLM integration means different URIs can have varying levels of priority and performance criteria.
 - The deployment of API archive file or the API requester archive can be done manually or it can be automated by extending an existing enterprise change control process. The associated deployment capabilities provide an enterprise with the ability to meet the demands of the fast-paced API economy.

For z/OS Connect EE as API provider

- Provides a standard framework to support service providers that you can write yourself, or can be supplied with the System of Record (SoR). The service provider forwards requests to the SoR.

- Provides a unified, extensible approach for discovery and invocation of RESTful APIs and services for z/OS-based business assets, opening up these assets to cloud and mobile-based Systems of Engagement (SoE) environments.
- Can uniformly track requests from cloud, mobile, and web environments by using z/OS System Management Facility (SMF) services. This tracking means that z/OS customers can use their existing processes for auditing and chargeback for requests from these environments.
- Can automatically convert the request payload from JSON form on input to binary form. The binary form is consumable by z/OS applications such as COBOL, PL/I, and C. For the response from the z/OS application, the output can be converted from binary.
- Provides a deployment artifact called an API archive file, which can be deployed to the z/OS Connect EE run time as a z/OS-based RESTful API. These RESTful API definitions, which are created by the supplied z/OS Connect EE API toolkit, include Swagger 2.0 standard descriptions. The descriptions instruct other application developers how to use the API. z/OS assets appear to mobile and cloud developers as any other system that provides RESTful API services. An enterprise API catalog can be populated with z/OS Connect EE-hosted APIs alongside any other provider and can be used in the same way.
- Provides security for individual or groups of z/OS Connect EE APIs and services with SAF security. Only specified users or groups have access to specified services.
- Provides a mapping capability that is used before the transformation from JSON messages to the required format of the z/OS subsystems. This mapping capability adds a powerful abstraction layer between the API consumer and the underlying z/OS assets and allows inline manipulation of requests, such as the mapping of HTTP headers, pass-through, redaction, or defaulting of JSON fields.

The API mapping functions improve the productivity of the z/OS developer in the creation of RESTful APIs.

For z/OS Connect EE as API requester

- Provides a unified approach to discovering published RESTful APIs and incorporating them into z/OS subsystems, taking advantage of the data and services from cloud, mobile-based applications and even from other z/OS subsystems.
- Provides a build toolkit to generate artifacts for the API requester based on API description files, including Swagger 2.0. These artifacts include an API requester archive (.ara) file to be deployed to the z/OS Connect EE server, API information file and data structures that you use in your z/OS application program for calling REST APIs. z/OS application developers can easily establish the communication between z/OS subsystems and RESTful endpoints through z/OS Connect EE and request data and service inside or outside z/OS without understanding Swagger 2.0.
- Can automatically convert the request payload from binary format on input to JSON format. The binary format is consumable by z/OS applications such as COBOL and PL/I. For the response from the RESTful endpoint, the output can be converted from JSON.

WebSphere Application Server Liberty profile

WebSphere Application Server Liberty profile is a component-based runtime for Java™ web and enterprise applications.

The z/OS Connect EE product is built on the WebSphere Application Server for z/OS Liberty profile and relies on many of its core capabilities for essential functions. For example, z/OS Connect EE server configuration is based on the same style of XML files used by Liberty, and can use many of the security features provided by the Liberty. For this reason, this documentation includes links to the *WebSphere Application Server for z/OS Liberty* documentation when a relevant topic is applicable to z/OS Connect EE.

Note:

The installation of Liberty that is included with z/OS Connect EE is dedicated to running z/OS Connect servers, and cannot be used to host any other applications. Therefore, not all of the facilities described in the Liberty documentation are applicable for use with a z/OS Connect EE solution.

For more information, see [Liberty overview](#) in the *WebSphere Application Server for z/OS Liberty* documentation.

z/OS Connect EE concepts

Understand the terms used in z/OS Connect EE before you get started with z/OS Connect EE.

Service providers

A z/OS Connect EE service provider forwards requests to a System of Record (SoR).

The following service providers are included with z/OS Connect EE:

- CICS
- IMS
- IMS Database
- IBM MQ
- REST client
- WOLA; this service provider can be used by both WOLA-enabled applications and CICS (to support z/OS Connect EE V2 configurations).

Alternatively you can use a service provider that is supplied with the SoR to plug into the framework, or you can write your own. All service providers must implement the `com.ibm.zosconnect.spi.Service SPI`.

Interceptors

One or more interceptors can be configured to be invoked either globally or for specific API, API requester or direct service calls.

You can use z/OS Connect EE interceptors to either monitor requests or control whether the request is processed. Interceptors cannot change requests. Interceptors are provided by z/OS Connect EE, third-party vendors, or you can write your own.

Interceptors are written and delivered by any component to plug into the framework. The framework allows for any number of qualities of service to be injected around the invocation of a service. For more information on using the interceptors see [“Configuring interceptors” on page 288](#).

z/OS Connect EE provides three interceptors:

- An *audit interceptor* is used to write SMF records. For more information, see [“Using SMF records to monitor requests” on page 475](#).
- An *authorization interceptor* performs authorization checks to control users' access to resources. For more information, see [“API provider authorization” on page 373](#).
- A *file system logger interceptor* logs request information to a file. For more information, see [“Configuring the file system logger interceptor” on page 290](#).

Data transformation providers

z/OS Connect EE provides the ability to optionally transform request and response payloads that are used for calling a business asset on operating systems. Data transformers are written and delivered by any component to plug into the framework. A data transformer is included with z/OS Connect EE: `com.ibm.zosconnect.xform.service`. It provides JSON to and from byte arrays that are consumable by COBOL, PL/I, and C programs on z/OS.

Using z/OS Connect EE for message payload transformations

IBM z/OS Connect EE provides the ability to optionally do a conversion of the request and response payloads that are used for calling a business asset on z/OS.

Payload transformers can be created to satisfy specific needs by implementing the required DataXform SPI provided by z/OS Connect EE. Payload conversion is enabled by adding a reference in the configuration to a data transformation implementation.

z/OS Connect EE provides an implementation that requires that the request and response message format be JSON. It supports the conversion of the request to a byte array, which can be mapped by a COBOL, PL/I, or C structure. The target program's structure, or copybook (description of its parameters in/out) is used to generate a binding file and JSON schema files by using a utility that is provided with z/OS Connect EE. The bind file that is generated by this utility is used by z/OS Connect EE to do the data conversion between JSON and native data formats as requests arrive and responses are returned. The JSON schemas for the request and response message can be retrieved with a provided REST API call.

Process types on z/OS

For the Liberty runtime environment on the z/OS platform, there are two types of process: the server process and the Angel process.

Before WebSphere Application Server 16.0.0.4, only one instance of the Angel process per LPAR was supported. If you have multiple products installed that require the Angel process, you must run the Angel process with the highest level of Liberty profile code that is shipped with any of those products to ensure you have the latest function.

For more information, see [Process types on z/OS](#) in the WebSphere Application Server documentation.

Differences between z/OS Connect Enterprise Edition and z/OS Connect Enterprise Edition Unlimited

z/OS Connect Enterprise Edition Unlimited provides an alternative pricing metric to z/OS Connect Enterprise Edition.

Both products share the same code, the same runtime operations and the same support process. The differences between the two products are shown in table [Table 1 on page 7](#).

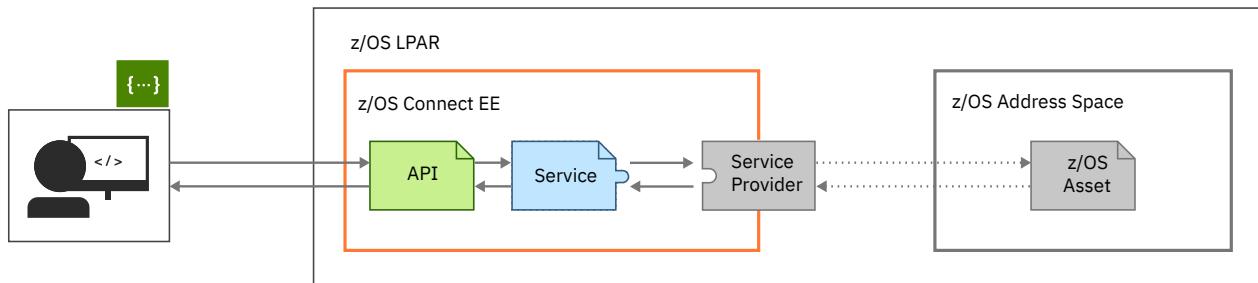
Table 1. Product differences. Comparison between z/OS Connect Enterprise Edition and z/OS Connect Enterprise Edition Unlimited		
	z/OS Connect Enterprise Edition	z/OS Connect Enterprise Edition Unlimited
Number of possible server instances	Restricted by license	Unrestricted
Product FMIDs	HZC3000 JZC3002	HZC3000 JZC3002 JZC3003
Message log title	For example: product = WAS FOR Z/OS 19.0.0.9, z/OS Connect 03.00.26 (wlp-1.0.32.c1190920190905-0148)	For example: product = WAS FOR Z/OS 19.0.0.9, z/OS Connect EE Unlimited 03.00.26 (wlp-1.0.32.c1190920190905-0148)
Message log registration message	For example: CWWKB0108I: IBM Corp product z/OS Connect version 03.00.26 successfully registered with z/OS.	For example: CWWKB0108I: IBM Corp product z/OS Connect EE Unlimited version 03.00.26 successfully registered with z/OS.
SMF record for registration	SMF record type 89 subtypes 1 and 2.	SMF record type 89 subtype 1.

The example text in table [Table 1 on page 7](#) includes version values, but versions will change as maintenance is applied.

For more information about the Program IDs, Component IDs, and FMIDs for these products, see the relevant Program Directory listed in [“Installing z/OS Connect EE” on page 196](#)

z/OS assets as REST APIs

To access and act on a resource on a z/OS subsystem, create a service that defines how the JSON schemas for the request and response messages map to the resource. Then design a REST API to define how an HTTP action such as GET, PUT, POST or DELETE would act on the service.



Services and service archive

A service in z/OS Connect EE is used by a REST API to act on a z/OS resource through connections and data transformation functions provided through a service provider. Information about the service is contained in a service archive (.sar) file and includes information about the request and response JSON schemas required by the service.

REST APIs for accessing services

For z/OS Connect EE services, you can create REST APIs that define how an HTTP action such as GET, PUT, POST or DELETE would act on the services. Information about the REST APIs for a service is contained in an API archive (.aar) file that can be deployed to z/OS Connect EE.

Service and API creation workflow

Examine the following general workflow and gather the requirements for resources that need to be exposed.

1. Generate a service archive.
 - Depending on the z/OS subsystem, you can use the z/OS Connect EE API toolkit, the build toolkit, or subsystem-specific utilities to create your service and generate a service archive.
 - Determine the resources that need to be exposed in the API. Expose all fields that might be needed for creating the REST API.
2. Design and create the API by using the z/OS Connect EE API toolkit.
 - A Swagger document is generated, along with the HTTP-to-JSON mapping and other API- and service-related information.
 - If necessary, edit the generated Swagger document in the API toolkit. Do so before you export your API project into an API archive.
3. Deploy the API to a connected server. Within the API toolkit, right-click the API project and selecting the corresponding menu item.
4. As a software source code management best practice, save a copy of the API project in your source code management system (SCM).
5. Examine and test your API in the editor by using the embedded Swagger UI.
6. Generate client code (such as downloading the open source code from Swagger, or importing the Swagger document into IBM API Connect).

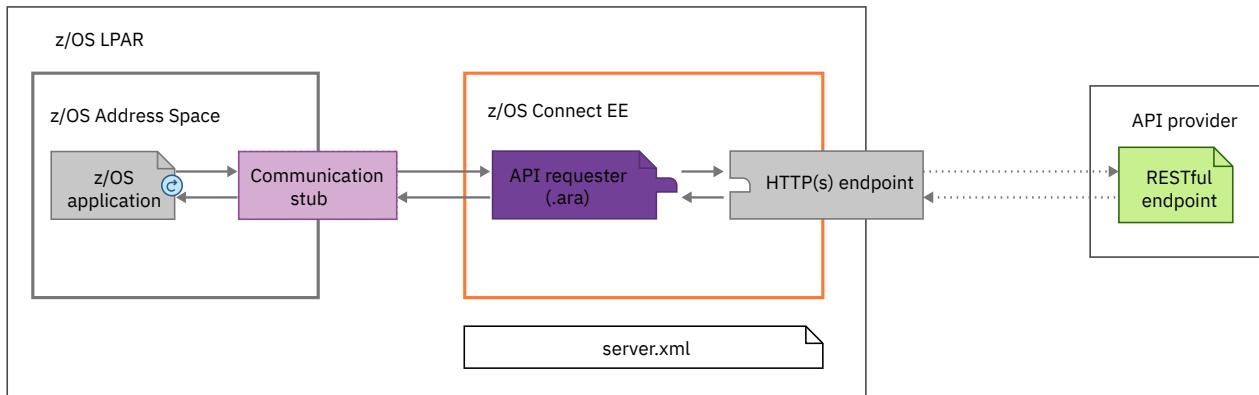
Related concepts

[“Creating a CICS, IMS or IBM MQ service” on page 488](#)

You can create a service for CICS, IMS, or IBM MQ by using the z/OS Connect EE API toolkit.

[z/OS applications to call REST APIs](#)

Before you plan to create your z/OS applications to call REST APIs through z/OS Connect EE, review the workflow for facilitating API calls by z/OS applications and gather the requirements you should meet.



To enable z/OS applications to call REST APIs, you must have an intermediary called API requester created and deployed in z/OS Connect EE to facilitate such API calls. You need a number of artifacts for an API requester:

- API requester archive (**.ara**): a file that is needed by the z/OS Connect EE to perform data mapping and transformation for REST API calls by z/OS applications.
- API information file: A file that contains information of an API that z/OS applications want to call.
- Data structures: request and response data structures for each operation in an API.

These artifacts are generated from REST API description files, for example, Swagger files. In addition, you should also establish connections between your z subsystem with the z/OS Connect EE server and between the z/OS Connect EE server and the request endpoints. The request endpoint is a REST endpoint that is called by z/OS applications through z/OS Connect.

The following procedure gives you an overview of this workflow:

1. Discover the REST API description file.

Discover the description file for the REST API that your z/OS application plans to call. Swagger files in JSON or YAML format are supported.

2. Generate artifacts for the API requester.

Based on the API description file discovered in Step 1, use the build toolkit to generate the artifacts for the API requester. The artifacts include an API requester archive (**.ara**) file to be deployed to the z/OS Connect EE server, and API information file and data structures that you use in your z/OS application program for calling REST APIs.

For more information about generating artifacts for the API requester, see [“Using the build toolkit to generate artifacts for an API requester” on page 617](#).

3. Configure z subsystems to support API calls by your z/OS applications.

To route API call requests issued by the z/OS application from the z subsystem to the z/OS Connect EE server, you must configure the z subsystem to communicate with the z/OS Connect EE server. For CICS, follow the instruction in [“Configure CICS to access z/OS Connect EE to call APIs” on page 306](#). For IMS, follow the instruction in [“Configuring IMS to access z/OS Connect EE for API calls” on page 309](#). For other z/OS applications, follow the instruction in [“Configuring other z/OS applications to access z/OS Connect EE for API calls” on page 312](#).

4. Establish the connection from the z/OS Connect EE server to the request endpoint.

To route the API call request issued by the z/OS application from the z/OS Connect EE server to the request endpoint, you must configure the z/OS Connect EE to establish a connection with the request

endpoint. For more information, see [“Establishing a connection from z/OS Connect EE to the request endpoint” on page 628](#).

5. Deploy the API requester archive to the z/OS Connect EE server.

You can deploy the API requester archive (.ara) file directly by copying the .ara file to the location where your API requesters are stored. For more information about deploying the API requester, see [“Deploying the API requester to z/OS Connect EE” on page 631](#).

6. Develop your z/OS application program to call a REST API by using the generated API information file and data structures.

To transfer request data from the z/OS application to the request endpoint, your z/OS application program should include the generated API information file. For more information about calling an API from your z/OS application, see [“Developing a z/OS application to call APIs” on page 632](#).

7. Examine and test your z/OS application program.

The z/OS Connect EE API toolkit

The z/OS Connect EE API toolkit is an Eclipse-based workstation tool that you install into IBM Explorer for z/OS to create services and REST APIs for accessing z/OS resources.

In the API toolkit, you can create two types of projects: service projects and API projects.

Service projects and service creation

You can create a CICS, IMS, Db2, or IBM MQ service in a z/OS Connect EE service project.

For Db2, the provided service project editor guides you through the steps to import a Db2 native REST service definition from Db2.

For CICS, IMS, and IBM MQ, the provided service project editor and service interface editor guide you through the steps to import COBOL copybooks, PL/I include files, or full programs and to customize the service interface. The service interface defines what fields to expose and how they are exposed in the service.

- Fields and data structures can be renamed and remarks can be added, allowing API creators to better understand and use the underlying services.
- Fields can be excluded from the interface.
- Data structures can be duplicated and the order can be adjusted (order is important for IMS segments).
- Complex data structures such as REDEFINES and OCCURS DEPENDING ON clauses for COBOL and REFER in PL/I are supported.

Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select						
Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start
✓ phonebookRequest						
✓ Segment 1						
✓ IVTNO_INPUT_MSG						
IN_LL	<input type="checkbox"/>	IN_LL		SHORT	2	1
IN_ZZ	<input type="checkbox"/>	IN_ZZ		SHORT	2	3
IN_TRANCDE	<input checked="" type="checkbox"/>	IN_TRANCDE		CHAR	10	5
IN_COMMAND	<input checked="" type="checkbox"/>	IN_COMMAND		CHAR	8	15
IN_LAST_NAME	<input checked="" type="checkbox"/>	IN_LAST_NAME		CHAR	10	23
IN_FIRST_NAME	<input checked="" type="checkbox"/>	IN_FIRST_NAME		CHAR	10	33
IN_EXTENSION	<input checked="" type="checkbox"/>	IN_EXTENSION		CHAR	10	43
IN_ZIP_CODE	<input checked="" type="checkbox"/>	IN_ZIP_CODE		CHAR	7	53
✓ Copy_Of_IVTNO_INPUT_MSG						
IN_LL	<input type="checkbox"/>			SHORT	2	1
IN_ZZ	<input type="checkbox"/>			SHORT	2	3
IN_TRANCDE	<input type="checkbox"/>			CHAR	10	5
IN_COMMAND	<input type="checkbox"/>			CHAR	8	15
IN_LAST_NAME	<input type="checkbox"/>			CHAR	10	23
IN_FIRST_NAME	<input type="checkbox"/>			CHAR	10	33
IN_EXTENSION	<input type="checkbox"/>			CHAR	10	43
IN_ZIP_CODE	<input type="checkbox"/>			CHAR	7	53
✓ IVTNO_OUTPUT_MSG						
OUT_LL	<input type="checkbox"/>	OUT_LL		SHORT	2	1
OUT_ZZ	<input type="checkbox"/>	OUT_ZZ		SHORT	2	3
OUT_MESSAGE	<input checked="" type="checkbox"/>	OUT_MESSAGE		CHAR	40	5

Figure 3. Defining the service interface

Defined services can be exported as a service archive for deployment.

API projects and API creation

You can create REST APIs from any service archives in a z/OS Connect EE API project. An API editor and a mapping editor are included to help you model and create the APIs for your z/OS Connect EE services.

- Create your API by specifying the path, query, and header parameters in the API editor.

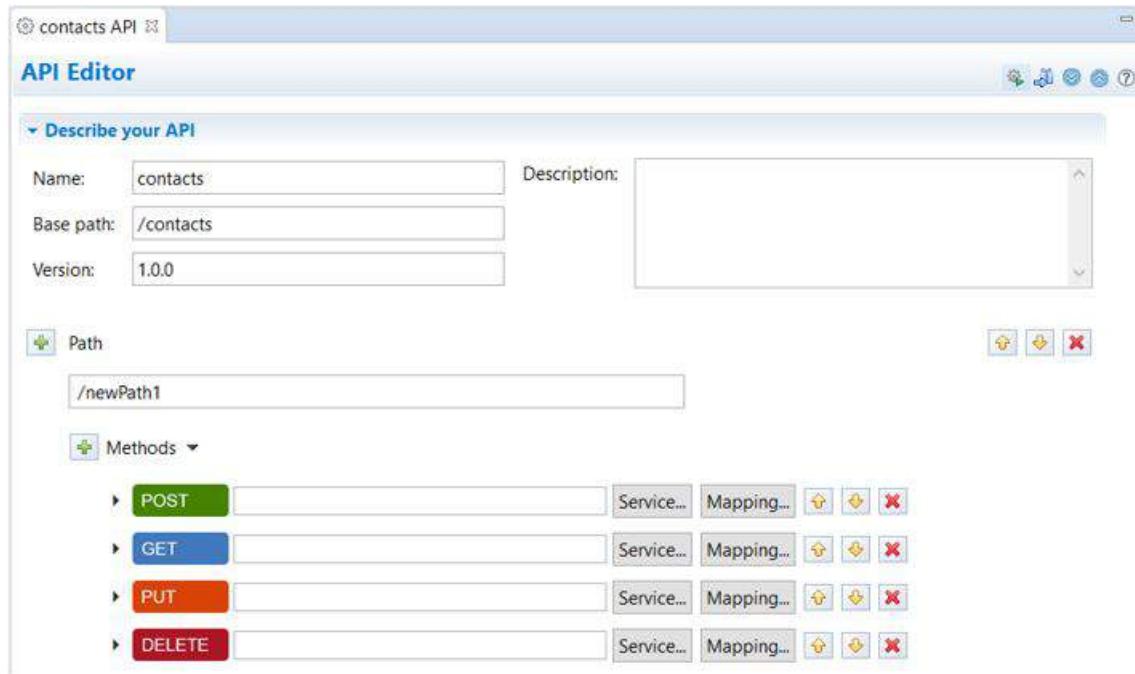


Figure 4. API editor

- Define how the path, query, and header parameters map to fields in the JavaScript Object Notation (JSON) schema of your service.

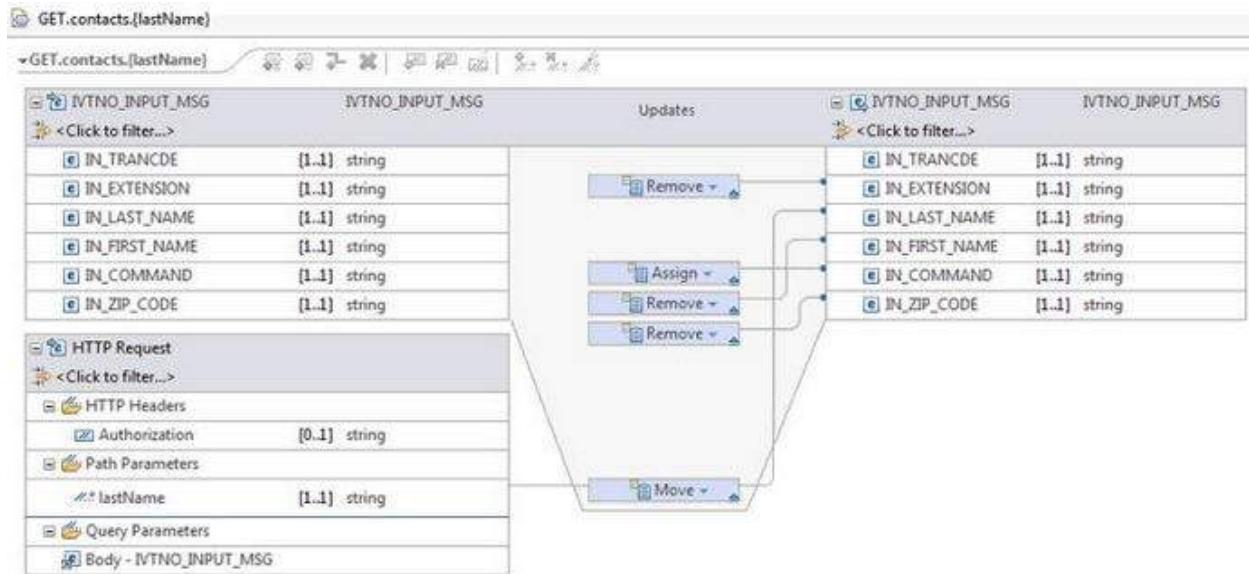


Figure 5. Mapping editor

- View services on connected servers.
- Deploy APIs to the servers.
- Visually inspect the APIs in the embedded Swagger UI.
- Start and stop the APIs.
- Immediately test the APIs directly from the editor.

Related concepts

["Restrictions on the data structure import function" on page 488](#)

Depending on the z/OS subsystem and types of programs, restrictions on the size and number of data structures apply. The data structure import function is supported on specific platforms.

["Creating a CICS, IMS or IBM MQ service" on page 488](#)

You can create a service for CICS, IMS, or IBM MQ by using the z/OS Connect EE API toolkit.

[“Creating APIs” on page 564](#)

z/OS Connect EE provides the tools you need to create APIs for your z/OS Connect EE services.

Related tasks

[“Installing z/OS Explorer and the z/OS Connect EE API toolkit” on page 201](#)

To design and create APIs you need to use the z/OS Connect EE API toolkit that is provided as a plug-in for IBM Explorer for z/OS Aqua.

The z/OS Connect EE build toolkit

The build toolkit is supplied with z/OS Connect EE for generating artifacts that z/OS needs to participate in the RESTful API economy as an API provider, as an API consumer, or both.

When z/OS Connect EE acts as an API provider

The build toolkit is used to generate service archive (.sar) files from z/OS assets. The service archive files enable APIs to connect to back-end applications on z/OS subsystems.

The build toolkit can also be used to generate API archive (.aar) files from API project directories that are created in z/OS Connect EE API toolkit.

When z/OS Connect EE acts as an API requester

The build toolkit is used to generate API requester archive (.ara) files from API description files. The API requester is the function that facilitates RESTful API calls by z/OS applications.

The build toolkit is the key to your DevOps process

Enterprises need a DevOps process to support agile development, testing, and deployment of services, APIs, and API requesters for continuous delivery. The build toolkit is the key to such an automated process.

In a DevOps pipeline, the source files for defining a service, API, and API requester are treated as the source code and stored in a source control management (SCM) system. A build automation script then generates the required archive files from the latest source files in the SCM by using the build toolkit. For more information, see [“DevOps with z/OS Connect EE” on page 14](#).

How the build toolkit is supplied

The build toolkit is supplied in a compressed file called `zconbt.zip` that you can download from the product installation directory. When you install a z/OS Connect EE server instance, you automatically have access to the `zconbt.zip` file. The compressed file contains both the command line interface (CLI) and Software Development Kit (SDK) versions of **zconbt**. A readme file that contains instructions on how to use **zconbt** is included in the package.

Related concepts

[“Using the build toolkit to generate artifacts for an API requester” on page 617](#)

You can use the build toolkit either from the command line interface or with the SDK.

Related tasks

[“Generating service archives for DevOps” on page 560](#)

Use the build toolkit **zconbt** command to generate the service archive from service projects that are created in the API toolkit or from a properties file that defines the service.

[“Installing the z/OS Connect EE build toolkit” on page 199](#)

Install the z/OS Connect EE build toolkit to create service archive (.sar) files or artifacts for an API requester.

DevOps with z/OS Connect EE

Identify a DevOps process before you start your development to automate the development and deployment of services, APIs, and API requesters for continuous integration and delivery.

Services, APIs, and API requesters in z/OS Connect EE are represented as single files (.sar, .aar, and .ara archive files). When these archive files are placed in the designated locations for services, APIs, and API requesters, they can be automatically deployed if the server is configured to monitor changes in these locations. You can run the build toolkit through a script to generate these archive files and then deploy them to the designated locations to automate the deployment process. Deployment of services and APIs can also be achieved through calls to the administration interface.

The build toolkit supports generation of service archives and API archives from projects created in the z/OS Connect EE API toolkit. For each service or API that you create in the API toolkit, the project directory in the Eclipse workspace and the files it contains serve as the input to the build toolkit for archive generation.

Enterprise DevOps workflow

An enterprise DevOps workflow typically involves automated processes. For example:

- When a service project or API project is created or modified in the API toolkit, the project files in the Eclipse workspace are checked into a source control management (SCM) system.
- Changes to the SCM trigger a script in a build environment, such as Jenkins, that runs the build toolkit to generate the archive file. Or a DevOps tool, such as IBM UrbanCode Deploy, is configured to pull the SCM for updates and subsequently run the build toolkit to generate the archive file.
- The generated archive file is then automatically deployed by using REST calls or by copying the files to the designated directories for automated deployment.

The following diagram demonstrates how the build toolkit supports the DevOps workflow by taking defined properties files or project directories from the API toolkit to create the service archive, API archive, and API requester archive files.

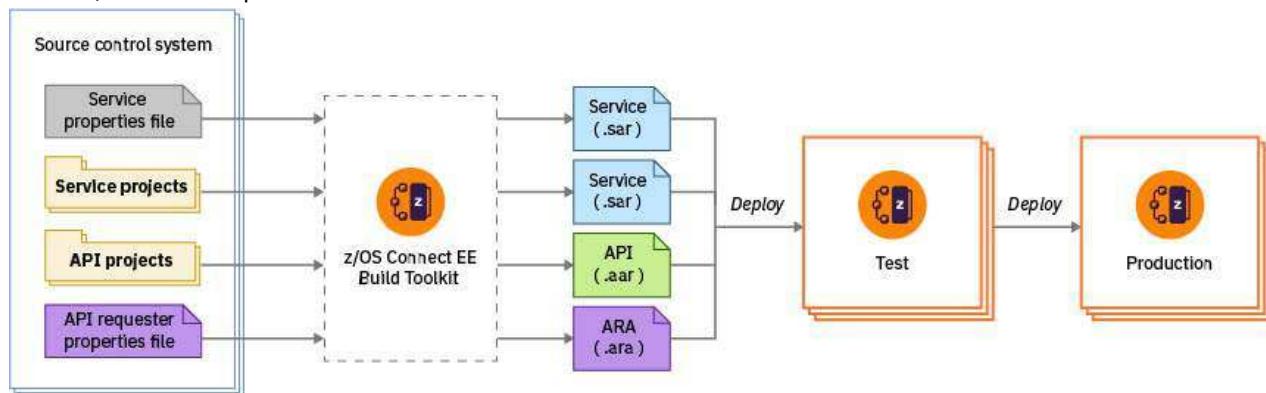


Figure 6. DevOps support with the build toolkit

A DevOps pipeline includes the following key characteristics:

- Properties files and API toolkit project files for generating the archive files are treated as source code.
- Changes to the source code are managed by an SCM.
- The build toolkit is used to generate the archive files as part of build automation scripts.
- Generated archive files are not stored in the SCM, but in a location that the deployment automation script can access.

The ability to export services and APIs as archive files and deploy the archive files to a test server from within the API toolkit is used during development time. When development work is done, the related project files are stored in an SCM. In the test and production environments, automated scripts build the archive files by running the build toolkit against the latest source code from the SCM. Generated archive files are stored in an artifact repository for deployment orchestration. The following diagram demonstrates this DevOps pattern:

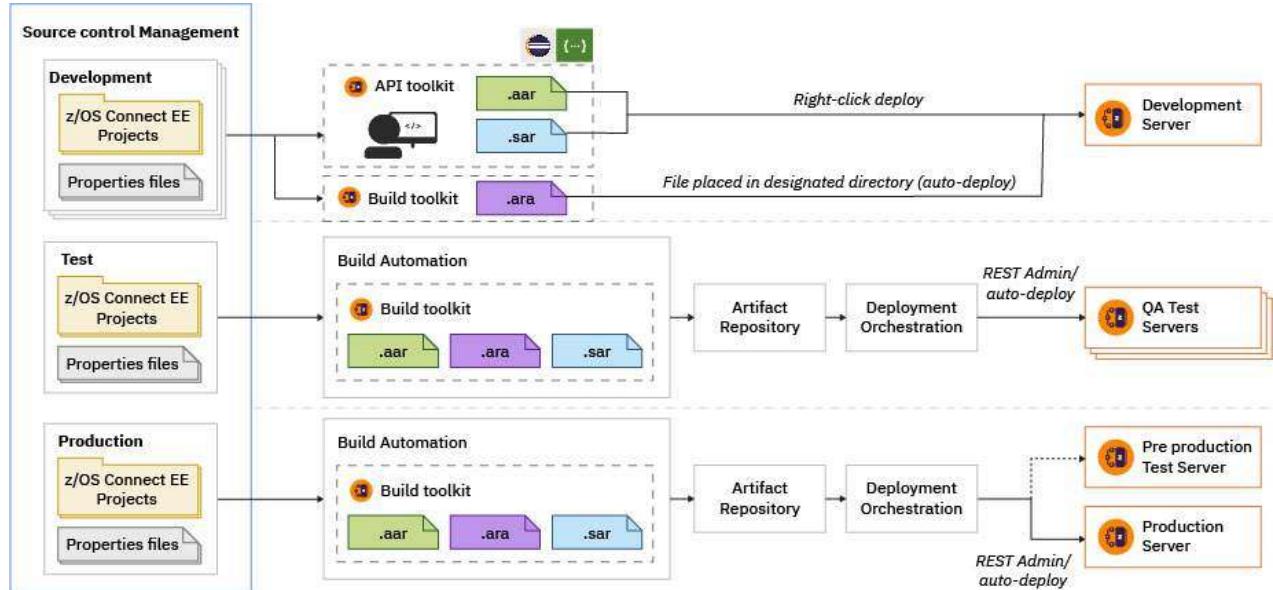


Figure 7. DevOps pattern and the build toolkit

Sample DevOps technologies

The following diagram demonstrates the technologies that are used in a sample DevOps solution. In this sample, Git or IBM Rational Team Concert source control management component (commonly referred to as Jazz® SCM) is used to store the source code. Jenkins is used for build automation that calls the build toolkit to generate the archive files and store them in JFrog Artifactory. Deployment orchestration scripts are set up in IBM UrbanCode Deploy to deploy the archive files to the server by either using the RESTful administration interface or copying the files to the designated locations.

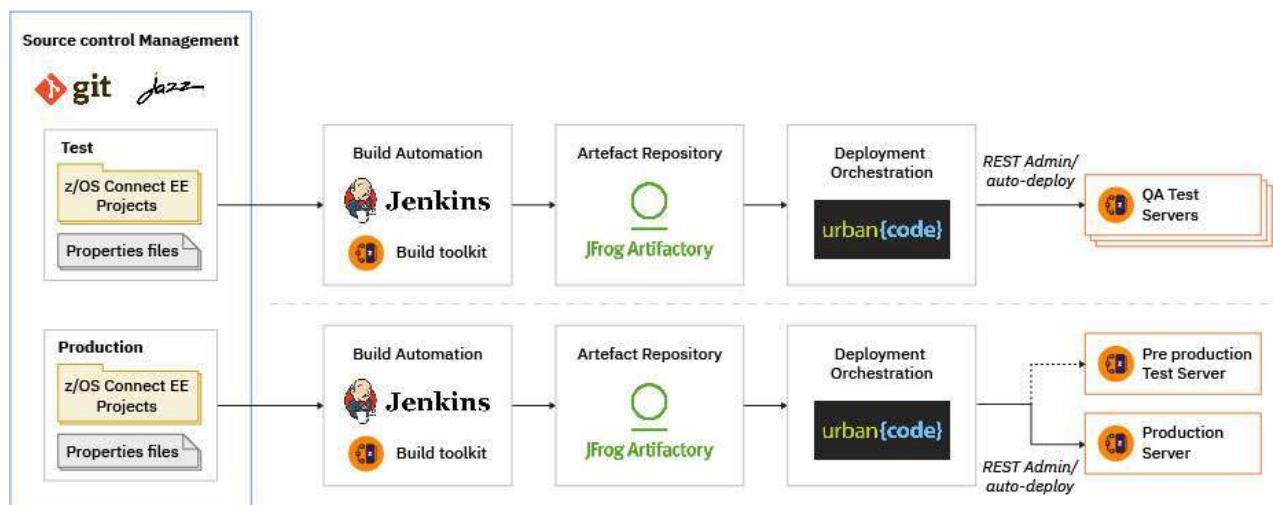


Figure 8. A sample DevOps solution

For a sample DevOps workflow for z/OS Connect EE, see [Sample workflow with IBM UrbanCode Deploy and Git](#).

The build toolkit

[“The z/OS Connect EE build toolkit” on page 13](#)

The build toolkit is supplied with z/OS Connect EE for generating artifacts that z/OS needs to participate in the RESTful API economy as an API provider, as an API consumer, or both.

[“zconbt command syntax” on page 795](#)

The **zconbt** command starts the build toolkit tool. You can use the build toolkit to generate archive files for services, APIs or API requesters.

[Deploying a service](#)

[“Automated service archive management” on page 670](#)

Deploy and undeploy your service archives automatically when you add or remove them from the services directory.

[“Generating service archives for DevOps” on page 560](#)

Use the build toolkit **zconbt** command to generate the service archive from service projects that are created in the API toolkit or from a properties file that defines the service.

[Deploying a service \(by using the administration interface\)](#)

[Deploying an API](#)

[“Automated API management” on page 673](#)

Deploy and undeploy your APIs automatically when you add or remove them from the API directory.

[“Generating API archives for DevOps” on page 602](#)

Use the build toolkit **zconbt** command to generate the API archive from API projects that are created in the API toolkit.

[Deploying an API \(by using the administration interface\)](#)

[Deploying an API requester](#)

[“Deploying the API requester to z/OS Connect EE” on page 631](#)

Deploy and undeploy your API requesters automatically when you add or remove API requester archive (.ara) files from the API requester folder.

Chapter 3. z/OS Connect Enterprise Edition change history

Use this information to discover the enhancements in each refresh of IBM z/OS Connect Enterprise Edition.

Note:

1. As stated in the license agreement, IBM provides fixes only for the current release and the two previous releases.
2. Best practice is to apply the latest release as soon as it is available, but some releases are the minimum required to resolve specific issues. Similarly, statically-linked API requester programs should be relinked to pick up new function and fixes.
3. This documentation describes the latest release of z/OS Connect EE V3.0. If you have an earlier release, some of the features and functions described in this documentation may not be available.
4. Before applying the latest release, also check the list of functions that are to be removed. See the table in [“Removal notices” on page 53](#).

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

Maintenance level	Enhancements and fixes
March 2021	
V3.0.43 (APAR PH34379)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• The z/OS Connect EE API requester is enhanced so you can generate a JWT, which contains the z/OS application asserted user ID as the "sub" (Subject) claim. For more information, see “Calling an API secured with a locally generated JWT” on page 431. <p>Fixes</p> <ul style="list-style-type: none">• PH25575 - The z/OS Connect EE runtime and build toolkit cannot support alternative MIME types defined in a Swagger document.• PH26053 - A <code>NullPointerException</code> is reported when non-JSON strings are configured in the <code>tokenRequest requestBody</code> attribute.• PH30400 - z/OS Connect EE incorrectly transforms some numbers with a lot of digits after the decimal point.• PH32251 - z/OS Connect EE reports BAQR1150E: An internal error occurred during the HTTP mapping. when running a new API.• PH31224 - A <code>java.lang.UnsatisfiedLinkError</code> is reported when using PassTickets with both IMS and Db2 in a z/OS Connect EE server.



Attention: Users of the IMS database service provider who specify the location of their UTC jars with a `resourceAdapter` element in `server.xml`. The `resourceAdapter` element must be replaced by the `imsDbUtcPaths` attribute of the `zosconnect_dbServices` element. For more information, see [“Configuring user type converter support for IMS database services” on page 262](#).

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
API toolkit V3.0.9.1 and V3.2.9.1	z/OS Connect EE V3.0.43 is compatible with API toolkit V3.0.9.1 and V3.2.9.1.

February 2021

V3.0.42 (APAR PH32853)	Enhancements
Server code update	<ul style="list-style-type: none">The IBM MQ service provider is enhanced to write the following fields to the Data SPI: TIME_SOR_SENT, TIME_SOR_RECV, SOR_IDENTIFIER, SOR_REFERENCE, and SOR_RESOURCE. This data can be written to SMF 123.1 records and consumed by monitoring interceptors.The version of Liberty that is embedded in z/OS Connect EE is upgraded to V21.0.0.1. For more information, see What is new in this release of Liberty in the <i>WebSphere Application Server for z/OS Liberty</i> documentation.
	<p> Attention: WebSphere Liberty Profile 21.0.0.1 includes the fix for APAR PH34376. This fix corrects the processing of the realm derived from JWT which is used in subsequent RACMAP processing. For more information, see “How to configure JWT authentication” on page 366 and APAR PH34376.</p>

Fixes

- PH31935 - Message BAQR7095E is reported when installing an API which was built using the z/OS Connect EE build toolkit on z/OS.
- PH33129 - Running ws-schemagen.jar with z/OS Connect EE configuration files produces messages CWWKG0073W and CWWKG0030E.
- PH33240 - Message BAQR7085E reports that capabilities are not supported for the z/OS Connect EE MQ service provider.

January 2021

V3.0.41 (APAR PH32637)	Enhancements
Server code update	<ul style="list-style-type: none">The RESTful service administration interface is updated to version 1.2.0 and adds version information to services for all service providers. This new version also adds connectionRef, transid, transidUsage, and ccsid information for services that use the CICS service provider. For more information, see “Administering services with the administration interface” on page 655.
	Fixes
	<ul style="list-style-type: none">PH30899 - Using JWTs with z/OS Connect EE API requesters reports an UnsupportedEncodingException with runtimeCodePage=1208.
API toolkit V3.0.9.0 and V3.2.9.0	Fixes
	<ul style="list-style-type: none">PH32226 - An incomplete or empty response is received for a service after re-importing a data structure in the z/OS Connect EE API toolkit.

December 2020

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.40 (APAR PH30597)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• The IMS service provider has been enhanced to enable Policy override of the interaction properties <code>imsInteractionTimeout</code>, <code>imsLtermOverrideName</code>, and <code>imsTranExpiration</code>. To learn more about these new policy overrides, see "Valid properties for use in policy rules" on page 687.• z/OS Connect EE is enhanced so a policy rule can be configured to perform an action when a specific header is present on an API request, regardless of the value of that header. For more information, see example 5 in "Format of a rule set" on page 685.
API toolkit V3.0.8.9 and V3.2.8.9	<p>Fixes</p> <ul style="list-style-type: none">• PH30073 - Swagger UI does not prompt for z/OS Connect EE credentials with CORS request when testing POST method. <p>Note:</p> <p>The "Try It Out" function causes a cross-origin (CORS) request to be made from the workstation to the z/OS Connect EE server. Swagger UI is changed to use browser credentials in CORS requests if an Authorization header value is not manually specified. This change allows the browser to prompt for credentials for CORS requests if the browser supports it and <code>allowCredentials="true"</code> is configured in the <code>cors</code> element of the <code>server.xml</code> configuration file.</p> <p>Due to limitations of the Eclipse internal web browser, configure Eclipse to use an external web browser. Click Preferences > General > Web Browser > Use external browser. The following browsers correctly prompt for both Basic Authentication and Client Authentication credentials with the "Try It Out" function: Internet Explorer 12+, Edge (Windows), Chrome, (Windows, macOS, Linux®), Firefox (Windows, macOS, Linux).</p>
November 2020	
V3.0.39 (APAR PH30278)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• z/OS Connect EE policies can now alter the following IBM MQ service properties: <code>mqConnectionFactory</code>, <code>mqDestination</code>, and <code>mqReplyDestination</code>. For more information, see "Valid properties for use in policy rules" on page 687.• New interceptor interfaces are added that enable monitoring support for API requester. Request data is available on the request origin and the target endpoint. Timestamps enable the calculation of the z/OS Connect EE processing time, the time spent connecting to the endpoint, and the time taken to retrieve access tokens. Monitoring of inflight requests and requests failing early is also enabled. Interceptor writers can take advantage of these new capabilities by implementing the <code>InterceptorRequester</code> interface and optionally the <code>EndpointInterceptor</code> and <code>EarlyFailureInterceptorRequester</code> interfaces. For more information, see "Intercepting API requester calls" on page 704. <p>Fixes</p> <ul style="list-style-type: none">• PH29643 - Message BAQR0407W is written after z/OS Connect EE is updated to version 3.0.30.0 or later.

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
API toolkit V3.0.8.8 and V3.2.8.8	Fixes <ul style="list-style-type: none">• PH31252 - The z/OS Connect EE API toolkit does not remove multi-dimensional array/occurs fields from the response.
October 2020	
V3.0.38 (APAR PH29265)	Enhancements Server code update <ul style="list-style-type: none">• The z/OS Connect EE API toolkit and server runtime have been enhanced to support COBOL and PL/I data structures for IMS messages longer than 32 K that have LL, ZZ, and TRANCODE fields in the request and LL and ZZ fields in the response data structures. To learn more about these options, see “Service-level data conversion customization” on page 537.• The z/OS Connect EE API requester CICS communication stub is updated to generate CICS application user trace, which can be enabled by using the trace facilities of the CICS region in which the stub runs. This trace capability is useful for problem determination. For more information, see “Enabling trace in communications stubs” on page 716. Fixes <ul style="list-style-type: none">• PH15844 - When HTTP request header Transfer-Encoding: chunked is specified, request data is not passed to the SOR.• PH25870 - A CICS service remains in stopped state after it is updated by the z/OS Connect EE RESTful administration PUT method.• PH26128 - z/OS Connect EE incorrectly logs message BAQR7004E when installing an API.• PH26260 - z/OS Connect EE reports messages BAQR7033E and SRVE8025E when redeploying an IMS service.
API toolkit V3.0.8.7 and V3.2.8.7	Enhancements <ul style="list-style-type: none">• The z/OS Connect EE API toolkit and server runtime are enhanced to support COBOL and PL/I data structures for IMS messages longer than 32 K that have LL, ZZ, and TRANCODE fields in the request and LL and ZZ fields in the response data structures. To learn more about these options, see “Service-level data conversion customization” on page 537.
September 2020	
V3.0.37 (APAR PH28518)	Enhancements Server code update <ul style="list-style-type: none">• When a CICS application is developed to call a RESTful service, the z/OS Connect EE server that is used to process the request can be chosen at run time within the CICS application. This option allows for splitting of workloads between servers, for example, based on business area. This dynamic routing capability is available for CICS applications only. For more information, see “Overriding the URIMAP in a CICS application” on page 643• The REST client service provider that is supplied with z/OS Connect EE to connect to an HTTP endpoint such as a Db2 native REST service, has been enhanced to provide improved performance.

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
API toolkit V3.0.8.6 and V3.2.8.6	Fixes <ul style="list-style-type: none">• PH27287 - z/OS Connect EE reports a COBOL parsing error caused by a J2C COBOL importer defect.
August 2020	
V3.0.36 (APAR PH27162)	Enhancements Server code update <ul style="list-style-type: none">• z/OS Connect EE policies can now alter the following Db2 service properties db2ConnectionRef, and db2CollectionID. For more information, see “Valid properties for use in policy rules” on page 687. Fixes <ul style="list-style-type: none">• PH15594 z/OS Connect EE and IMS Connect encounter issues when a session connected via a persistent socket drops or is canceled.• PH26697 The z/OS Connect EE IMS database service provider causes a NullPointerException in method sqlResourceAdapter.• PH26836 The z/OS Connect EE build toolkit reports a NullPointerException while processing the security field of a swagger document.• PH25101 An ephemeral port listening on 127.0.0.1 is opened when a z/OS Connect EE server is started. <p> Attention: Installing the PTF for this APAR will change the default behavior of z/OS Connect EE to not open the Liberty command listener port. If required, the Liberty command listener port can be specified in the server's bootstrap.properties. See APAR PH25101 for details.</p>
API toolkit V3.0.8.5 and V3.2.8.5	Fixes <ul style="list-style-type: none">• PH26129 - The z/OS Connect EE API toolkit does not copy business descriptions specified for fields to the generated Swagger file.
July 2020	

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.35 (APAR PH26291)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">The text of messages BAQR0417W and BAQR0418W has been updated. For more information, see z/OS Connect EE Runtime Messages. <p>Fixes</p> <ul style="list-style-type: none">PH21761 A CICS region reports SOS DFHSM0133 WBSEBUF when z/OS Connect EE requester is in use.PH25345 Passing user credentials in the request body to the authentication server to obtain a JWT causes a NPE in z/OS Connect EE.PH21819 z/OS Connect EE sets some CORS headers automatically. <p> Attention: When this fix is applied, additional CORS configuration is required in <code>server.xml</code> to enable connections from the z/OS Connect EE API toolkit and JavaScript clients. For more information, see “Configuring Cross-Origin Resource Sharing on a z/OS Connect Enterprise Edition Server” on page 300</p>
API toolkit	<p>Fixes</p>
V3.0.8.4 and V3.2.8.4	<ul style="list-style-type: none">PH25185 The Authorization header is not present in the <code>swagger.json</code> when there is a slash before the query parameters.PH25451 Inconsistent swagger is generated by the z/OS Connect EE API toolkit when creating a multi-path API.
<hr/> <p>June 2020</p>	
V3.0.34 (APAR PH25369)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">z/OS Connect EE has been enhanced with the capability to expose service fields as <i>OpenAPI 2.0 Specification</i> compliant dates. Specifying host date patterns defines how date values are represented in system of record (SOR) fields, and enables conversion between SOR date values and <i>OpenAPI 2.0</i> compliant date values. For more information, see “Defining Date fields” on page 503.The version of Liberty that is embedded in z/OS Connect EE is upgraded to V20.0.0.6. For more information, see What is new in this release of Liberty in the <i>WebSphere Application Server for z/OS Liberty</i> documentation. <p>Fixes</p> <ul style="list-style-type: none">PH10474 Generating API requesters for PL/I and COBOL using the same swagger results in duplicate API name.PH17588 z/OS Connect EE reports an FFDC event when processing an API requester status code that has no mapping defined.
API toolkit	<p>Enhancements</p>
V3.0.8.3 and V3.2.8.3	<ul style="list-style-type: none">The API toolkit has been enhanced to support the specification of API contact information in the generated <i>OpenAPI 2.0</i> documentation. For more information, see “Defining your APIs with the z/OS Connect EE API toolkit” on page 579.
<hr/> <p>May 2020</p>	

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.33 (APAR PH24143)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• z/OS Connect EE has been enhanced to support the use of PassTicket authentication for IMS services. Using PassTickets allows for z/OS Connect EE to enable SAF authentication for IMS without access to the SAF password. For more information, see “Configuring PassTicket support for IMS services” on page 241. <p>Fixes</p> <ul style="list-style-type: none">• PH15192 The IMS service provider causes message CWYBS0497E The resource bundle locale en_us cannot be loaded when processing DFS error.• PH21070 Message BAQR0558E is reported when a message greater than 4KB in length is sent to Db2.• PH21577 There is inconsistent behavior between deploying z/OS Connect EE .aar or .ara files and .sar files in an HA environment.• PH21965 Timeouts are reported when multiple requests are sent via the z/OS Connect EE IMS service provider Keyring.• PH23181 NullPointerExceptions are reported with z/OS Connect EE IMS service provider when pinging IMS Connect.• PH23267 Message BAQR1143E Internal server error reported when changing the status of API requesters.• PH24239 A NullPointerException is reported when an incorrect PSB name is supplied with the IMS DB service provider.• PH24563 A NullPointerException is reported in the IMS database service provider setResourceFactoryRef method.• PH24608 The build timestamp of the IMS service provider is not reported correctly.• PH24677 The z/OS Connect EE build toolkit zconbt reports different version numbers depending on how it is run.

April 2020

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.32 (APAR PH23299)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• The IBM MQ service provider with z/OS Connect EE is enhanced to provide advanced data transformation options for IBM MQ services, and the capability to send and receive IBM MQ messages up to 100MB in length for services that were created with the API toolkit. Services created with the build toolkit are limited to a message size of 32KB. For more information, see “Create a two-way IBM MQ service” on page 96.• z/OS Connect EE is enhanced to allow users to specify a custom CCSID when creating a CICS service. To use the custom CCSID, you implement your own <code>CharsetProvider</code> and <code>Charset</code> classes to extend the Java Runtime Environment (JRE). For more information, see “How to specify a custom CCSID for a CICS service” on page 527.• The version of Liberty that is embedded in z/OS Connect EE is upgraded to V20.0.0.3. For more information, see What is new in this release of Liberty in the <i>WebSphere Application Server for z/OS Liberty</i> documentation. <p>Fixes</p> <ul style="list-style-type: none">• PH22476 The order of fields in the JSON response is not preserved by the IMS service provider.
API toolkit V3.0.8.1 and V3.2.8.1	<p>Enhancements</p> <ul style="list-style-type: none">• The z/OS Connect EE API toolkit is enhanced to support the creation of IBM MQ services in z/OS Connect EE service projects using the service project editor. For more information, see “Create a two-way IBM MQ service” on page 96.
March 2020	
V3.0.31 (APAR PH22287)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• z/OS Connect EE has been enhanced to support the creation of IMS database services. IMS database services are created in the API toolkit by specifying IMS database connection information, an IMS database name, and SQL query. For more information, see “Using the IMS database service provider” on page 257. <p>Fixes</p> <ul style="list-style-type: none">• PH22959 The <code>API_SERVICE_NAME</code> field in SMF 123 subtype 1 V1 records always contains the service name.• PH18222 When an API is deleted from the file system, the API cannot be redeployed to another z/OS Connect EE server sharing the same zFS.• PH16762 Message BAQR7004E is written when an API is updated using the RESTful administration interface.

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
API toolkit V3.0.8.0 and V3.2.8.0	<p>Enhancements</p> <ul style="list-style-type: none">The API toolkit is enhanced to support the creation of IMS database services by specifying IMS database connection information, an IMS database name, and SQL query. For more information, see Using the IMS DB Service provider. <p>Fixes</p> <ul style="list-style-type: none">PH21858 PICTURE '999V99' fields in PL/I structures are imported as CHAR fields by the z/OS Connect EE API toolkit.

February 2020

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.30 (APAR PH19977)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• You can now configure z/OS Connect EE to record SMF 123 subtype 1 version 2 records, which provide more detailed information about API provider requests. The version 2 records are not an extension of the version 1 records. Therefore, do not enable recording of version 2 records until all the products you use to process your records support this new version. For more information, see “API provider data from SMF type 123 subtype 1 version 2 records” on page 476.• Since z/OS Connect EE V3.0.16.0, a tracking token has been included on every request sent to CICS over IPIC. The tracking token is now moved from the USERCORRDATA field of the CICS task association data to the ODADPTRDATA1 field of the task association data. You must modify any CICS programs that use this field. For more information, see “Tracking selected requests” on page 710. If you are using IBM Z® APM Connect to monitor z/OS Connect EE, you must apply the IBM Z® APM Connect PTF UJ01628 before upgrading to z/OS Connect EE V3.0.30.0.• Both of the communication stubs for the z/OS Connect EE API requester are updated to set a return code, which is available to COBOL applications in the special variable RETURN-CODE. For more information, see “Error handling for API requester calls” on page 648.• To ensure that all error messages from the z/OS Connect EE API requester CICS communication stub can be accommodated in the TD queue, you must ensure that your BAQQ TD queue definition is set with the attributes RECORDSIZE=508 and BLOCKSIZE=512. The CICS BAQAPIR.CSD sample is updated accordingly.• The build toolkit is enhanced to check the presence of mandatory description fields and to detect errors in the Responses section of the Swagger document when building API requester artifacts. The enhanced build toolkit is V1.2. If you rebuild existing API requester artifacts with the new build toolkit, parsing of the Swagger document might now fail where it previously succeeded. For more information, see The z/OS Connect EE build toolkit• If you use the build toolkit in your DevOps automation, note that the text of the following messages has changed: BAQB0003, BAQB0022, BAQB0025 and message BAQB0012 has changed from type E to W. For more information, see z/OS Connect EE Build Toolkit Messages. <p>Fixes</p> <ul style="list-style-type: none">• PH04288 z/OS Connect EE API requester returns an incorrect BAQ-STATUS-MESSAGE-LEN value when using the CICS communication stub.• PH10042 BAQZCBT fails with message JVMCFRE003 bad major version.• PH10398 Message BAQB0017E Failed to process operation is issued when the "+" character is used in a property name.• PH10611 The z/OS Connect EE build toolkit does not report that the apikey is not supported in the path.• PH16570 The z/OS Connect EE build toolkit zconbt completes with code 0 even when some errors or warnings are reported.• PH16590 The z/OS Connect EE build toolkit reports "unsupported schema type:" for integer or boolean values.• PH19390 The z/OS application and IMS API requester communication stub truncates OAuth headers.• PH19824 Unexpected data is returned to an interceptor when querying HTTP headers.

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
API toolkit V3.0.7.1	Enhancements <ul style="list-style-type: none">URL and DOC links can now be copied from the properties view of APIs and services within the API toolkit so that they can be easily pasted as text.
API toolkit V3.2.7.1	Enhancements <ul style="list-style-type: none">URL and DOC links can now be copied from the properties view of APIs and services within the API toolkit so that they can be easily pasted as text.
January 2020	
V3.0.29 (APAR PH19949)	Enhancements <p>Server code update</p> <ul style="list-style-type: none">The version of Liberty that is embedded in z/OS Connect EE is upgraded to V19.0.0.12. For more information, see What is new in this release of Liberty in the <i>WebSphere Application Server for z/OS Liberty</i> documentation.The z/OS Connect EE API requester is enhanced to allow you to specify the <code>requireAuth</code> attribute on the <code>zosconnect_apiRequesters</code> element for all API requesters. For more information about the attribute, see “zosconnect_apiRequesters” on page 754. Fixes <ul style="list-style-type: none">PH15842 Message BAQR0553E does not give enough information when a <code>connectionRef</code> is defined in a SAR but the relevant connection is not.PH20674 Message CWLWKE0701E and a <code>Java.lang.NullPointerException</code> are reported in <code>ApiManagerImpl</code> after applying V3.0.28.0.
API toolkit V3.0.7.0	Enhancements <ul style="list-style-type: none">The z/OS Connect EE API toolkit is enhanced to support the creation of Db2 service projects. The service can be defined by importing a Db2 service from the Db2 service manager. For more information, see “Db2 services” on page 547. Fixes <ul style="list-style-type: none">PH17905 - The API toolkit allows an API with a base path containing only space characters to be deployed.
API toolkit V3.2.7.0	Enhancements <ul style="list-style-type: none">The z/OS Connect EE API toolkit is enhanced to support the creation of Db2 service projects. The service can be defined by importing a Db2 service from the Db2 service manager. For more information, see “Db2 services” on page 547. Fixes <ul style="list-style-type: none">PH17905 - The API toolkit allows an API with a base path containing only space characters to be deployed.
December 2019	

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.28 (APAR PH19001)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• z/OS Connect EE API requester is enhanced to support JWT caching. For more information, see “JSON Web Token (JWT) caching” on page 427.• New interceptor interfaces are added that enable monitoring of API requests between z/OS Connect EE and the system of record, and requests that fail early before interceptors are currently invoked. Interceptor writers can take advantage of these new capabilities by implementing either or both of the new interfaces <code>ServiceProviderInterceptor</code> and <code>EarlyFailureInterceptor</code>. For more information, see “Creating a monitoring interceptor” on page 706• The credentials stored in a SAF cache can now be cleared by using a new parameter on the <code>modify</code> command. SAF credentials can also be made to expire automatically by setting a new attribute on the “<code>zosconnect_authorizationInterceptor</code>” on page 758 configuration element.• z/OS Connect Enterprise Edition Unlimited, V3 delivers a new pricing metric option to clients who might want to enable the deployment of unlimited server instances for a set price. It includes the same features as z/OS Connect Enterprise Edition V3. For more information, see Announcement letter 219-467 <p>Fixes</p> <ul style="list-style-type: none">• PH02075 The z/OS Connect EE API toolkit fails to deploy AAR files containing clashing variable and fixed elements.• PH18448 BAQR1145E API endpoint connection error , followed by CEE3500S Not enough storage was available to load module.• PH18497 Performance issue with API requester ID assertion for user IDs which are in a large number of SAF groups.• PH16753 z/OS Connect EE service requests fail with message BAQR0452E after WOLA service archive file update of a COMMAREA service.

November 2019

V3.0.27 (APAR PH18365) This refresh contains maintenance fixes only.

Server code update

October 2019

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.26 (APAR PH16810)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">The version of Liberty that is embedded in z/OS Connect EE is upgraded to V19.0.0.9. For more information, see What is new in this release of Liberty in the <i>WebSphere Application Server for z/OS Liberty</i> documentation. <p>Fixes</p> <ul style="list-style-type: none">PH15486 - Policy definitions do not modify the IMS transaction code when invoking IMS large data services. <p>End of support</p> <ul style="list-style-type: none">Support for Java 7 and the IBM SDK and JRE for Java 7 is removed. For more information, see “Requirements” on page 195.
API toolkit V3.0.6.10	<p>Enhancements</p> <ul style="list-style-type: none">The z/OS Connect EE API toolkit has been enhanced with the capability to expose service fields in JSON payloads as a Boolean data type. By defining specific request and response conditions to process true or false values for a given service field, the definition of the underlying COBOL or PL/I application field remains unchanged. For more information, see “Defining service fields” on page 497. <p>Note: z/OS Connect EE Version 3.0.26 or later is required.</p> <p>Fixes</p> <ul style="list-style-type: none">PH15899 - A negative time error is reported when importing a SAR file and then setting the Windows locale to Thai.
API toolkit V3.2.6.10	<p>Enhancements</p> <ul style="list-style-type: none">The z/OS Connect EE API toolkit has been enhanced with the capability to expose service fields in JSON payloads as a Boolean data type. By defining specific request and response conditions to process true or false values for a given service field, the definition of the underlying COBOL or PL/I application field remains unchanged. For more information, see “Defining service fields” on page 497. <p>Note: z/OS Connect EE Version 3.0.26 or later is required.</p> <p>Fixes</p> <ul style="list-style-type: none">PH15899 - A negative time error is reported when importing a SAR file and then setting the Windows locale to Thai.

September 2019

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.25 (APAR PH15511)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• In API requester, z/OS Connect EE is enhanced to call an API that is secured with a JSON Web Token (JWT). For more information, see “Calling an API secured with a JSON Web Token (JWT)” on page 426.• In API requester, the maximum size of the JSON string field is increased from 32,767 to 16,777,214 bytes. For more information, see the description of the characterVaryingLimit property in “The build toolkit properties file” on page 621. <p>Fixes</p> <ul style="list-style-type: none">• PH00006 - Message DFHPI9730E is misleading when a field greater than 32KB is created using the z/OS Connect EE build toolkit.PH15332 - z/OS Connect EE transformation array counter functionality thread safety issue can cause incorrect values.PH16284 - z/OS Connect EE API requester receives 403-Authorization error when invoking an API configuration to support identity assertion.
API toolkit V3.0.6.9	<p>Fixes</p> <ul style="list-style-type: none">• PH15332 - When processing multiple concurrent requests using the array counter functionality that was introduced in z/OS Connect EE V3.0.23, they could interact causing incorrect values to be used. z/OS Connect EE transformation array counter functionality thread safety issue can cause incorrect values.
API toolkit V3.2.6.9	<p>Fixes</p> <ul style="list-style-type: none">• PH15332 - When processing multiple concurrent requests using the array counter functionality that was introduced in z/OS Connect EE V3.0.23, they could interact causing incorrect values to be used. z/OS Connect EE transformation array counter functionality thread safety issue can cause incorrect values.

August 2019

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.24 (APAR PH14597)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• z/OS Connect EE supports identity assertion for API requesters. With the identity assertion support, you can invoke an API requester from a z/OS application with an identity that is provided in the application context. For more information, see “Identity assertion for API requesters” on page 441.• If you use an interceptor to monitor requests, the Data Interface in the z/OS Connect EE SPI is updated to provide system of record information for the REST and WOLA service providers. In addition, the SAF mapped user identity associated with a request can now be obtained, if one exists. For more information, see com.ibm.zosconnect.spi (z/OS Connect SPI) in the <i>Reference</i> section.• The REST client service provider now sends the User-Agent HTTP header on requests to identify the client as z/OS Connect EE with the version number. <p>Fixes</p> <ul style="list-style-type: none">• PH14220 - Using z/OS Connect EE policies to alter the IMS transaction code causes IBM OMEGAMON® for JVM on z/OS to report incorrect data.• PH13803 - An API requester program receives message BAQI0005E after applying the PTF for APAR OA56234.• PH11435 - Message BAQR0409W does not identify the resource that could not be accessed.

July 2019

V3.0.23 (APAR PH13344)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• Customers using API requester can now call the BAQTERM function from their IMS applications to close and clear the cached connection used by z/OS Connect EE. <p> Attention: If you use API requester and cached connections with IMS, you must add a call in your IMS application to close and clear the cached connection. Failing to do this will lead to memory not being reusable until the IMS process terminates.</p> <ul style="list-style-type: none">• The version of Liberty that is embedded in z/OS Connect EE is upgraded to V19.0.0.6. For more information, see What is new in this release of Liberty in the <i>WebSphere Application Server for z/OS Liberty</i> documentation. <p>Fixes</p> <ul style="list-style-type: none">• PH09124 - Class loading issue for com.ibm.jsse2 when enabling an AT-TLS connection for the IMS service provider.• PH09275 - z/OS Connect EE does not correctly apply updates to zosConnectAPI sub elements.• PH11409 - An API or service request with a Accept-Encoding:gzip,deflate header fails with HTTP response code 500 and message BAQR0429W.• PH12685 - A monitoring interceptor throws a NullPointerException for an unauthorized z/OS Connect EE user ID.
------------------------	---

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes						
API toolkit V3.0.6.7	<p>Enhancements</p> <ul style="list-style-type: none">The API toolkit is enhanced to allow numeric service fields to be defined as counters for fixed and variable-length arrays. These array counters are used to optimize both the necessary service data required for an API request, as well as the JSON payload issued from an API response. To learn more about array counters, see “Defining service fields” on page 497. <p>Note: Check compatibility for this feature in “Capabilities compatibility” on page 751.</p>						
API toolkit V3.2.6.7	<p>Enhancements</p> <ul style="list-style-type: none">The API toolkit is enhanced to allow numeric service fields to be defined as counters for fixed and variable-length arrays. These array counters are used to optimize both the necessary service data required for an API request, as well as the JSON payload issued from an API response. To learn more about array counters, see “Defining service fields” on page 497.						
June 2019							
V3.0.22 (APAR PH11931)	<p>Fixes</p> <table><tr><td>Server code update</td><td><ul style="list-style-type: none">PH11690 - Calling a non-existent service causes an FFDC record to be written for each service called.PH09188 - The z/OS Connect EE built toolkit creates an invalid service archive file on z/OS.PH08996 - Message DFS064 does not provide enough information to identify the cause of the problem.</td></tr><tr><td>API toolkit V3.0.6.6</td><td><ul style="list-style-type: none">PH11166 - Existing CICS and IMS data structures cannot be re-imported using the z/OS Connect EE API toolkit.PH12148 - Using Microsoft Windows Group Policies to customize the Internet Explorer UserAgent string results in an unsupported configuration for Swagger UI.</td></tr><tr><td>API toolkit V3.2.6.6</td><td><ul style="list-style-type: none">PH11166 - Existing CICS and IMS data structures cannot be re-imported using the z/OS Connect EE API toolkit.PH12148 - Using Microsoft Windows Group Policies to customize the Internet Explorer UserAgent string results in an unsupported configuration for Swagger UI.</td></tr></table>	Server code update	<ul style="list-style-type: none">PH11690 - Calling a non-existent service causes an FFDC record to be written for each service called.PH09188 - The z/OS Connect EE built toolkit creates an invalid service archive file on z/OS.PH08996 - Message DFS064 does not provide enough information to identify the cause of the problem.	API toolkit V3.0.6.6	<ul style="list-style-type: none">PH11166 - Existing CICS and IMS data structures cannot be re-imported using the z/OS Connect EE API toolkit.PH12148 - Using Microsoft Windows Group Policies to customize the Internet Explorer UserAgent string results in an unsupported configuration for Swagger UI.	API toolkit V3.2.6.6	<ul style="list-style-type: none">PH11166 - Existing CICS and IMS data structures cannot be re-imported using the z/OS Connect EE API toolkit.PH12148 - Using Microsoft Windows Group Policies to customize the Internet Explorer UserAgent string results in an unsupported configuration for Swagger UI.
Server code update	<ul style="list-style-type: none">PH11690 - Calling a non-existent service causes an FFDC record to be written for each service called.PH09188 - The z/OS Connect EE built toolkit creates an invalid service archive file on z/OS.PH08996 - Message DFS064 does not provide enough information to identify the cause of the problem.						
API toolkit V3.0.6.6	<ul style="list-style-type: none">PH11166 - Existing CICS and IMS data structures cannot be re-imported using the z/OS Connect EE API toolkit.PH12148 - Using Microsoft Windows Group Policies to customize the Internet Explorer UserAgent string results in an unsupported configuration for Swagger UI.						
API toolkit V3.2.6.6	<ul style="list-style-type: none">PH11166 - Existing CICS and IMS data structures cannot be re-imported using the z/OS Connect EE API toolkit.PH12148 - Using Microsoft Windows Group Policies to customize the Internet Explorer UserAgent string results in an unsupported configuration for Swagger UI.						
May 2019							

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.21 (APAR PH11043)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• The IBM MQ service provider is now built into the z/OS Connect EE server, so you do not need to copy it from an IBM MQ installation data set. The new IBM MQ service provider adds support for generating service archive files with the Build Toolkit. The resulting .sar files can then be deployed directly to the z/OS Connect EE server. For more information, see “Using the IBM MQ service provider” on page 266.• The Security section of the IBM Knowledge Centre has been enhanced for API requesters to provide more information about the security methods supported by z/OS Connect EE. The new content provides more conceptual information that will help you to choose the most suitable configuration for your installation. For more information, see Chapter 9, “Securing z/OS Connect EE resources,” on page 331. <p>Fixes</p> <ul style="list-style-type: none">• PH08317 - z/OS Connect EE does not provide sufficient diagnostic information for Db2 RESTful service failures.• PH08248 - When an .ara file was updated on the file system, z/OS Connect EE did not check whether the updated .ara file was a duplicate of an API requester already loaded into the z/OS Connect EE server's memory. This resulted in the API requester in the z/OS Connect EE server's memory being corrupted.• PH08140 - z/OS Connect EE build toolkit reports a "main" java.lang.StackOverflowError.• PH07028 - z/OS Connect EE API requester calls are repeated after a timeout failure.• PH05927 - The ws-schemagen.jar tool generates an incorrect schema for a z/OS Connect EE installation.
API toolkit V3.0.6.5	<p>Fixes</p> <ul style="list-style-type: none">• PH09920 - The IMS service provider does not support space delimited IMS transaction codes which are less than 8 bytes in length. A new API toolkit option, Space delimited, is added to the Service Editor to allow space delimited IMS transaction codes to be defined.• PH10952 - The Swagger UI "Try it out!" function in the API toolkit returns cached data from the browser if parameters are not changed between function calls.

April 2019

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.20 (APAR PH09950)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">A new capability is available that enables monitoring products to implement selective tracking of z/OS Connect EE requests in the System of Record that processes the request. This capability is currently limited to API provider requests that are sent to CICS and IMS systems. To take advantage of this capability, interceptor writers must implement the new <code>TrackingInterceptor</code> interface. For more information, see “Tracking selected requests” on page 710.Multiple group names can now be specified for the invoke, operations and reader authorization groups. This enhancement applies to both the global authorization attributes on the <code>zosconnect_zosConnectManager</code> element, as well as for specific authorization attributes on the <code>zosconnect_zosConnectAPIs > zosConnectAPI</code>, <code>zosconnect_services > service</code>, <code>zosconnect_zosConnectService</code> and <code>zosconnect_apiRequesters > apiRequester</code> elements. Multiple group names were already supported for the administration authorization groups. See “Configuration elements” on page 753. <p>Important: If you upgrade to 3.0.20.0 and you are currently using the Authorization interceptor with an LDAP registry, even if you do not use multiple groups, you must change the values of <code>admin</code>, <code>invoke</code>, <code>operations</code>, and <code>reader</code> group attributes to escape the commas in LDAP group names.</p> <p>Fixes</p> <ul style="list-style-type: none">PI97411 - Specifying <code>JSON-SCHEMA-CODEPAGE=UTF-8</code> corrupts the JSON schemas in generated SAR files.PH06013 - <code>globalAdminGroup</code> and <code>adminGroup</code> attributes do not work when an LDAP registry is specified.
API toolkit V3.0.6.4	<p>Fixes</p> <ul style="list-style-type: none">PH10225 - SAR files containing double byte characters cannot be imported into the API toolkit.PH09240 - Using the z/OS Connect EE API toolkit with invalid credentials causes the user ID to be revoked unexpectedly.PH07557 - The JSON <code>RESPONSESCHEMA</code> reported by DB2 service DISCOVER is incorrect.
March 2019	
V3.0.19 (APAR PH08861)	<p>Fixes</p> <p>Server code update</p> <ul style="list-style-type: none">PH08209 - Adds support for CICS 5.5 to WebSphere Optimized Local Adapters.
API toolkit V3.0.6.3	<p>Fixes</p> <ul style="list-style-type: none">PH08772 - When the API toolkit is installed into IBM Developer for z Systems (IDz), “<code>project xx not open</code>” errors are logged by the z/OS Connect EE project builders.
February 2019	

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.18 (APAR PH07350)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• Diagnostics in the IMS or z/OS applications stub is improved with the addition of new messages and trace information. For more information, see “Error handling for API requester calls” on page 648.• The Security section of this IBM Knowledge Center is extensively rewritten to provide more information about the supported security methods when z/OS Connect EE is used as an API provider. The new content provides more conceptual information and improved samples that helps you to choose the most suitable configuration for your installation. See Chapter 9, “Securing z/OS Connect EE resources,” on page 331.• When interceptors that are defined globally or specifically for APIs or services, are activated and deactivated, messages are now written to the <code>messages.log</code> file to provide verification of interceptor configuration. <p>Fixes</p> <ul style="list-style-type: none">• PH00662 - Message BAQI0004E is received in response to an API requester call from an IMS or z/OS application.• PH00704 - Unhelpful messages are issued by the z/OS Connect EE API requester communication stub for IMS and z/OS applications.• PH03346 - Failed updates to an API can cause the <code>.aar</code> file to be deleted from the z/OS Connect EE server.• PH03436 - A message "Retrieving network security from interaction spec" is written to the <code>messages.log</code> for each IMS SP request.• PH04711 - A GMOMW0005E data conversion error message is reported for a field that is defined as PIC V9(8) by z/OS Connect EE.• PH05135 - The z/OS Connect EE build toolkit, zconbt, reports BAQB0038E: Unexpected error has occurred during an <code>.aar</code> build: No space left on device.• PH08237 - A z/OS Connect EE API requester application received return code BAQ-SUCCESS when the request was not successful.• Resolution of memory problems in the z/OS Connect EE API requester communication stub for IMS and z/OS applications.
API toolkit V3.0.6.2	<p>Fixes</p> <ul style="list-style-type: none">• PH03342 - An API created with invalid duplicate paths using API toolkit earlier than V3.0.5.1 is deployed despite reported errors.• PH04314 - The z/OS Connect EE API toolkit allows the user to set a default value for the response service interface.

January 2019

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.17 (APAR PH06570)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• The version of Liberty that is embedded in z/OS Connect EE is upgraded to V18.0.0.4. For more information, see What is new in this release of Liberty in the <i>WebSphere Application Server for z/OS Liberty</i> documentation. <p>Fixes</p> <ul style="list-style-type: none">• PH00368 - The z/OS Connect EE build toolkit generates copybooks with illegal nesting levels.• PH03485 - The z/OS Connect EE build toolkit generates API archives with COBOL data structure field names that end with hyphens.• PH05307 - Message BAQR1140E is written when a z/OS Connect EE RESTful API with OAuth configured is invoked.• PH06095 - An API passes index value array fields but the program receives spaces.

December 2018

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.16 (APAR PH05647)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• The format of DateTime output of the file system logger has been corrected to give the time in 24-hour rather than 12-hour format, and the precision is extended to include milliseconds. For more information, see “Configuring the file system logger interceptor” on page 290.• The Data SPI that provides request-specific data to both interceptors and services, is enhanced to provide additional data. <p>API, service, and administration requests are now clearly distinguished. Timestamps enable the calculation of the z/OS Connect EE processing time and the time spent connecting to the system of record. Other information now available includes data about the system of record when using the CICS or IMS service provider and the HTTP response code. In addition, a unique correlator is sent on each request to CICS and IMS systems to enable monitoring products to track z/OS Connect EE requests.</p> <p>For more information, see z/OS Connect EE SPI Javadoc in the <i>Reference</i> section.</p> <ul style="list-style-type: none">• The Security Overview topic in the IBM Documentation has been rewritten to provide more information useful to solution architects and security administrators. See “Overview of z/OS Connect EE security” on page 331.• The IBM Documentation has a new section to provide information about improving the performance of z/OS Connect EE servers, APIs and services. See Chapter 6, “Performance considerations,” on page 205. <p>Fixes</p> <ul style="list-style-type: none">• PH04997 - Adding the <code>zosLocalAdapters-1.0</code> feature to the z/OS Connect EE server configuration causes message <code>CWWKE0701E</code>.• PH05806 - z/OS Connect EE service specific interceptors are not used for some administration options.• PH06538 - Messages <code>BAQR7057E</code> and <code>BAQR7033E</code> are written when deploying an API with the updated API toolkit V3.0.6.0.
API toolkit V3.0.6.1	<p>Fixes</p> <ul style="list-style-type: none">• PH07327 - Authorization interceptors do not correctly handle RACF® reader group which results in message <code>BAQR0428W</code> being written.

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
API toolkit V3.0.6.0	<p>Enhancements</p> <ul style="list-style-type: none">The API toolkit is enhanced to allow the definition of multiple response codes for an API operation. Rules that are specified for each response code are used to determine the HTTP status code, and unique response mapping, that are returned in the HTTP response. For more information, see “How to define multiple response codes” on page 593 <p>Support for the multiple response codes capability provided in this release requires V3.0.16 of the z/OS Connect EE server runtime. If you plan on upgrading the API toolkit to V3.0.6, ensure that you also upgrade the z/OS Connect EE server runtime to V3.0.16.</p> <p>Fixes</p> <ul style="list-style-type: none">PH03360 - A blank path prevents Swagger from working in the API toolkit Swagger UI.
November 2018	
V3.0.15 (APAR PH05461)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">z/OS Connect EE now supports identity propagation over REST to Db2 when you use a RACF PassTicket to propagate the SAF ID and have it reauthenticated in the System of Record (SoR). For more information, see Configuring PassTicket support for Db2 RESTful services. <p>Fixes</p> <ul style="list-style-type: none">PI98283 - z/OS Connect EE setting <code>preserveJsonObjectPayloadOrder="true"</code> is not honored for APIs.PH04287 - z/OS Connect EE API requester archives (ARAs) are not loaded and applications receive message BAQR1131E.PH04600 - z/OS Connect EE message BAQR1106E <filename> is incomplete. Missing file: api//POST/mapping.xml is issued.
October 2018	

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.14 (APAR PH03400)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• The RESTful administration interface for API requesters is enhanced to add the POST, PUT, and DELETE methods to support API requester deployment, API requester update, and API requester removal. For more information, see “How to use the RESTful administration interface to manage API requesters” on page 694.• The version of Liberty that is embedded in z/OS Connect EE is upgraded to V18.0.0.3. For more information, see What is new in this release of Liberty in the <i>WebSphere Application Server for z/OS Liberty</i> documentation.• When services and APIs are deployed, a compatibility check is made to ensure that the capabilities of the service or API are compatible with the current release of the z/OS Connect EE runtime. Messages BAQR7085E and BAQR7086E are issued if a compatibility issue is found. <p>Fixes</p> <ul style="list-style-type: none">• PI95793 - Services cannot be displayed via HTTP when requireSecure="true" is set on one of the <code>zosconnect_zosConnectService</code> definitions.• PH02634 - Message BAQR0666E is reported on startup when IPIC connections to CICS over SSL are in use.• PH02652 - Base64 encoded basic auth data is present in z/OS Connect EE trace.• PH04052 - Intermittently, no response is received when using z/OS Connect EE API requester from a CICS transaction.

September 2018

V3.0.13 (APAR PH02414)	<p>Fixes</p> <p>Server code update</p> <ul style="list-style-type: none">• PI90297 - When a SAR file causes a problem in the z/OS Connect EE SAR loader, the loader stops, preventing further attempts to load SAR files. As a result, services cannot be displayed by the z/OS Connect EE API toolkit nor can they be started. <p>Technotes</p> <ul style="list-style-type: none">• Liberty V18.0.0.2 is only compatible with the Angel process included with Liberty V18.0.0.2. For more information, see Compatibility issue with Liberty V18.0.0.2 and the Angel process.
------------------------	---

August 2018

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.12 (APAR PH01052)	<p>Fixes</p> <p>Server code update</p> <ul style="list-style-type: none">• PI99540 - A CICS service provider service does not return a response if the CICS container is deleted and added again.• PI92644 - z/OS Connect EE reports a <code>NullPointerException</code> on SAR update when multiple <code>zosconnect_services</code> elements are defined. <p>Technotes</p> <ul style="list-style-type: none">• Liberty V18.0.0.2 is only compatible with the Angel process included with Liberty V18.0.0.2. For more information, see Compatibility issue with Liberty V18.0.0.2 and the Angel process.
API toolkit V3.0.5.3	<p>Fixes</p> <ul style="list-style-type: none">• PI93123 - The payload on a request is corrupted because z/OS Connect EE generates an incorrect length field used in the schema.

July 2018

V3.0.11 (APAR PI99563)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• z/OS Connect EE policies can now be applied to specific APIs. You define the policy to use in the <code>zosconnectAPI</code> element of the configuration file. For more information, see "Configuring z/OS Connect EE policies" on page 690• Server templates now use <code>setUTF8ResponseEncoding="true"</code> on the server configuration file element <code>zosconnect_zosConnectManager</code> to set the response character encoding to UTF-8 for newly created server instances. The default setting has not changed, so existing servers are not affected.• The version of Liberty that is embedded in z/OS Connect EE is upgraded to V18.0.0.2. For more information, see What is new in this release of Liberty in the WebSphere Application Server for z/OS Liberty documentation. <p>Note: Liberty V18.0.0.2 is only compatible with the Angel process included with Liberty V18.0.0.2. For more information, see Compatibility issue with Liberty V18.0.0.2 and the Angel process.</p> <p>Fixes</p> <ul style="list-style-type: none">• PH00403 After many requests are made to BAQCSTUB, message BAQI0902E is issued to indicate insufficient storage.
------------------------	---

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
API toolkit V3.0.5.2	Fixes <ul style="list-style-type: none">• Hover help in service interface editor does not highlight the behavior and precedence of field value.• Service and API project wizards do not check that an API or service name is POSIX portable. Service names, API names, AAR file names, and SAR files names are now all checked to be POSIX portable to prevent z/OS UNIX file system errors.• Minimum array occurrences for a response are not enforced. A new advanced option Enforce minimum array occurrence is added in the Service Editor to enable validation of minimum array occurrences for response interfaces.• Meaningful error messages are not provided on copybook import for service project. When importing source that contains COBOL or PL/I data structures, the UI now displays detailed information on structures that were omitted due to having unsupported data types.• z/OS Connect actions still show for both API and Service options in right-click menu after right-clicking to close a service or API project in Project Explorer. No z/OS Connect actions appears on closed projects now.• Exporting an API with no services in methods still creates a .aar file on disk.
June 2018	
V3.0.10 (APAR PI98232)	Enhancements
Server code update	<ul style="list-style-type: none">• The communication stub for IMS and other non-CICS z/OS applications is enhanced to support persistent connections to the z/OS Connect EE server. For more information, see “Configuring IMS to access z/OS Connect EE for API calls” on page 309 and “Configuring other z/OS applications to access z/OS Connect EE for API calls” on page 312.• z/OS Connect EE policies can now alter the following CICS service properties <code>cicsCcsid</code>, <code>cicsConnectionRef</code>, and <code>cicsTransId</code>. For more information, see “Valid properties for use in policy rules” on page 687• The API Requester is enhanced to support code pages other than 037, to provide code page conversion between the runtime code page and UTF-8. For more information, see “Configuring for API requesters” on page 304.
	Fixes
	<ul style="list-style-type: none">• PI97371 - URISyntaxException: Illegal character in path is returned if a z/OS Connect EE service with a space in the name is deployed.

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
API toolkit V3.0.5.1	<p>Fixes</p> <ul style="list-style-type: none">• APAR PI76190: An API exported to an AAR file has a different name when deployed using the REST interface.• Service information is incorrectly generated when the service project name contains a space.• API toolkit incorrectly allows APIs with duplicate paths.• API operations in generated Swagger should be tagged with the API title.• The API toolkit does not remember the last export location for service archive and API archive.• The API editor does not remember the last service import location.• Editors in the API toolkit are not clearly labeled to ease communication. Titles are now added or modified in the service project editor, service interface editor, and API editor. The Overview tab in the service project editor is renamed to Definition to more appropriately reflect the purpose of the page.

May 2018

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.9 (APAR PI96947)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• Support for array type query parameters, path parameters, and HTTP headers for APIs is added. This feature requires API toolkit V3.0.5 or later. For more information, see “Defining and mapping headers, query parameters, or path parameters” on page 581.• When a request times out due to the setting of the <code>asyncRequestTimeout</code> attribute on the <code>zosconnect_zosConnectManager</code> element in <code>server.xml</code>, the client now receives a 503 HTTP response code, instead of a 500 HTTP response code, see “Configuration elements” on page 753.• The CICS service provider is updated so that if a request is timed out during asynchronous processing, the request is terminated and the associated IPIC session is freed. Previously the request would be left to attempt to run to completion in the server. For more information, see “Handling request timeouts” on page 229.• The <code>requestStateMap</code> parameter that is passed on the <code>invoke</code> method of the <code>com.ibm.zosconnect.spi.Service</code> interface has been updated to contain a new key, <code>IBM_ZOS_CONNECT_TIMEOUT</code>, which contains the time remaining for an asynchronous request. This enables service providers to terminate processing and free resources when a request is timed out.• The Welcome page of this product documentation has been enhanced with links to the most important topics so you can easily find what you need. We welcome your comments or suggestions for improvement of the documentation. Use the comments box at the base of each page to provide feedback. <p>Fixes</p> <ul style="list-style-type: none">• APAR PI90630: Message GMOBA0113E is not written to the z/OS Connect EE <code>messages.log</code> file.• APAR PI94737: z/OS Connect EE does not honor <code>preserveJsonObjectPayloadOrder="true"</code>.• APAR PI97210: A z/OS Connect EE service receives an incomplete response and message BAQR0404W is reported.

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
API toolkit V3.0.5	<p>Enhancements</p> <ul style="list-style-type: none">For request and response message mapping in the mapping editor, the data type for HTTP headers, query parameters, and path parameters can now be set to array. When the data type is set to array, you must specify the array data type (string, integer, boolean or number) and array format, which defines how the values in the parameter or header are delimited. This new feature requires server code V3.0.9 or later. For more information, see “Defining and mapping headers, query parameters, or path parameters” on page 581. <p>Fixes</p> <ul style="list-style-type: none">APAR PI97244: The API toolkit throws an exception when a service project is exported to a service archive (.sar) file from a directory that is different from the current workspace. The API toolkit has been updated to support service project export from a location that is different from the current workspace.APAR PI97413: When the base path of an API is specified with an ending slash ("/"), calling the deployed API from the embedded Swagger UI resulted in a BAQR7040E error message stating that the resource is not found. The API toolkit has been updated to not allow an ending slash in the API base path.

April 2018

V3.0.8 (APAR PI95501)	<p>Enhancements</p> <ul style="list-style-type: none">The build toolkit is updated to V1.1 to support the generation of the service archive (.sar) file and API archive (.aar) file from project directories created in the API toolkit. After a service project or API project is created or modified in the API toolkit, you can run the build toolkit to generate the service archive or API archive with the project directory as the input. This functionality enables continuous integration and delivery of services and APIs in a DevOps environment. For more information, see “DevOps with z/OS Connect EE” on page 14.The version of Liberty that is embedded in z/OS Connect EE is upgraded to V18.0.0.1. For more information, see What is new in this release of Liberty in the <i>WebSphere Application Server for z/OS Liberty</i> documentation.For users generating artifacts for an API requester, the z/OS Connect EE build toolkit can now process the allOf keyword in a Swagger file. <p>Fixes</p> <p>This release addresses the following issue:</p> <ul style="list-style-type: none">PI92773 SAR file changes are not retained on z/OS Connect EE server restart.
-----------------------	---

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
API toolkit V3.0.4.1	<p>Fixes</p> <p>This release addresses the following issues:</p> <ul style="list-style-type: none">• Unable to open an API in the Swagger UI when the API name contains special characters.• The help button in the Import Input or Output Message Data Structures dialog does not link to the correct topic.• No service deployment icon in the Configuration tab in the service project editor for ease of service deployment.• The buttons for adding and removing context containers HTTP headers in the Configuration tab for CICS channel services need appropriate tool tips.
March 2018	
V3.0.7 (APARs PI88452, PI89182, PI91663, PI93350) Server code update	<p>Enhancements</p> <ul style="list-style-type: none">• CICS service provider services can now be configured to send context containers to CICS programs, providing the program with information about the context in which the service was invoked. The use of context containers can be configured in both the API toolkit and the build toolkit. For more information, see “Context containers” on page 230. <p>Fixes</p> <ul style="list-style-type: none">• The IMSPingService service that tests if the IMS service provider is configured correctly for communication with IMS Connect was throwing a NullPointerException. This issue is addressed. For more information about the IMS Ping service, see “Verifying server communication with IMS” on page 245.• The z/OS Connect EE server tried to process integer fields even though they were not requested and not sent. This caused a NullPointerException and message BAQR7020E was written to the messages.log.• Data conversion problems returned an HTTP 500 error with no error details. This fix now returns an HTTP 400® bad request error, with the data conversion error details included in the JSON response. Some GMOMW messages for the data conversion function have been updated to use the correct term <i>service interface</i> instead of <i>transaction message</i>. See GMOMW messages for the updated message texts.
API toolkit V3.0.4	<p>Enhancements</p> <ul style="list-style-type: none">• CICS channel services can now be configured to send context containers to the target CICS program. This includes the ability to name specific HTTP headers to be sent when they are present on the request. Properties in the configuration tab have also been enhanced to use tool tips and provide context sensitive help. <p>Fixes</p> <ul style="list-style-type: none">• Tool tips are added to clarify the usage of the Container name and Interface rename fields for CICS channel services.
February 2018	

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.6 (APAR PI93830)	<p>Enhancements</p> <p>Server code update</p> <ul style="list-style-type: none">• The version of Liberty that is embedded in z/OS Connect EE is upgraded to V17.0.0.4. For more information, see What is new in this release of Liberty in the <i>WebSphere Application Server for z/OS Liberty</i> documentation.• The Scenario section of this IBM Knowledge Center is enhanced to make it easier for you to start using the features of z/OS Connect EE. See Chapter 4, “Quick Start Scenarios,” on page 55 and “Calling an API from a System of Record (SoR)” on page 176.• You can now use polling or MBeans to dynamically update and activate changes made to rule sets used by policies without needing to restart your server. Alternatively you can use the Refresh Modify command to apply changes to your policies and rules sets. For more information, see “Administering z/OS Connect EE policies” on page 683.• z/OS Connect EE now supports API key authentication. When calling APIs that are protected by API keys, your z/OS application can send the API key credentials through z/OS Connect EE either in a query string or as a request header. For more information, see “Call an API secured with an API key” on page 419. <p>Fixes</p> <ul style="list-style-type: none">• Previously the generated <code>ims-connections.xml</code> file upon initial creation of a server instance based on the <code>imsmobile</code> template contained the <code>connectionTimeout</code> and <code>connectionType</code> attributes that have no effect in z/OS Connect EE V3. However, if the <code>connectionTimeout</code> attribute was removed from the <code>ims-connections.xml</code> file, a CWWKG0058E error was incorrectly thrown upon server startup, complaining that the required <code>connectionTimeout</code> attribute was missing. These issues are now corrected. The generated <code>ims-connections.xml</code> file no longer contains these two extraneous attributes, and the <code>connectionTimeout</code> attribute is not expected to be present upon server startup.
API toolkit V3.0.3.1	<p>Fixes</p> <p>For service creation, FIXED BINARY and UNSIGNED FIXED BINARY (UBYTE) data types in PL/I were incorrectly converted to a data type of string in the generated JSON schema. They are now correctly converted to the data type of integer. This fix requires server code V3.0.6.</p>

January 2018

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
V3.0.5 (APAR PI92254)	Enhancements <ul style="list-style-type: none">• z/OS Connect EE build toolkit is enhanced to support JSON additional properties that are not explicitly defined in the properties section of a Swagger file. A new attribute additionalPropertiesSize is added in the build toolkit properties file. You can use this attribute to specify the maximum size for each JSON additional property when you use the build toolkit to generate artifacts for an API requester.• A sample COBOL application that calls out to a REST API through the API requester is provided in the hlq.SBAQSAMP data set. Two versions of the sample application are available: BAQPHBK1 (IMS transaction) and BAQPHBKB (z/OS application). These samples are used in the “Call an API from an IMS or z/OS application” on page 184 scenario, which provides step-by-step instructions on the required configuration, artifact generation, and client application development.• z/OS Connect EE build toolkit has been enhanced for the API Requester to support multidimensional arrays defined in Swagger files.• The version of Liberty that is embedded in z/OS Connect EE is upgraded to V17.0.0.3. For more information, see What is new in this release of Liberty in the <i>WebSphere Application Server for z/OS Liberty</i> documentation.
December 2017	
V3.0.4 (APAR PI90096)	Enhancements <ul style="list-style-type: none">• The RESTful service administration interface is updated to version 1.1.0. This new version adds the support for POST, PUT, and DELETE methods to support service deployment, service updates, service status updates, and service removal. You can also retrieve the service request schema and response schema with the GET method by using a URI without the action=getRequestSchema query string. For more information, see “Administering services with the administration interface” on page 655.• z/OS Connect EE is enhanced to support OAuth 2.0 to allow a z/OS application to call an API that is protected by OAuth 2.0 on a request endpoint. For more information, see “Call an API secured with OAuth 2.0” on page 421.• By using variable substitution in policy rules, you can override the IMS transaction code (imsTranCode) that the service invokes at run time. For more information, see “Valid properties for use in policy rules” on page 687.
API toolkit	
V3.0.3	Enhancements <ul style="list-style-type: none">• API toolkit V3.0.3 adds the support for deploying services directly from your service project without having to export the project to a service archive. You can view all deployed services, examine service properties, update a service, and start and stop a service directly from within the API toolkit. Server code V3.0.4 is required.
November 2017	

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
Server code update V3.0.3 (APAR PI88611)	<p>Enhancements</p> <ul style="list-style-type: none">• You can use CICS IPIC high availability to provide a single point of access from a z/OS Connect EE server to a cluster of CICS regions via a TCP/IP network. For more information, see “CICS IPIC high availability” on page 320.• The global admin group and admin group of a specific resource (services/ APIs/ API requesters) are enhanced to support multiple groups. You can set the value of the <code>globalAdminGroup</code> or <code>adminGroup</code> attribute in the <code>server.xml</code> file to a group name list with separating comma. The administration requests that are submitted by a user who belongs to any group listed in <code>globalAdminGroup</code> or <code>adminGroup</code> can be processed.• You can now use z/OS Connect EE policies to adjust how an API request is processed in z/OS Connect EE, based on the HTTP header values sent in by the client. For more information, see “Administering z/OS Connect EE policies” on page 683. <p>Fixes</p> <p>Empty array elements from service-level data conversion based on the settings in the Advanced Service Data Conversion Options dialog (by clicking Advanced Options in the service project editor in the API toolkit) are now correctly removed from the JSON output. If no valid array elements were converted, the array is completely removed from the JSON. For more information about advanced data conversion options, see “Service-level data conversion customization” on page 537.</p>
API toolkit V3.0.2.1	<p>Fixes</p> <p>This release includes minor fixes, including better user input validation of the value type for CCSID (CICS services) and the maximum length for transaction code (IMS services).</p>

October 2017

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
API toolkit V3.0.2	<p>Enhancements</p> <ul style="list-style-type: none">When customizing service interfaces, you can now use the Code page conversion option in the Edit Field dialog to indicate whether data of the "CHAR" type should be converted between UTF-8 and the host code page (such as IBM-1140) for the request and response interfaces.<ul style="list-style-type: none">By default, for the request interface, the JSON property value is converted from UTF-8 to the host code page prior to being placed in the respective field.By default, for the response interface, the field value is converted from the host code page to UTF-8 prior to being placed in the respective JSON property value. <p>This default behavior can be turned off by clearing the Code page conversion option.</p> <p>Fixes</p> <ul style="list-style-type: none">The mapping editor no longer interprets JSON properties of type "number" as integers when the "number" type has no fraction digits. The previous behavior was causing incorrect validation where incoming numeric quantities were limited to 10 digits.When you right-click a server in the z/OS Connect EE Servers view and select Refresh, the result now returns noticeably faster when you have more than a few deployed APIs.The mapping editor now correctly populates Swagger with type information for query parameters, path parameters, and headers that are mapped to float or double type service fields.

September 2017

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
Server code update	Enhancements <ul style="list-style-type: none">• The CICS service provider is enhanced to support distributed identity propagation to CICS during CICS authorization and for subsequent accountability and trace purposes. Distributed identities must be defined in an LDAP user registry and mapped to one or more SAF user IDs. Alternatively, an authenticated SAF user ID associated with the current request, can be passed to CICS for use during CICS authorization. The SAF user ID can originate from basic SAF credentials, an SSL client certificate, or other credential such as a JWT token, associated with A SAF user ID. Distributed identity support requires an IPIC connection that is configured with Userauth=Identify. For more information, see “Configuring security for an IPIC connection” on page 225.• The IMS service provider is enhanced to support distributed identity propagation to IMS Connect V15 for subsequent accountability and trace purposes. A distributed identity consists of a user ID and a network session ID (or realm). The user ID must be defined in an LDAP user registry, basic registry, or SAF registry. When the IMS service provider detects that the IMS Connect it is connecting to is V15, it sends the distributed identity by default. This distributed identity is not used by IMS for authentication or authorization purposes. For more information, see “Configuring distributed identity propagation to IMS” on page 253.• A PL/I sample is added to demonstrate how to handle large IMS data structures. The entire set of samples is also enhanced to allow setting and experimenting with different data structure sizes.<ul style="list-style-type: none">– A sample PL/I transaction program, its copybook, and related utilities and JCLs are added to the SBAQSAMP sample data set.– This set of samples is enhanced to allow you to set the size of the data structure that is sent to and received from IMS by changing the number of IO_RECORD array objects in the provided sample JSON documents. The existing COBOL sample is updated to work with this enhancement.
V3.0.2 (APARs PI85054 and PI87854)	<p>For more information, see “Handling large IMS data structures” on page 540.</p> <p>Note: Three member names for the existing COBOL sample have changed. The letter P in the following member names are removed: BAQLDSP (sample transaction) is now BAQLDS, BAQLDSPC (sample copybook) is now BAQLDSC, and BAQLDSPJ (sample JCL) is now BAQLDSJ. The letter P is used only in members that are for the PL/I sample.</p> <ul style="list-style-type: none">• The API requester is enhanced to support connection pooling in z/OS Connect EE to improve the efficiency of resource access to the request endpoints. For more information, see Connection pooling.• The API requester is enhanced to support specifying an additional domain path in a URL for an API that your z/OS applications plan to call. For more information, see “Setting the domain base path” on page 631.• New features are provided in the z/OS Connect EE product documentation:<ul style="list-style-type: none">– With the addition of two new languages, Spanish and Traditional Chinese, the IBM Documentation for z/OS Connect EE is now translated into 14 languages.– Where a code example is provided, you can now click  to copy the code into your clipboard.– APARs included in this release are listed in the documentation. They include links to the full description of the APAR.

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
API toolkit V3.0.1	<p>Enhancements</p> <ul style="list-style-type: none">A new service type IMS Large Data Structure Service is added to support importing a program with large data structures into the service interface editor for effective service interface definition. <p>Fixes</p> <ul style="list-style-type: none">API toolkit V3.0.1 addresses the following issues:<ul style="list-style-type: none">When COBOL copybooks or full programs or PL/I includes or full programs contain POINTER types, the import wizard does not tolerate POINTER types even though they are not part of the data structure the user selects. For COBOL, the message "POINTER not yet supported" is displayed and the data structure name remains disabled. For PL/I, the message "POINTER not yet supported" is displayed, and the user is allowed to select a structure. However, when the structure is expanded in the service interface editor, a null pointer exception occurs. This issue is addressed, and data structures containing POINTER types are now omitted from the data structure selection, allowing the user to proceed to select the structure to import.PL/I leaf arrays are not converted correctly for REFER statements, with minItems incorrectly set to 1 and maxItems set to 2 in the JSON schema. This issue is addressed. minItems is now set to 0 unless an LBOUND exists for the REFER array, whereas maxItems is always unbounded.PL/I string type leaf arrays are not converted correctly in the maxLength facet in the JSON schema. For example, an array of 5 items of a total of 255 characters, as shown in the following sample, is generated with a maxLength of 1275, when the maxLength should be 255.<pre>04 String_Array(5) CHARACTER(255)</pre>This issue is addressed.

July 2017

Server code update V3.0.1 (APAR PI83949)	<p>Enhancements</p> <ul style="list-style-type: none">A new feature, API requester, is now available in z/OS Connect EE. The new API requester feature allows z/OS applications to call REST APIs through z/OS Connect EE. You can use the build toolkit to generate the required artifacts based on the API description file of the REST API you plan to call. For more information, see "z/OS applications to call REST APIs" on page 9. <p>Fixes</p> <ul style="list-style-type: none">Stopping an IMS service resulted in an IMSGatewayException error, stating that "the creation or change to the <serviceName> configuration element SERVICE was not processed by the server." This issue is addressed.
--	---

Table 2. Enhancements and fixes.

The following table lists the maintenance level and the changes in each level for both the server code and the API toolkit.

(continued)

Maintenance level	Enhancements and fixes
API toolkit	Enhancements
V3.0.0.1	<ul style="list-style-type: none">The Remote Systems view is now available in the z/OS Connect EE perspective to ease service deployment. For more information about creating a connection to a remote system, see Creating a connection to a z/OS system (IBM Explorer for z/OS V3.1 product documentation).

New features introduced in z/OS Connect Enterprise Edition V3.0

z/OS Connect Enterprise Edition V3.0 builds upon the capabilities in IBM z/OS Connect Enterprise Edition V2.0 with these new features.

API toolkit

- The new IBM z/OS Connect Enterprise Edition V3.0 API toolkit simplifies the definition and deployment of application services for CICS and IMS environments. Previously known as the z/OS Connect Enterprise Edition API editor in V2, the z/OS Connect Enterprise Edition API toolkit now provides additional functions for service creation.
- The API toolkit provides a new type of project for services to complement the API projects, enabling end-to-end creation of z/OS Connect EE API and dependent services from within the same Eclipse-based environment.
- The new service projects that are introduced with the Service Interface Editor in the API toolkit, increase the number of eligible z/OS-based applications that can be API-enabled to include those with complex data structures, such as REDEFINES and OCCURS DEPENDING ON clauses for COBOL, and REFER in PL/I. See “[COBOL to JSON schema mapping](#)” on page 531 and “[PL/I to JSON schema mapping](#)” on page 535.
- The Service Interface Editor in the API toolkit enables the creator of the service project to perform renaming, redaction, and remarking for data fields that are used at the service level, encapsulating unique working knowledge of the target business logic that can greatly assist API creators to understand and use the true value of underlying services. See “[Customizing the service interface](#)” on page 496.

API requester

- A new feature, API requester, is now available from z/OS Connect Enterprise Edition V3.0.1. The API requester feature allows z/OS applications to call REST APIs through z/OS Connect EE. For z/OS applications, both COBOL and PL/I are supported.
- The new z/OS Connect EE build toolkit provides the ability to generate artifacts for an API requester based on the API description file of the REST API you plan to call. These artifacts include an API requester archive (.ara) file to be deployed to the z/OS Connect EE server, API information file and data structures that you use in your z/OS application program for calling REST APIs.

CICS service provider

- The new CICS service provider uses IP interconnectivity (IPIC) to connect to a remote CICS region, or an optimized direct link can be used when z/OS Connect EE is deployed to a CICS Liberty JVM server.
- Both COMMAREA and channel based CICS programs are supported. This includes channels with multiple containers and channels with both BIT and CHAR type containers, further expanding the applications that can be API enabled.

- The CICS service provider supports the automatic deployment of services (service archive files) when copied to the services directory. See “[Automated service archive management](#)” on page 670.

WOLA service provider updates

- The WOLA service provider supports the automatic deployment of services (service archive files) when copied to the services directory.

IMS service provider updates

- Programs running in IMS that are written in COBOL and PL/I, with data structures that exceed traditional message segment limits, are now supported, allowing more applications to be API enabled. For an IMS COBOL sample, see “[Handling large IMS data structures](#)” on page 540.
- The IMS service provider supports the automatic deployment of services (service archive files) when service archive files are copied to the services directory.
- Use the z/OS Connect EE API toolkit to create IMS services. The API toolkit now provides a consistent and integrated service creation experience for both the IMS service provider and the CICS service provider.

Important: As a result of the consolidated processes for service creation, deployment, and management:

- IMS Explorer for Development is not compatible and cannot connect to a z/OS Connect EE V3 server.
- IMS services that are created for V3 servers are not compatible with V2 servers. See “[Upgrading from z/OS Connect EE V2](#)” on page 203 for IMS service compatibility and migration issues if you are upgrading from V2.

Product documentation

- z/OS Connect EE V3.0 product documentation that is available in the online IBM Documentation can now be installed on a workstation for local access. You can install z/OS Connect EE product documentation from an installation repository. For more information, see <https://www.ibm.com/support/pages/node/733167>Installing and deploying IBM Documentation on IBM Intelligent Operations Center Windows and Linux Development Environments.

Other enhancements

- Use the REFRESH MODIFY command to refresh the z/OS Connect EE server artifacts such as Bindfiles, .aar, .sar, and .ara files, and the server.xml configuration file.

Removal notices

A removal notice signifies that the item can be removed, typically no sooner than two years from the time the notice was issued.

Table 3. Removal notices

EOS date	Category	Item to be removed	Strategic alternative
September 2019	Java	Removal of support Java 7 and the IBM SDK and Java runtime environment (JRE) for Java 7. For more information, see IBM developer kits lifecycle dates . In z/OS Connect EE V3.0.26, the Liberty kernel was recompiled and can no longer run with Java SE 7.	Use a more recent Java software development kit (SDK) or JRE. Liberty can run with any supported and compliant Java SDK or JRE, including IBM Java SDKs.

Table 3. Removal notices (continued)

EOS date	Category	Item to be removed	Strategic alternative
TBA	CICS	<p>Running z/OS Connect EE embedded in CICS is deprecated and support will be removed in a future release.</p> <p>If you run with z/OS Connect EE embedded in CICS, the minimum level of z/OS Connect EE is 3.0.41, and you must have APAR PH16415 or later applied to your CICS region to bring the minimum level of Liberty to 19.0.0.9.</p>	Use z/OS Connect EE stand-alone as described in this documentation.

The *end of service (EOS)* date is when IBM stops making fixes generally available.

Note: The API toolkit requires that CORS is enabled on the z/OS Connect EE server to connect, so z/OS Connect EE enabled CORS automatically, even when CORS support was not configured in `server.xml`. This automatic CORS enablement was removed in z/OS Connect EE V3.0.35. You should use the enhanced CORS capability that is supplied in Liberty by adding `Origin`, `Content-Type`, `Authorization`, `Cache-Control`, `Expires`, and `Pragma` to the **allowedHeaders** attribute of the `cors` configuration element. If you currently have no CORS configuration, you are advised to update your configuration to include CORS. For more information, see [“Configuring Cross-Origin Resource Sharing on a z/OS Connect Enterprise Edition Server” on page 300](#).

Chapter 4. Quick Start Scenarios

Follow the Quick Start Scenarios to quickly configure an end-to-end solution. There are quick start scenarios available for both API provider and API requester.

Click on the link to find the scenario you need:

API provider scenarios

- “[Using the CICS service provider to invoke the CICS catalog manager program.](#)” on page 55
- “[Using the IMS service provider to invoke the IMS phonebook transaction.](#)” on page 55
- “[Using the IMS database service provider to query the IMS database.](#)” on page 56
- “[Using the IBM MQ service provider to invoke the stock query application.](#)” on page 56
- “[Call a Db2 native REST service.](#)” on page 56
- “[Configure HA for connections to z/OS Connect EE.](#)” on page 57
- “[Configure HA for connections to CICS.](#)” on page 58

API requester scenarios

- “[Calling an API from a CICS transaction.](#)” on page 58
- “[Calling an API from an IMS or z/OS application.](#)” on page 59

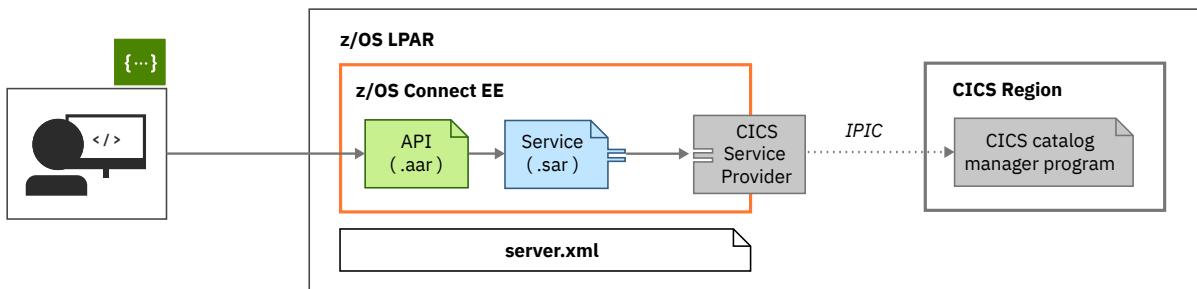
API provider Quick Start Scenarios

Templates are provided with pre-configured z/OS Connect EE configuration files and pre-generated artifacts for services and APIs. The scenarios also describe how these services and APIs are generated by the tools so you can re-create the services and APIs yourself.

Follow these scenarios to learn how to create and deploy services and APIs to access and act on resources on a z/OS subsystem through REST APIs.

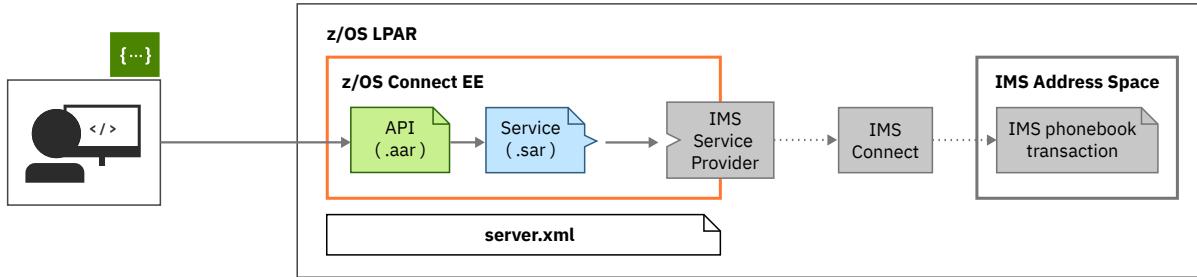
Using the CICS service provider to invoke the CICS catalog manager program.

In this scenario, you prepare the CICS catalog manager program, then create a z/OS Connect EE server by using the supplied template. This template includes the services and API required to call the CICS catalog manager program. It also contains a pre-configured `server.xml` file, which requires minor customization for your environment (for example, the host and port values for your IPIC connection to the CICS region). The scenario starts by preparing the CICS catalog manager program.



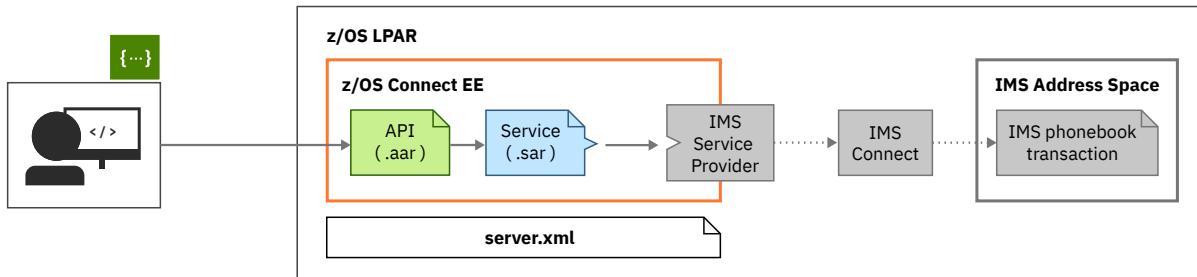
Using the IMS service provider to invoke the IMS phonebook transaction.

In this scenario, you prepare the IMS phonebook transaction first. You then create a z/OS Connect EE server by using the supplied template. This template includes the services, API, connection, and interactions profiles and a pre-configured `server.xml` file, that are required to call the IMS phonebook transaction. Only minor customization for your environment is required (for example, the host and port values for your connection to the IMS system). The scenario starts by preparing the IMS phonebook transaction.



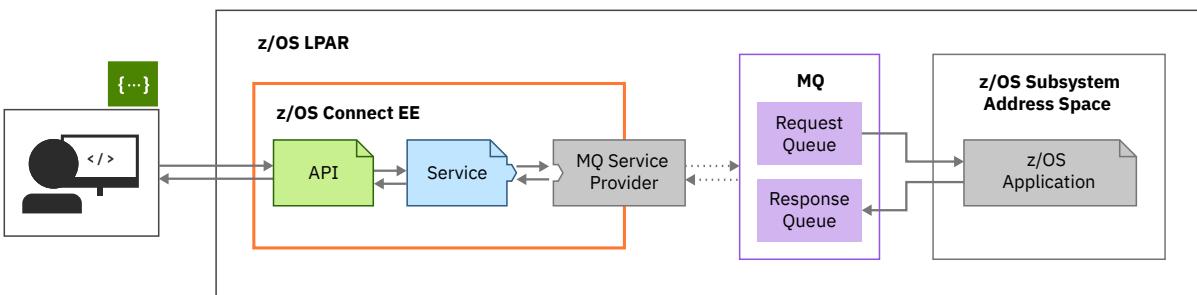
Using the IMS database service provider to query the IMS database.

In this scenario, you prepare the IMS database first. You then create a z/OS Connect EE server by using the supplied template. This template includes the services, connection profiles, and a pre-configured `server.xml` file that are required to query the IMS database. Only minor customization for your environment is required (for example, the host and port values for your connection to the IMS database). The scenario starts by preparing the sample IMS database.



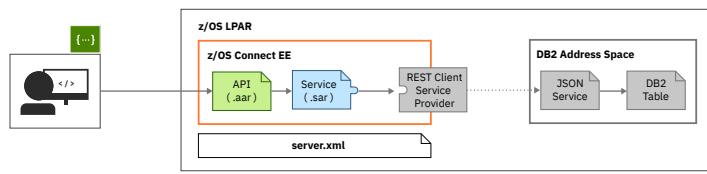
Using the IBM MQ service provider to invoke the stock query application.

In this scenario, you prepare the stock query application first. You then create a z/OS Connect EE server, a two-way service, and an API to invoke the stock query application. The scenario starts by preparing the stock query application.



Call a Db2 native REST service.

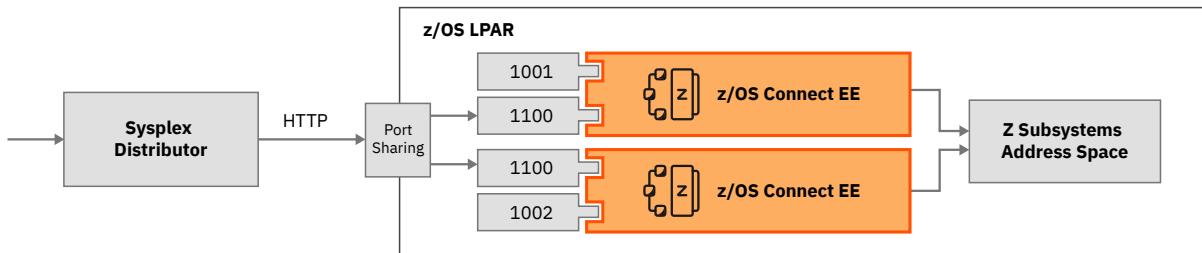
In this scenario, Db2 native REST services are created based on the Db2 sample Employee table. You then create a z/OS Connect EE server by using the supplied template. This template includes the services and API required to call the created Db2 native REST services. It also contains a pre-configured `server.xml` file, which requires minor customization for your environment. For example, the host and port values for your connection to the Db2 instance. This scenario starts by preparing Db2 to create native REST services.



Configure HA for connections to z/OS Connect EE.

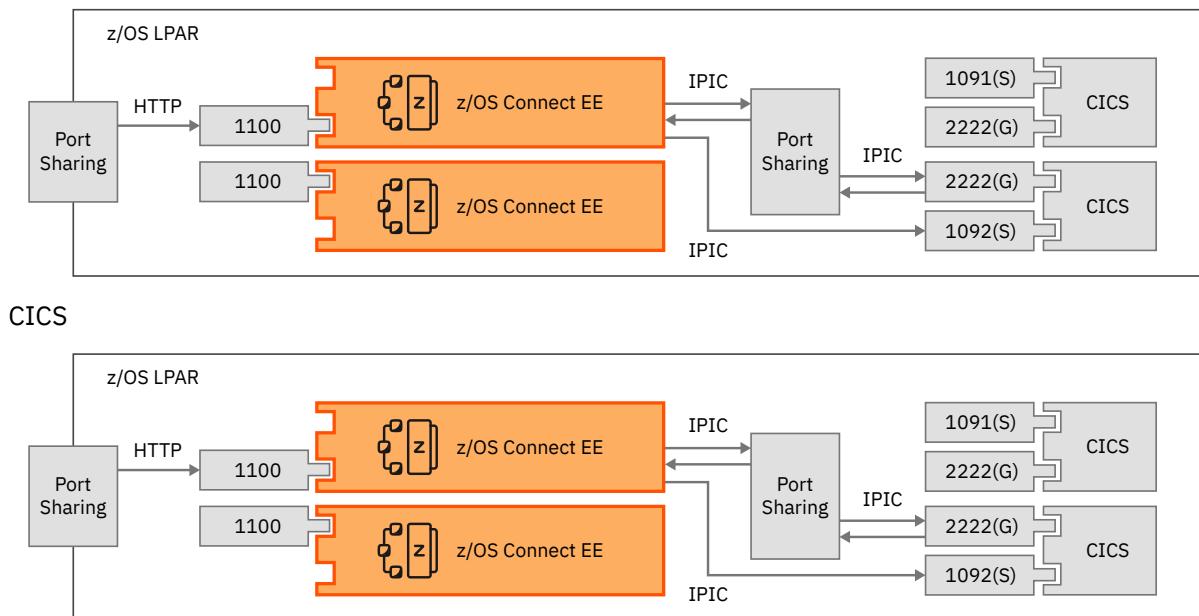
This scenario shows you how to implement high availability for inbound requests to two z/OS Connect EE servers that use port sharing and shared configuration data. The scenario guides you through setting up TCP/IP port sharing, creating a shared configuration file, setting up shared resources, the

z/OS Connect EE servers, and test your scenario works in a high availability fashion. The scenario starts with [configuring HA connections to z/OS Connect EE](#).



Configure HA for connections to CICS.

This scenario shows you how to implement high availability for connections to two CICS regions. The scenario guides you in configuring your CICS regions and z/OS Connect EE servers for IPIC HA, and tests your scenario is working in a high availability fashion. The scenario starts with [configuring HA connections into CICS](#).



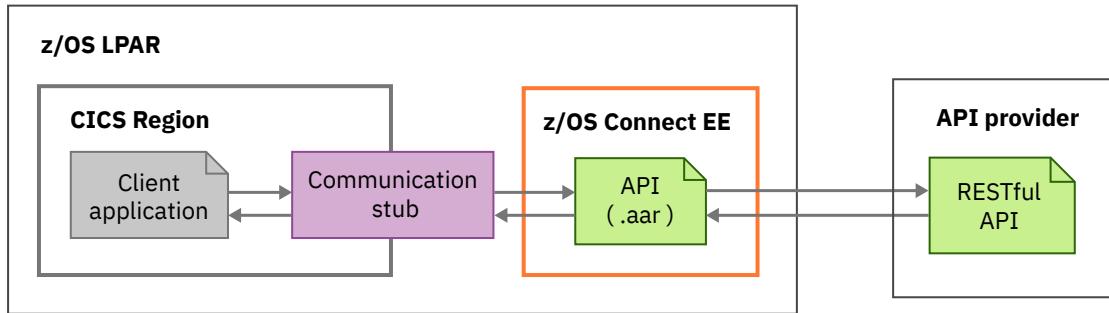
API requester Quick Start Scenarios

These scenarios provide step-by-step instructions for your CICS, IMS, or z/OS application to call RESTful APIs through z/OS Connect Enterprise Edition.

These scenarios describe how to create your API requester by using the supplied tools and call a remote RESTful API endpoint.

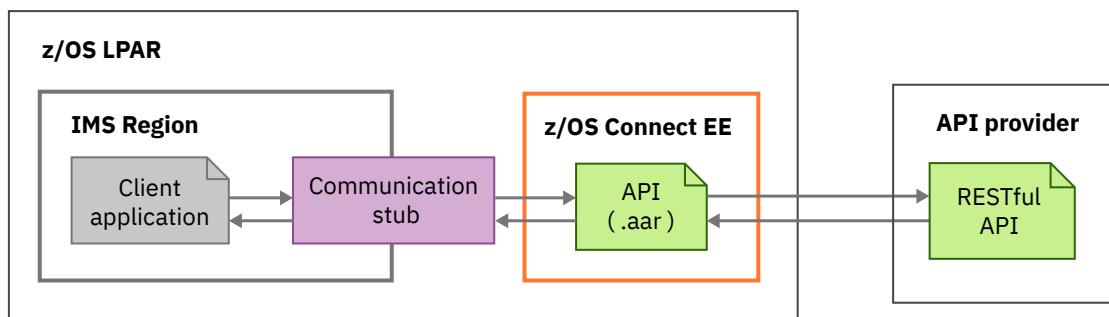
Calling an API from a CICS transaction.

This scenario shows you how to make your CICS application call your z/OS Connect EE server. The server then makes an API requester call to the RESTful API endpoint, which is simulated by a second z/OS Connect EE server and a CICS region that hosts the catalog manager program. The scenario starts with [preparing the RESTful API endpoint](#).



Calling an API from an IMS or z/OS application.

This scenario shows you how to make a call from an IMS or z/OS application to your z/OS Connect EE server. Your server then makes an API requester call to the RESTful API endpoint, which is simulated by a second z/OS Connect EE server and IMS region that hosts the phonebook application. The scenario starts with [preparing the RESTful API endpoint](#).



Prepare a sample System of Record (SoR) application

These tasks show you how to install and configure a sample System of Record (SoR) application that is invoked by z/OS Connect EE in other scenarios.

Select the appropriate task for your SoR.

Prepare the sample CICS application

The CICS Catalog Manager sample application is used by the z/OS Connect EE scenarios that use the CICS service provider. This sample application needs to be installed and configured in your target CICS region.

This scenario shows you how to prepare the CICS catalog manager sample application and test that it is configured and working correctly. In subsequent scenarios, you use z/OS Connect EE to send requests to this application.

The following tasks describe how to prepare the sample CICS application.

Install and configure the CICS catalog manager application

This task shows you how to install and configure the CICS catalog manager sample application in your CICS region.

Procedure

Ask your CICS system administrator to set up the base application. Follow the procedure in [Installing and setting up the base application](#) in the *CICS Transaction Server* documentation.

Note: You do not need to configure web service support for this scenario.

Pay special attention to the following steps in the *base application* section of the documentation:

- Ensure that SDFHLOAD is in your DFHRPL concatenation to install the CICS sample applications.
- Create and define the two VSAM data sets.
- Create TRANSACTION definitions for the two user interface transactions, ECFG, and EGUI.
- Run ECFG to change any defaults for the application configuration.

Results

The sample application is now configured on CICS.

What to do next

Test the application to ensure that it is installed and configured correctly. See “[Test the CICS catalog manager application](#)” on page 60.

Test the CICS catalog manager application

This task shows you how to test the CICS catalog manager sample application using a CICS 3270 emulator to ensure it is installed correctly.

Procedure

1. In a CICS 3270 emulator session, enter the transaction ID EGUI.

The application menu is displayed:

```
CICS EXAMPLE CATALOG APPLICATION - Main Menu
Select an action, then press ENTER
Action . . . . - 1. List Items
                           2. Order Item Number ----
                           3. Exit

F3=EXIT    F12=CANCEL
```

2. Type 1 and press **Enter** to select the List Items option.

The application displays a list of items in the catalog.

```
CICS EXAMPLE CATALOG APPLICATION - Inquire Catalog
Select a single item to order with /, then press ENTER

Item   Description          Cost     Order
-----+
0010  Ball Pens Black 24pk    2.90    -
0020  Ball Pens Blue 24pk    2.90    -
0030  Ball Pens Red 24pk    2.90    -
0040  Ball Pens Green 24pk   2.90    -
0050  Pencil with eraser 12pk 1.78    -
0060  Highlighters Assorted 5pk 3.89    -
0070  Laser Paper 28-lb 108 Bright 500/ream 7.44    -
0080  Laser Paper 28-lb 108 Bright 2500/case 33.54    -
0090  Blue Laser Paper 20lb 500/ream      5.35    -
0100  Green Laser Paper 20lb 500/ream     5.35    -
0110  IBM Network Printer 24 - Toner cart 169.56    -
0120  Standard Diary: Week to view 8 1/4x5 3/4 25.99    -
0130  Wall Planner: Eraseable 36x24       18.85    -
0140  70 Sheet Hard Back wire bound notepad 5.89    -
0150  Sticky Notes 3x3 Assorted Colors 5pk    5.35    -

F3=EXIT    F7=BACK    F8=FORWARD   F12=CANCEL
```

3. Type / in the Order column next to an item and press **Enter** to place an order for that item.

For example, if you selected item 0010, the following screen is displayed.

CICS EXAMPLE CATALOG APPLICATION - Details of your order					
Enter order details, then press ENTER					
Item	Description	Cost	Stock	On Order	
0010	Ball Pens Black 24pk	2.90	0126	000	
Order Quantity: <input type="text"/> User Name: <input type="text"/> Charge Dept: <input type="text"/>					
F3=EXIT F12=CANCEL					

4. If stock levels are sufficient to fulfill the order, enter the following information:
 - a) In the **Order Quantity** field, specify the number of items you want to order.
 - b) In the **User Name** field, enter a 1-to 8-character string. The sample application does not check the value that is entered here.
 - c) In the **Charge Dept** field, enter a 1-to 8-character string. The sample application does not check the value that is entered here.
5. Press **Enter** to submit the order and return to the main menu.
6. Press **F3** to end the application.

Results

The CICS catalog manager example application is successfully installed and configured and you can use this sample with the CICS scenarios that are described in this documentation.

What to do next

Follow the steps in [“Create a server and connect to CICS” on page 69](#) to create a connection to the CICS region.

Prepare the sample IMS application

The IMS Phonebook sample application (the IVTNO transaction) is used by the z/OS Connect EE scenarios that use the IMS service provider. This sample application needs to be installed in your target IMS region.

The IMS phonebook application is set up through the installation verification program (IVP) H216T and H213J. Refer to the [IVP sample application](#). In the IMS documentation the application is referred to as the telephone application.

Test the IMS phonebook application

This task shows you how to test the IMS IMS phonebook sample application using an IMS user terminal to ensure it is installed correctly.

Procedure

1. In an IMS user terminal, enter /FOR IVTNO.

The application menu is displayed:

```

*****
*      IMS INSTALLATION VERIFICATION PROCEDURE      *
*****
TRANSACTION TYPE : NON-CONV (OSAM DB)
DATE           : 01/26/2018

PROCESS CODE  (*1) :
LAST  NAME     :
FIRST NAME    :
EXTENSION NUMBER :
INTERNAL ZIP CODE :

(*1) PROCESS CODE
      ADD
      DELETE
      UPDATE
      DISPLAY
      TADD

SEGMENT# :

```

2. Type DISPLAY in the PROCESS CODE field, LAST5 in the LAST NAME field, and press **Enter**.
The application displays the details for the person whose last name is LAST5.

```

*****
*      IMS INSTALLATION VERIFICATION PROCEDURE      *
*****
TRANSACTION TYPE : NON-CONV (OSAM DB)
DATE           : 01/26/2018

PROCESS CODE  (*1) : DISPLAY
LAST  NAME     : LAST5
FIRST NAME    : FIRST5
EXTENSION NUMBER : 8-111-5555
INTERNAL ZIP CODE : D05/R05

(*1) PROCESS CODE
      ADD
      DELETE
      UPDATE
      DISPLAY
      TADD

ENTRY WAS DISPLAYED
SEGMENT# : 0001

```

3. Clear the screen to end the application.

Results

The IMS phonebook sample application is successfully installed and configured and you can use this sample with the IMS scenarios that are described in this documentation.

What to do next

Follow the steps in [“Create a server and connect to IMS” on page 72](#) to create a connection to the IMS region.

Prepare the sample IMS database

The IMS installation verification program (IVP) sample application database is used by the z/OS Connect EE scenarios that use the IMS database service provider. This sample IVP database must be installed in your target IMS region to follow this guide.

The IMS database used in the following example is set up during the process of running the installation verification program. Refer to [IMS installation verification program \(IVP\) overview](#) in the *IMS documentation* to learn more about configuring the sample application database with IMS IVP.

The IMS database service provider uses the IMS Open Database solution to establish a connection, and make requests to, an IMS database.

Restriction: There are multiple restrictions for the IMS database service provider. Ensure that you understand the capabilities of the IMS database service provider by reading the [operational requirements](#).

Important: Ensure that your IVP sample application database is configured with the additional following jobs and tasks:

- DBC system (DBCTL)
- DBT system (DB/DC)

To learn more about the jobs and tasks that are involved in the IVP process, see [IVP jobs and tasks](#) for more.

If you are new to using the installation verification program, see [IMS installation verification program \(IVP\) overview](#).

What to do next

Follow the steps in “[Create a server and connect to an IMS database](#)” on page 74 to create a connection to the IMS database.

Prepare the sample IBM MQ stock query application

This scenario describes a simple stock query application, which is written in Java, that uses the Java Message Service (JMS) API to interact with IBM MQ. This application is used by the z/OS Connect EE scenarios to demonstrate two-way services with the IBM MQ service provider.

The stock query application takes request messages from an IBM MQ queue. The messages contain the ID of the item that the caller wants information on. The application then generates a response message that contains information on the requested item and sends it to a response queue. The requesting application takes the response message from the response queue.

Compiling the application

The code for the stock query application is shown in [Figure 9 on page 64](#).

```

import javax.jms.JMSConsumer;
import javax.jms.JMSCContext;
import javax.jms.JMSProducer;
import javax.jms.Message;
import javax.jms.Queue;
import javax.jms.TextMessage;
import com.ibm.mq.jms.MQConnectionFactory;

/**
 * Simple back end application which generates responses to stock queries.
 */
public class TwoWayBackend {
    public static void main(String[] args) {
        String queueManagerName = args[0];
        String requestQueueName = args[1];
        String responseQueueName = args[2];

        try {
            //Connect to queue manager.
            MQConnectionFactory cf = new MQConnectionFactory();
            cf.setQueueManager(queueManagerName);

            JMSCContext ctx = cf.createContext();

            //Connect to request queue.
            JMSConsumer consumer = ctx.createConsumer(ctx.createQueue(requestQueueName));

            //Create JMS producer.
            JMSProducer producer = ctx.createProducer();

            //Create response queue.
            Queue responseQueue = ctx.createQueue(responseQueueName);

            //Loop round getting messages, and generating a response.
            while(true) {
                System.out.println("Waiting for message. Press Ctrl+C to exit.");

                TextMessage requestMessage = null;

                //This loop provides a succinct way of breaking out of the loop by pressing Ctrl
                +C.
                while(requestMessage == null) {
                    requestMessage = (TextMessage) consumer.receiveNoWait();
                    if (requestMessage == null) {
                        Thread.sleep(1000);
                    }
                }

                //Request message payload is a 6 digit integer representing an item id.
                String itemID = requestMessage.getText().trim();

                System.out.println("Message received: " + itemID + ".");

                //Response message is a 6 digit integer item id,
                //followed by a 20 character description,
                //a 6 digit integer item count and a
                //a 6 digit integer item cost.

                //Return a hard coded response body, with the itemID inserted.
                String responseMessagePayload = String.format("%sA description.
0005000000045", itemID);

                Message responseMessage = ctx.createTextMessage(responseMessagePayload);

                //Response messages have the correlation id set to the message id of the
                request message.
                responseMessage.setJMSCorrelationID(requestMessage.getJMSMessageID());

                //Send response.
                producer.send(responseQueue, responseMessage);
            }
        } catch(Exception e) {
            System.out.println("Caught exception ");
            e.printStackTrace();
        }
    }
}

```

Figure 9. Sample stock query application

To compile the application, copy the sample code in [Figure 9 on page 64](#) to a file called `TwoWayBackend.java` on a UNIX System Services environment on a z/OS LPAR where IBM MQ for z/OS is installed, including the IBM MQ for z/OS UNIX Systems Services Components feature.

Using a Java 7.1 or later Java SE Development Kit, enter the following command to compile the application:

```
javac -cp /usr/lpp/mqm/V9R1M0/java/lib/com.ibm.mq.allclient.jar TwoWayBackend.java
```

If necessary, replace `/usr/lpp/mqm/V9R1M0/` with the path to the installation of the IBM MQ for z/OS UNIX Systems Services Components feature.

Defining the queues for the application

Work with your IBM MQ administration to define a pair of queues for use by the stock query application. Both a request and a response queue are needed. For example, to define a request queue called `STOCK_REQUEST` and a response queue called `STOCK_RESPONSE` enter the following MQSC commands:

```
DEF QL(STOCK_REQUEST)
DEF QL(STOCK_RESPONSE)
```

Running the application

To run the stock query application, first ensure that the UNIX System Services STEPLIB includes the IBM MQ SCSQAUTH and SCSQANLE libraries. Specify these libraries in the startup JCL or using the `.profile` file. From UNIX and Linux® System Services, you can include these libraries by adding the following line to your `.profile`. Replace `thlqual` with the high-level data set qualifier that you chose when installing IBM MQ:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

For more information, see [STEPLIB configuration for IBM MQ classes for JMS on z/OS](#) in the *IBM MQ* documentation.

Enter the following command from UNIX System Services, replacing `MQ21` with the name of the queue manager where the `STOCK_REQUEST` and `STOCK_RESPONSE` queues were defined. If necessary, replace both instances of `/usr/lpp/mqm/V9R1M0/` with the installation path of the IBM MQ for z/OS UNIX System Services Components feature.

```
java -cp /usr/lpp/mqm/V9R1M0/java/lib/com.ibm.mq.allclient.jar:. -Djava.library.path=/usr/lpp/mqm/V9R1M0/java/lib/ TwoWayBackend MQ21 STOCK_REQUEST STOCK_RESPONSE
```

The following output is displayed.

```
Waiting for message. Press Ctrl+C to exit.
```

If a problem occurs, exception information is displayed that you can use to analyze the problem.

To stop the application, enter `Ctrl+C` in the command window where the application was run.

Testing the application

Use [MQ Explorer](#) to connect to your [MQ queue manager on z/OS](#) and [send a message](#) to the `STOCK_REQUEST` queue. Use a message payload value that consists of a six-digit integer, left padded with zeros. For example, `001234`.

The following output is displayed by the stock query application:

```
Waiting for message. Press Ctrl+C to exit.
Message received: 001234
Waiting for message. Press Ctrl+C to exit.
```

If a problem occurs, exception information is displayed that you can use to analyze the problem.

You can now use IBM MQ Explorer to [browse](#) the response message on the STOCK_RESPONSE queue.

What to do next

Follow the steps in “[Create a server to connect to IBM MQ](#)” on page 77 to create a connection to the IBM MQ queue manager.

Prepare the sample Db2 native REST services

This scenario shows you how to prepare Db2 sample tables, then create, enable, and test Db2 native REST services.

In subsequent scenarios, you use the z/OS Connect EE API toolkit to create a Db2 service project and finally create a RESTful API to expose the Db2 native REST services.

The following tasks describe how to prepare the Db2 native REST services.

Enable and create Db2 native REST services

Enable and create Db2 native REST services that use the Db2 sample tables. These services are then used in subsequent scenarios.

About this task

In this scenario, you create three Db2 native REST services:

- `employeeList` - Returns a list of employees.
- `employeeDetails` - Returns the details of an individual employee.
- `employeeUpdate` - Updates the details of an individual employee.

These services are created using the Db2 Service Manager native REST service. Alternatively, the BIND SERVICE subcommand can be used to create a new Db2 native REST service. For each request replace `<db2_host_name>` with the host name of your Db2 instance and `<db2_port>` with the port of your Db2 instance. You might also need to include authentication credentials such as the Authorization HTTP header.

The services created in this scenario use the default Db2 collection SYSIBMSERVICE.

Procedure

1. Ask your Db2 system administrator to create the Db2 sample tables and enable the creation of Db2 native REST services by following this procedure.
 - a) Create the Db2 sample tables by running the Db2 installation sample jobs DSNTEJ1 and DSNTEJ7. The full procedure to install the Db2 samples tables is documented in the [Db2 sample table topic](#) in the *Db2 for z/OS* documentation.
 - b) Enable Db2 native REST service creation
Follow the steps documented in the [Enabling Db2 REST services topic](#) in the *Db2 for z/OS* documentation.
- Note:** You must apply the PTF for Db2 APAR PI98649: New function update of Db2 native RESTful services support.
2. Create the `employeeList` service.
Using a REST client, send a HTTP POST request to `https://<db2_host_name>:<db2_port>/services/DB2ServiceManager` with header `Content-Type: application/json` and the following body:

```
{  
  "requestType": "createService",  
  "sqlStmt": "SELECT EMPNO AS \"employeeNumber\", FIRSTNAME AS \"firstName\", LASTNAME AS  
  \"lastName\", WORKDEPT AS \"department\" FROM DSN81210.EMP ORDER BY LASTNAME",  
  "serviceName": "employeeList",  
}
```

```

    "description": "Returns a list of employees.",
    "collectionID": "SYSIBMSERVICE"
}

```

The SQL statement provided in sqlStmt uses SQL aliases to rename columns when they appear in the response JSON, this helps provide a more natural JSON property name than the Db2 table column names would normally provide. A HTTP Created (201) response code is returned if the Db2 native REST service is created successfully.

3. Create the employeeDetails service.

Using a REST client, send a HTTP POST request to `https://<db2_host_name>:<db2_port>/services/DB2ServiceManager` with header `Content-Type: application/json` and the following body:

```

{
  "requestType": "createService",
  "sqlStmt": "SELECT EMPNO AS \"employeeNumber\", FIRSTNME AS \"firstName\", LASTNAME AS \"lastName\", WORKDEPT AS \"department\", PHONENO AS \"phoneNumber\", HIREDATE AS \"hireDate\" FROM DSN81210.EMP WHERE EMPNO = :employeeNumber",
  "serviceName": "employeeDetails",
  "description": "Returns the details of an individual employee.",
  "collectionID": "SYSIBMSERVICE"
}

```

A HTTP Created (201) response code is returned if the Db2 native REST service is created successfully.

4. Create the employeeUpdate service.

Using a REST client, send a HTTP POST request to `https://<db2_host_name>:<db2_port>/services/DB2ServiceManager` with header `Content-Type: application/json` and the following body:

```

{
  "requestType": "createService",
  "sqlStmt": "UPDATE DSN81210.EMP SET FIRSTNME = :firstName, LASTNAME = :lastName, WORKDEPT = :department, PHONENO = :phoneNumber, HIREDATE = :hireDate WHERE EMPNO = :employeeNumber",
  "serviceName": "employeeUpdate",
  "description": "Updates the details of an individual employee.",
  "collectionID": "SYSIBMSERVICE"
}

```

A HTTP Created (201) response code is returned if the Db2 native REST service is created successfully.

Results

Db2 native REST services `employeeList`, `employeeDetails`, and `employeeUpdate` have been created as part of the `SYSIBMSERVICE` collection in Db2.

What to do next

Test that the Db2 native REST services have been created correctly. See [“Test the Db2 native REST services” on page 67](#).

Test the Db2 native REST services

Test the created Db2 native REST services to ensure they are installed correctly.

About this task

Use a REST client to test the Db2 native REST services. Three different API requests are made, one for each of the created Db2 native REST services.

For each request, replace `<db2_host_name>` with the hostname of your Db2 instance and `<db2_port>` with the port of your Db2 instance. You can also include authentication credentials such as the Authorization HTTP header.

Procedure

1. Use a REST client to test the employeeList service.

Send an HTTP POST request to https://<db2_host_name>:<db2_port>/services/SYSIBMSERVICE/employeeList/V1 with header Content-Type: application/json and the following body:

```
{}
```

A response code HTTP OK (200) indicates success. No JSON properties are required for the employeeList Db2 native REST service, so an empty JSON object payload is used. The response body starts with:

```
{
  "ResultSet Output": [
    {
      "employeeNumber": "000150",
      "firstName": "BRUCE",
      "lastName": "ADAMSON",
      "department": "D11"
    },
    ...
  ],
  "StatusCode": 200,
  "StatusDescription": "Execution Successful"
}
```

2. Use a REST client to test the employeeDetails service.

Send an HTTP POST request to https://<db2_host_name>:<db2_port>/services/SYSIBMSERVICE/employeeDetails/V1 with header Content-Type: application/json and the following body:

```
{
  "employeeNumber": "000010"
}
```

A response code HTTP OK (200) indicates success. A JSON property named employeeNumber is required for the employeeDetails Db2 native REST service. The value of this property is inserted into the SQL statement that is executed. The response body is:

```
{
  "ResultSet Output": [
    {
      "employeeNumber": "000010",
      "firstName": "CHRISTINE",
      "lastName": "HAAS",
      "department": "A00",
      "phoneNumber": "3978",
      "hireDate": "1965-01-01"
    }
  ],
  "StatusCode": 200,
  "StatusDescription": "Execution Successful"
}
```

3. Use a REST client to test the employeeUpdate service.

Send an HTTP POST request to https://<db2_host_name>:<db2_port>/services/SYSIBMSERVICE/employeeUpdate/V1 with header Content-Type: application/json and the following body:

```
{
  "employeeNumber": "000010",
  "firstName": "CHRISTINE",
  "lastName": "HAAS",
  "department": "A00",
  "phoneNumber": "3979",
  "hireDate": "1965-01-01"
}
```

A response code HTTP OK (200) indicates success. Six JSON properties named employeeNumber, firstName, lastName, department, phoneNumber, and hireDate are required for the employeeUpdate Db2 native REST service. The values of both properties are inserted into the SQL statement that is executed. The response body is:

```
{  
    "Update Count": 1,  
    "StatusCode": 200,  
    "StatusDescription": "Execution Successful"  
}
```

Results

The Db2 native REST services employeeList, employeeDetails, and employeeUpdate returned the expected responses.

What to do next

Complete the task “Create a server to connect to Db2” on page 80 to create a connection to the Db2 region in z/OS Connect EE.

Create a server and connect to a System of record (SoR)

These tasks show you how to create a z/OS Connect EE server and configure a connection to a System of Record (SoR).

Create a server and connect to CICS

This task shows you how to create a z/OS Connect EE server and configure an IPIC connection to a CICS region.

Before you begin

Ensure that the following tasks are complete:

1. Install and configure the CICS catalog manager example application. The procedure is described in the scenario “[Prepare the sample CICS application](#)” on page 59.
2. Install z/OS Connect EE v3.0 or later. For more information, see “[Installing z/OS Connect EE](#)” on page 196.

Pre-built services and an API are provided to simplify testing of this scenario.

About this task

This task consists of three parts:

- Creating a z/OS Connect EE server with ready-to-use services and an API for the CICS catalog manager example application.
- Configuring an IPIC connection between your z/OS Connect EE server and CICS.
- Testing the IPIC connection.



Procedure

1. Define a TCPIPSERVICE to listen for inbound IPIC requests in your CICS region. This scenario uses a port value of 1091. For more information about defining a TCPIPSERVICE for inbound IPIC requests, see [Configuring the IPIC connection](#) in the *CICS Transaction Server* documentation.

2. Create a server.

Enter the following command from the <installation_path>/bin directory. For example, /usr/lpp/IBM/zosconnect/v3r0/bin:

```
zosconnect create catalogManager --
template=zosconnect:sampleCicsIpicCatalogManager
```

The following artifacts are created:

- A catalog API service archive file, catalog.aar, in the directory <WLP_USER_DIR>/servers/catalogManager/resources/zosconnect/apis.
- Three services archive files, inquireCatalog.sar, inquireSingle.sar, and placeOrder.sar, in the directory <WLP_USER_DIR>/servers/catalogManager/resources/zosconnect/services.
- A server.xml configuration file in the directory <WLP_USER_DIR>/servers/catalogManager with the zosconnect:cicsService-1.0 feature included.

3. Customize the z/OS Connect EE server configuration file.

- Update the CICS connection element:

```
<zosconnect_cicsIpicConnection id="cicsConn" host="localhost" port="1091"/>
  - If your server and CICS region are on different LPARs, replace the host value localhost with the hostname or IP address of the LPAR hosting your CICS region.
  - Replace the port value 1234 with the port that your CICS TCPIPSERVICE is configured to listen for inbound IPIC requests. This scenario uses port 1091.
```

- Update the httpPort for inbound connections into z/OS Connect EE in the following element, if necessary:

```
<httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="9080"
httpsPort="-1"/>
```

- Enable security.

Test the IPIC connection

You can test your connection to your CICS region by invoking the catalog API directly from z/OS Connect EE API toolkit by using the "Try it out!" function in the Swagger UI that is embedded in the editor.

Procedure

1. Start CICS and ensure that the TCPIPSERVICE is open.

2. Start your z/OS Connect EE server. For more information, see ["Starting and stopping z/OS Connect EE"](#) on page 471.

- a) Check the messages.log file for the following messages that confirm that the services are installed.

```
BAQR7043I: z/OS Connect EE service archive placeOrder installed successfully.
BAQR7043I: z/OS Connect EE service archive inquireSingle installed successfully.
BAQR7043I: z/OS Connect EE service archive inquireCatalog installed successfully.
```

- b) Check the messages.log file for the following message that confirms that the API is installed.

```
BAQR7000I: z/OS Connect API archive file catalog installed successfully.
```

3. In the **z/OS Connect EE Servers** view under the **APIs** folder, double-click on the catalog API.

The API is opened in the Swagger UI.

4. Click **List Operations** to see available operations in the API.

catalog

The screenshot shows the Swagger UI interface for a 'catalog' API. At the top, there are three buttons: 'Show/Hide', 'List Operations' (which is highlighted with a red box), and 'Expand Operations'. Below these are three operation entries: a blue 'GET' button for '/items', a blue 'GET' button for '/items/{itemID}', and a green 'POST' button for '/orders'. At the bottom left, it says '[BASE URL: /catalogManager , API VERSION: 1.0.0]'.

Figure 10. Operations in the API

5. Test inquiring on an item by clicking **GET /items/{itemID}**.

- Specify a value of 10 for the itemID.

The screenshot shows the configuration for a GET request to '/items/{itemID}'. The 'Response Class (Status 200)' is listed as 'normal response'. The 'Model' tab is selected, showing a JSON schema for the response body. The 'Parameters' section shows two entries: 'itemID' with a value of '10' and 'Authorization' with an empty value. At the bottom, there are 'Try it out!' and 'Hide Response' buttons.

Figure 11. Testing the GET request

- Click **Try it out!**.

Information about the request URL, request headers, response body, response code, and response headers are provided. The response body contains the output message:

- "CA_SNGL_DESCRIPTION": "Ball Pens Black 24pk"

Results

You have verified your IPIC connection to CICS is working.

What to do next

Follow the steps in ["Create a CICS service" on page 83](#) or ["Sample for creating a CICS service with a properties file" on page 810](#) to see how to create a service for yourself.

Create a server and connect to IMS

This task shows you how to create a z/OS Connect EE server and configure a connection to an IMS system.

Before you begin

Ensure that the following tasks are complete:

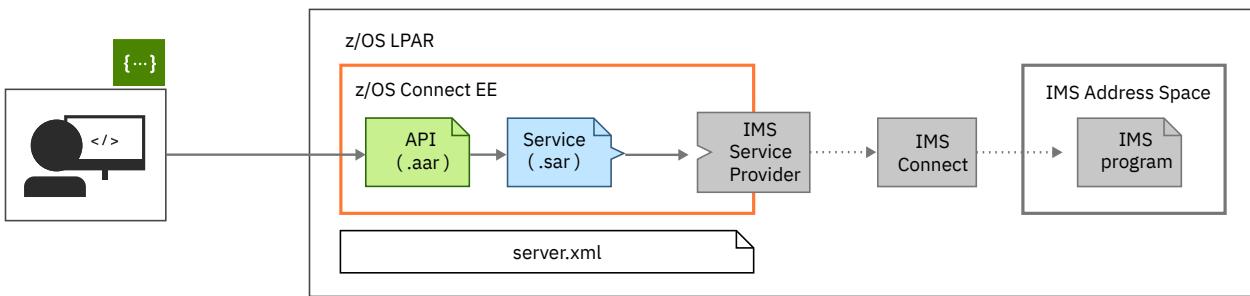
1. Install and configure the IMS phonebook application. The procedure is described in the scenario “[Prepare the sample IMS application](#)” on page 61.
2. Install z/OS Connect EE v3.0 or later. For more information, see “[Installing z/OS Connect EE](#)” on page 196.

Pre-built services and an API are provided to simplify testing of this scenario.

About this task

This task consist of three parts:

- Creating a z/OS Connect EE server with ready-to-use services and an API for the IMS phonebook application.
- Configuring an IMS connection between your z/OS Connect EE server and IMS.
- Testing the IMS connection.



Procedure

1. On your IMS system, define a port for IMS Connect to listen for inbound requests. This scenario uses a port value of 1080. For more information about defining a port on IMS Connect for inbound requests, see [IMS Connect and TCP/IP communications](#) in the *IMS* documentation.
2. Create a server.

Enter the following command from the `<installation_dir>/bin` directory. For example, `/usr/lpp/IBM/zosconnect/v3r0/bin`:

```
zosconnect create phonebook --template=zosconnect:sampleImsPhonebook
```

The following artifacts are created:

- A contacts API archive file, `contacts.aar`, in the directory `<WLP_USER_DIR>/servers/phonebook/resources/zosconnect/apis`.
- A service archive file, `phonebook.sar`, in the directory `<WLP_USER_DIR>/servers/phonebook/resources/zosconnect/services`.
- A connection profile, `ims-connections.xml`, in the directory `<WLP_USER_DIR>/servers/phonebook/resources/imsmobile-config/connections`.
- An interactions profile, `ims-interactions.xml`, in the directory `<WLP_USER_DIR>/servers/phonebook/resources/imsmobile-config/interactions`.
- A `server.xml` configuration file in the directory `<WLP_USER_DIR>/servers/phonebook`.

3. Customize the z/OS Connect EE server configuration file.

The server.xml file is created with the following definition:

- Feature imsmobile:imsmobile-2.0

Update the value for httpPort if necessary. Security is disabled.

For your IMS connection you need to configure the following artifacts:

- In ims-connections.xml, replace the host and port values with the hostname or IP address of the LPAR hosting IMS Connect and the port IMS Connect is listening on for inbound requests.
- In ims-interactions.xml, replace the value for imsDatastoreName with your IMS data store name.

What to do next

Follow the steps in [“Test the connection to IMS” on page 73](#)

Test the connection to IMS

You can test your connection to IMS by invoking the contacts API directly from z/OS Connect EE API toolkit by using the “Try it out!” function in the Swagger UI that is embedded in the editor.

Procedure

To test your connection to IMS using the contacts API:

1. Start IMS and ensure that IMS Connect is listening on the expected port.
2. Start your z/OS Connect EE server. For more information, see [“Starting and stopping z/OS Connect EE” on page 471](#).
 - a) Check the messages.log file for the following messages that confirm that the service is installed.
BAQR7043I: z/OS Connect EE service archive phonebook installed successfully.
 - b) Check the messages.log file for the following messages that confirm that the API is installed.
BAQR7000I: z/OS Connect API archive file contacts installed successfully.

3. In the **z/OS Connect EE Servers** view under the **APIs** folder, double-click on the contacts API.

The API is opened in the Swagger UI.

4. Click **List Operations** to see available operations in the API.

contacts

This is an API for managing contacts.

The screenshot shows the 'contacts' API endpoint in the z/OS Connect EE Swagger UI. The table lists four operations: POST /contacts, DELETE /contacts/{lastName}, GET /contacts/{lastName}, and PUT /contacts/{lastName}. The 'List Operations' button is highlighted with a red arrow.

default	
POST	/contacts
DELETE	/contacts/{lastName}
GET	/contacts/{lastName}
PUT	/contacts/{lastName}

[BASE URL: /phonebook , API VERSION: 1.0.0]

Figure 12. Operations in the API

5. Test adding a contact record by clicking **POST**.
 - a) Click the **Example Value** box to copy the request message format into your contacts POST request.
 - b) Specify the last name, first name, zip code, and extension.

Parameters		
Parameter	Value	Description
postPhonebookService_request	<input type="button" value="-"/> <div style="border: 1px solid #ccc; padding: 5px; width: 200px;"> IVTNO_INPUT_MSG <input type="button" value="-"/> IN_TRANCODE IVTNO IN_LAST_NAME Smith IN_FIRST_NAME John IN_EXTENSION 000-0000 IN_ZIP_CODE 00000 </div>	request body
	Parameter content type:	<input type="button" value="application/json"/>
Authorization <input type="button" value="Try it out!"/> <input type="button" value="Hide Response"/>		

Figure 13. Testing the POST request

c) Click **Try it out!**.

Information about the request URL, request headers, response body, response code, and response headers are provided. The response body contains the output message. The OUT_MESSAGE would contain one of the following messages:

- ENTRY WAS ADDED
- ADDITION OF ENTRY HAS FAILED (this message indicates that the IN_LAST_NAME value you specified already exists)

Results

You have verified your connection to IMS is working.

Create a server and connect to an IMS database

This task shows you how to create a z/OS Connect EE server and configure a connection to an IMS database.

Before you begin

Ensure that the following tasks are complete:

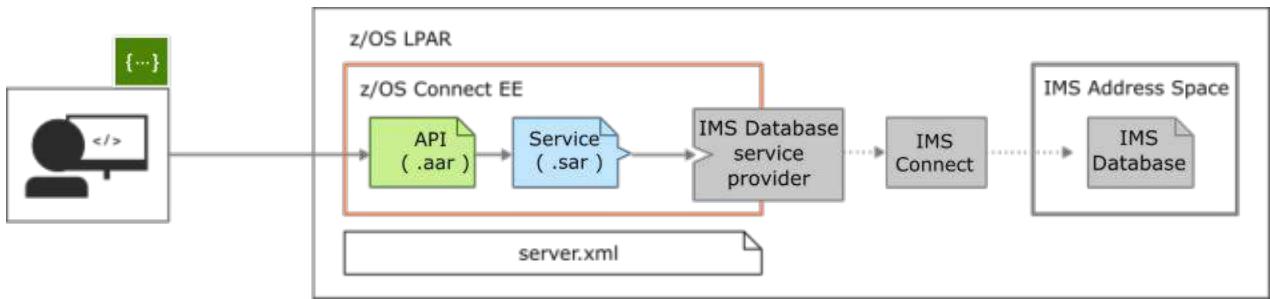
1. Install and configure the IMS IVP sample application database. The procedure is described in the scenario “[Prepare the sample IMS database](#)” on page 62.
2. Install the latest version of z/OS Connect EE. For more information, see “[Installing z/OS Connect EE](#)” on page 196.

About this task

This task guides you through the following:

- Creating a z/OS Connect EE server, and configuring it for use with an IMS database service.

- Configuring an IMS database connection between your z/OS Connect EE server and an IMS database.



Procedure

To configure z/OS Connect EE for IMS database services:

- On your IMS system, define a port for IMS Connect to listen for inbound requests. For more information about defining a port on IMS Connect for inbound requests, see [IMS Connect and TCP/IP communications](#) in the *IMS* documentation.
- Use the following command to create a server:

```
zosconnect create imsDatabase --template=zosconnect:sampleImsDatabase
```

The following artifacts are created:

- A `server.xml` configuration file in the directory `<WLP_USER_DIR>/servers/imsDatabase`.
- An API archive file `dbcontacts.aar`, in the directory `<WLP_USER_DIR>/servers/imsDatabase/resources/zosconnect/apis`
- A service archive file `dbPhonebook.sar`, in the directory `<WLP_USER_DIR>/servers/imsDatabase/resources/zosconnect/services`.

- Customize the z/OS Connect EE server configuration file.

The `server.xml` file is created with the following definition:

- Feature `zosconnect:dbService-1.0`

Update the value for `httpPort` if necessary. Security has been disabled.

For your IMS database connection you need to configure the following:

- In `server.xml`, update the `<connectionFactory>` element that has the id `IMSIIVPCF` with your IMS database connection information. For `databaseName`, specify the name of your PSB. For `datastoreName`, specify the hostname or IP address of your IMS connect. Update `portNumber` with the port IMS Connect is listening on for inbound requests.

If you have security enabled for your IMS, you must also specify the attributes `username` and `password`. To learn more, see [“IMS database service security process flow” on page 258](#).

For more information on configuring your own `server.xml` file, follow the instructions in [“Configuring for the IMS database service provider” on page 263](#).

What to do next

[“Test the connection to an IMS database” on page 76](#)

Related concepts

[“Prepare the sample IMS database” on page 62](#)

The IMS installation verification program (IVP) sample application database is used by the z/OS Connect EE scenarios that use the IMS database service provider. This sample IVP database must be installed in your target IMS region to follow this guide.

Test the connection to an IMS database

You can test your connection to an IMS database by invoking the provided IMS database contacts API directly from z/OS Connect EE API toolkit by using the "Try it out!" function in the Swagger UI that is embedded in the editor.

Procedure

To test your connection to IMS using the contacts API:

1. Start IMS and ensure that IMS Connect is listening on the expected port.
2. Start your z/OS Connect EE server. For more information, see [“Starting and stopping z/OS Connect EE” on page 471](#).
 - a) Check the messages.log file for the following messages that confirm that the API is installed.
BAQR7000I: z/OS Connect API archive file dbcontacts installed successfully.
 - b) Check the messages.log file for the following messages that confirm that the service is installed.
BAQR7043I: z/OS Connect EE service archive dbPhonebook installed successfully.

3. In the **z/OS Connect EE Servers** view under the **APIs** folder, double-click on the IMS database API. The API is opened in the Swagger UI.
4. Click **List Operations** to see available operations in the API.

dbcontacts

The screenshot shows the Swagger UI interface for the dbcontacts API. At the top, there's a navigation bar with 'Show/Hide' and 'List Operations' buttons. Below that is a header with 'dbcontacts' and a 'GET' button. Underneath the header, the URL '[BASE URL: /dbcontacts , API VERSION: 1.0.0]' is displayed. The main content area shows the response class 'Response Class (Status 200)' and the status code 'OK'. There are two tabs: 'Model' and 'Example Value'. The 'Example Value' tab is active, showing a JSON response structure. The JSON is as follows:

```
{ "response": { "result": [ { "LASTNAME": "string", "FIRSTNAME": "string", "EXTENSION": "string", "ZIPCODE": "string" } ] } }
```

Figure 14. Operations in the API

5. Test querying a contact record by clicking **GET**.
 - a) The **Example Value** box opens, demonstrating the response message payload format from your database GET request.

This screenshot shows the 'Example Value' tab for the GET operation in the Swagger UI. It displays the JSON response structure shown in Figure 14. At the bottom of the screen, there is a 'Response Content Type' dropdown set to 'application/json'.

5. Test querying a contact record by clicking **GET**.
 - b) Under the **Parameters** list, specify a **lastName** value of LAST1

c) Click **Try it out!**

Information about the request URL, request headers, response body, response code, and response headers are provided.

You will receive the following response body.

```
{  
  "response": {  
    "result": [  
      {  
        "LASTNAME": "LAST1",  
        "ZIPCODE": "D01/r01",  
        "FIRSTNAME": "FIRST1",  
        "EXTENSION": "8-111-1111"  
      }  
    ]  
  }  
}
```

Results

You have verified your connection to an IMS database is working.

What to do next

[Create an IMS database service.](#)

Create a server to connect to IBM MQ

Create a z/OS Connect EE server and configure it to connect to an IBM MQ queue manager on the same z/OS LPAR for a two-way service.

Before you begin

Ensure that the following tasks are complete:

- Compile and test the sample IBM MQ stock query application, and create the necessary IBM MQ queue definitions. For more information, see [“Prepare the sample IBM MQ stock query application” on page 63](#).
- Install z/OS Connect EE v3.0 or later. For more information, see [“Installing z/OS Connect EE” on page 196](#).

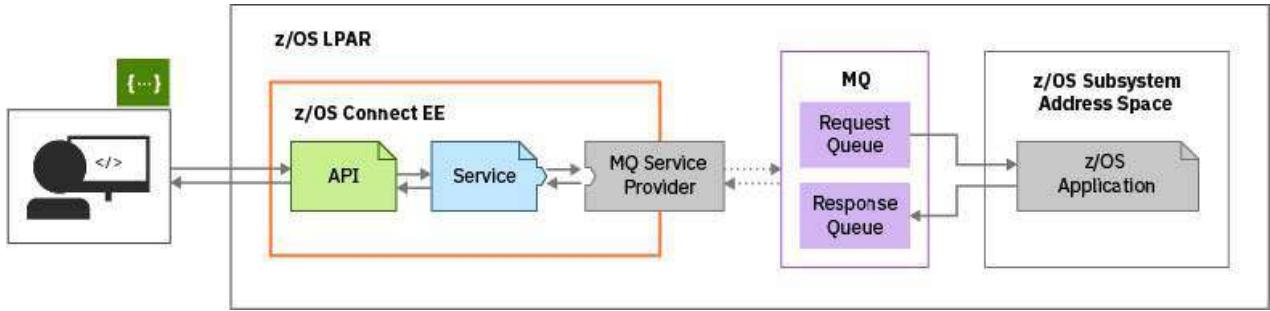
Pre-built services and an API are provided to simplify testing of this scenario.

About this task

This task guides you through the following steps:

- Creating a z/OS Connect EE server with ready-to-use services and a stock manager API for the IBM MQ stock query example application.
- Configure the z/OS Connect EE server's started task to include the IBM MQ libraries.
- Test the connection to IBM MQ.

This task does not configure security, so that you can focus on the specifics of configuring the IBM MQ service provider.



Procedure

- Enter the following command to create a z/OS Connect EE server:

```
zosconnect create stockManager --template=zosconnect:sampleMqStockManager
```

The following artifacts are created:

- A stock manager API archive file, `stockManager.aar`, in the directory `<WLP_USER_DIR>/servers/stockManager/resources/zosconnect/apis`.
- A stock query service archive file, `stockQuery.sar`, in the directory `<WLP_USER_DIR>/servers/stockManager/resources/zosconnect/services`.
- A `server.xml` configuration file in the directory `<WLP_USER_DIR>/servers/stockManager` with the `zosconnect:mqService-1.0` feature and template JMS artifacts included.

- Customize the z/OS Connect EE server configuration file, `server.xml`:

- Locate the UNIX System Services Components directory of the version of IBM MQ that you want to connect to.

For example, this directory might be called `/usr/lpp/mqm/V9R1M0/`. The directory contains a `java/lib` subdirectory, which contains several native libraries (.so files).

Note: If you are going to connect to multiple versions of queue manager, use the libraries from the most recent version.

- Replace the two instances of `MQ_INSTALL_ROOT` in `server.xml` with the value obtained in the previous step.

For example, if the directory you located in step “2.a” on page 78 is `/usr/lpp/mqm/V9R1M0/java/lib`, the configuration file should contain the following specification:

```
<variable name="wmqJmsClient.rar.location"
          value="/usr/lpp/mqm/V9R1M0/java/lib/jca/wmq.jmsra.rar"/>
<wmqJmsClient nativeLibraryPath="/usr/lpp/mqm/V9R1M0/java/lib"/>
```

- In the `jmsConnectionFactory` element change `QMGR_NAME` to the name of a local queue manager.

- Set up the STEPLIB of your server to include the IBM MQ libraries.

This step ensures that the native libraries specified in step “2.b” on page 78 can connect to the queue manager. Edit the JCL that starts your server, to include the following STEPLIB elements:

```
//STEPLIB DD DSN=HLQ.SCSQAUTH,DISP=SHR
//           DD DSN=HLQ.SCSQANLE,DISP=SHR
```

In this code sample, `HLQ` is the high-level qualifier of the data sets containing the IBM MQ installation. If you connect to multiple versions of IBM MQ from the same server, use the data sets from the most recent version.

- Enable the TXRRS authorized service by following the procedure described in “[Configuring the Liberty Angel process and z/OS authorized services](#)” on page 465.

This service must be available for the IBM MQ service provider to function correctly.

- Start the `stockManager` server. For more information, see “[Starting and stopping z/OS Connect EE](#)” on page 471.

6. Validate that the TXRRS authorized service has been set up correctly by checking the server logs located at <WLP_USER_DIR>/servers/stockManager/logs/messages.log.

These are ASCII files, and should contain output similar to the following example:

```
A CWWKE0001I: The server test has been launched.  
I CWWKB0103I: Authorized service group LOCALCOM is available.  
I CWWKB0103I: Authorized service group PRODMGR is available.  
I CWWKB0103I: Authorized service group SAFCRED is available.  
I CWWKB0103I: Authorized service group TXRRS is available.  
I CWWKB0103I: Authorized service group WOLA is available.  
I CWWKB0103I: Authorized service group ZOSDUMP is available.  
I CWWKB0103I: Authorized service group ZOSWLM is available.  
I CWWKB0103I: Authorized service group CLIENT.WOLA is available.  
I CWWKB0108I: IBM CORP product z/OS Connect version 03.00 successfully registered with  
z/OS
```

Check the output to verify that the TXRRS authorized service group is available, as in the preceding example.

What to do next

Verify the connection by following the steps in [“Test the connection to IBM MQ” on page 79](#), then follow the steps in [“Create a two-way IBM MQ service” on page 96](#) to create a service.

Test the connection to IBM MQ

You can test the connection to IBM MQ by invoking the stockManager API directly from the z/OS Connect EE API toolkit by using the “Try it out!” function in the Swagger UI that is embedded in the editor.

Procedure

1. Start your IBM MQ queue manager.
2. Start the stock query application, see [“Prepare the sample IBM MQ stock query application” on page 63](#).
3. Start your z/OS Connect EE server. For more information, see [“Starting and stopping z/OS Connect EE” on page 471](#).
 - a) Check the messages.log file for the following message to confirm that the service is installed.

BAQR7043I: z/OS Connect EE service archive stockQuery installed successfully.

- b) Check the messages.log file for the following message to confirm that the API is installed.

BAQR7000I: z/OS Connect API archive file stockmanager installed successfully.

4. In the **z/OS Connect EE Servers** view of the API toolkit, under the **APIs** folder, double-click the stockmanager API.
The API is opened in the Swagger UI.
5. Click **List Operations** to see available operations in the API.

stockmanager

A stock manager application based on MQ

stockmanager

Show/Hide | List Operations | Expand Operations

GET /stock/items/{item-id}

[BASE URL: /stockmanager , API VERSION: 1.0.0]

Figure 15. Operations in the API

6. Query the state of the stock with item ID 2033.
 - a) Specify a value of 2033 for the item-id and click **GET**.

GET /stock/items/{item-id}

Response Class (Status 200)
OK

Model Example Value

```
{
  "SQRESP": {
    "ITEM_ID": 0,
    "ITEM_DESC": "string",
    "ITEM_COUNT": 0,
    "ITEM_COST": 0
  }
}
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
Authorization			header	string
item-id	2033		path	string

Try it out!

Figure 16. Testing the GET request

b) Click **Try it out!**.

Information about the request URL, request headers, response body, response code, and response headers are provided.

An HTTP response code of 200 is expected with the following response body:

```
{
  "SQRESP": {
    "ITEM_ID": 2033,
    "ITEM_DESC": "A description.",
    "ITEM_COST": 4,
    "ITEM_COUNT": 500
  }
}
```

Results

You have verified that your newly created stockManager API, using the stock query service running in z/OS Connect EE, has successfully called the stock query application.

What to do next

Follow the steps in “[Create a two-way IBM MQ service](#)” on page 96 to create a service.

Create a server to connect to Db2

This task shows you how to create a z/OS Connect EE server and configure a connection to a Db2.

Before you begin

Ensure that the following tasks are complete:

1. Install and configure Db2 native REST services. The procedure is described in “[Enable and create Db2 native REST services](#)” on page 66.
2. Install z/OS Connect EE v3.0.29 or later. For more information, see “[Installing z/OS Connect EE](#)” on page 196.

3. Install the API Toolkit, see [“Installing z/OS Explorer and the z/OS Connect EE API toolkit” on page 201](#).

Pre-built services and an API are provided to simplify testing of this scenario.

About this task

This task guides you through the following:

- Creating a z/OS Connect EE server with ready-to-use services and an API for the employee Db2 native REST services.
- Configuring a connection between your z/OS Connect EE server and Db2.
- Testing the connection to Db2 using the supplied API.

Procedure

1. Enter the following command to create a server:

```
zosconnect create db2ProjectManager --  
template=zosconnect:sampleDb2ProjectManager
```

The following artifacts are created:

- An projectManager API archive file, `projectManager.aar`, in the directory `<WLP_USER_DIR>/servers/db2ProjectManager/resources/zosconnect/apis`.
- Three service archive files, `employeeList.sar`, `employeeDetails.sar`, and `employeeUpdate.sar`, in the directory `<WLP_USER_DIR>/servers/db2ProjectManager/resources/zosconnect/services`.
- A `server.xml` configuration file in the directory `<WLP_USER_DIR>/servers/db2ProjectManager`.

2. Customize the z/OS Connect EE `server.xml` configuration file.

- a) Update the REST client connection element to define a connection to Db2:

```
<zosconnect_zosConnectServiceRestClientConnection id="db2Conn" host="localhost"  
port="1234" />
```

- i) If your server and Db2 instance are on different LPARs, replace the host value `localhost` with the host name or IP address of the LPAR hosting your Db2 instance.
- ii) Replace the port value `1234` with the port that your Db2 instance is configured to use.
- iii) Enable security if required by Db2 configuration. For more information, see the [“Configuring security for a REST client connection to Db2” on page 282](#) topic.

- b) Update the `httpPort` for inbound connections to z/OS Connect EE in the `httpEndpoint` element, if necessary:

```
<httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="9080" httpsPort="-1"/>
```

What to do next

Test your newly created server.

Test the Db2 connection

You can test your connection to your Db2 instance by invoking the `projectManager` API directly from z/OS Connect EE API toolkit by using the “Try it out!” function in the Swagger UI that is embedded in the editor.

Procedure

1. Start your z/OS Connect EE server. For more information, see [“Starting and stopping z/OS Connect EE” on page 471](#).

- a) Check the messages.log file for the following messages that confirm that the services are installed.

```
BAQR7043I: z/OS Connect EE service archive employeeList installed successfully.
BAQR7043I: z/OS Connect EE service archive employeeDetails installed successfully.
BAQR7043I: z/OS Connect EE service archive employeeUpdate installed successfully.
```

- b) Check the messages.log file for the following message that confirms that the API is installed.

```
BAQR7000I: z/OS Connect API archive file projectManager installed successfully.
```

2. Create a host connection to your z/OS Connect EE server. For more information, see [“Connecting to a z/OS Connect EE server” on page 603](#).
3. In the **z/OS Connect EE Servers** view under the **APIs** folder, double-click on the catalog API.
The API is opened in the Swagger UI.
4. Click **List Operations** to see available operations in the API.

Figure 17. Operations in the API

5. Test listing employees by clicking **GET /employees**.

- a) Click **Try it out!**

Information about the request URL, request headers, response body, response code, and response headers are provided. The response body starts with:

```
{
  "ResultSet Output": [
    {
      "employeeNumber": "000150",
      "firstName": "BRUCE",
      "lastName": "ADAMSON",
      "department": "D11"
    },
    ...
  ],
  "StatusDescription": "Execution Successful"
}
```

Results

You have verified your connection to Db2 is working.

What to do next

Follow the steps [“Create a Db2 service” on page 108](#) to see how to create a service for yourself.

Create services

These tasks show you how to quickly get started with creating, deploying, and testing z/OS Connect EE services by using supplied samples.

A service archive (.sar) file contains the information that is needed by a z/OS Connect EE service provider to install and provide the service, and to enable the service as a JSON asset. A service archive is also required to build APIs in the z/OS Connect EE API toolkit.

In some of the sample scenarios, you can use the Swagger UI to test the service because these scenarios supply the API which calls the service, so the server is created using a sample template that includes services and an API. The scenarios delete the service and recreate it as a demonstration of how the

service was initially created. Therefore the newly recreated service is being tested by calling the supplied API. In a typical development environment where services are created before the API, the API toolkit cannot be used to test the service. In this case, a REST client, such as postman, is required.

Create a CICS service

This scenario shows how you can recreate the CICS inquireSingle service for the catalog manager example application using the API toolkit.

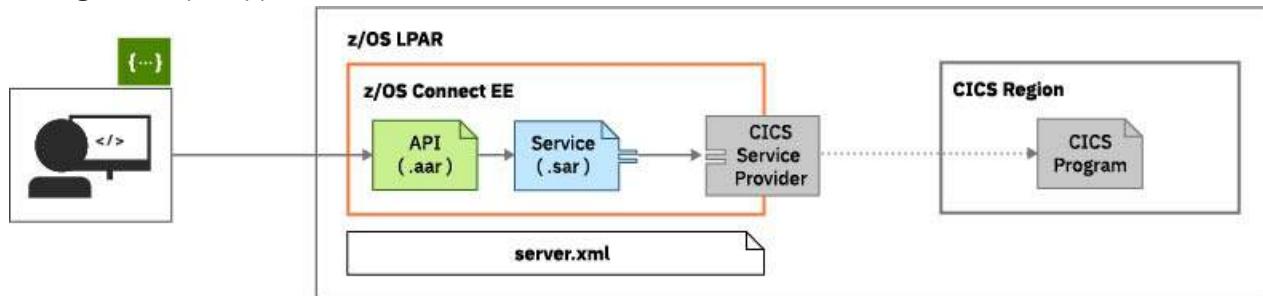
Before you begin

To create and test the inquireSingle service ensure that the following tasks are complete:

1. Install and configure the CICS catalog manager example application. The procedure is described in the scenario “[Prepare the sample CICS application](#)” on page 59
2. Install z/OS Connect EE v3.0 or later. For more information, see “[Installing z/OS Connect EE](#)” on page 196.
3. Configure and test your IPIC connection to CICS. The procedure is described in the scenario “[Create a server and connect to CICS](#)” on page 69 and installs relevant artifacts using the supplied configuration template `sampleCicsIpicCatalogManager`. This includes the inquireSingle service.
4. Use the z/OS Connect EE API toolkit to Stop the inquireSingle service, and then Remove the inquireSingle service.

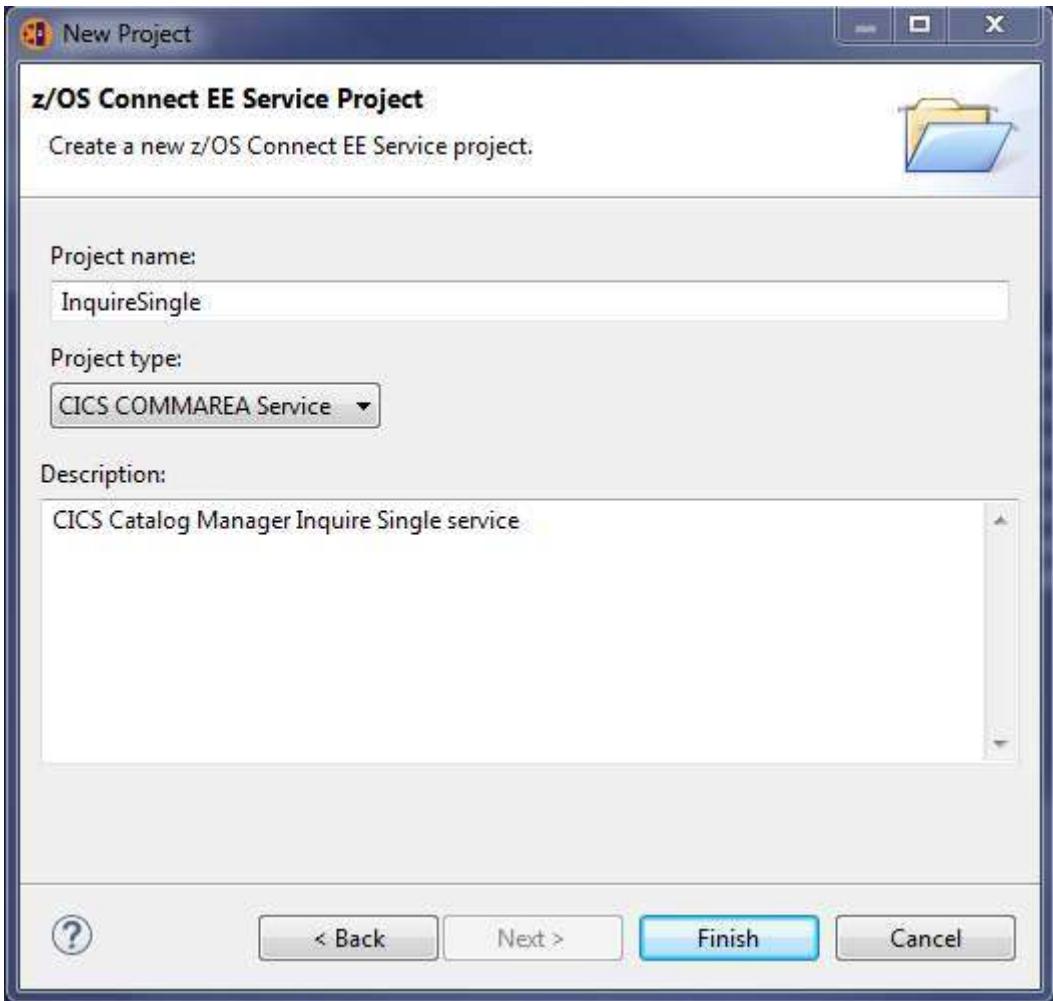
About this task

In this scenario we use the API toolkit running on a Windows workstation to recreate the CICS inquireSingle service. The service archive file uses an existing CICS COBOL copybook from the catalog manager example application.



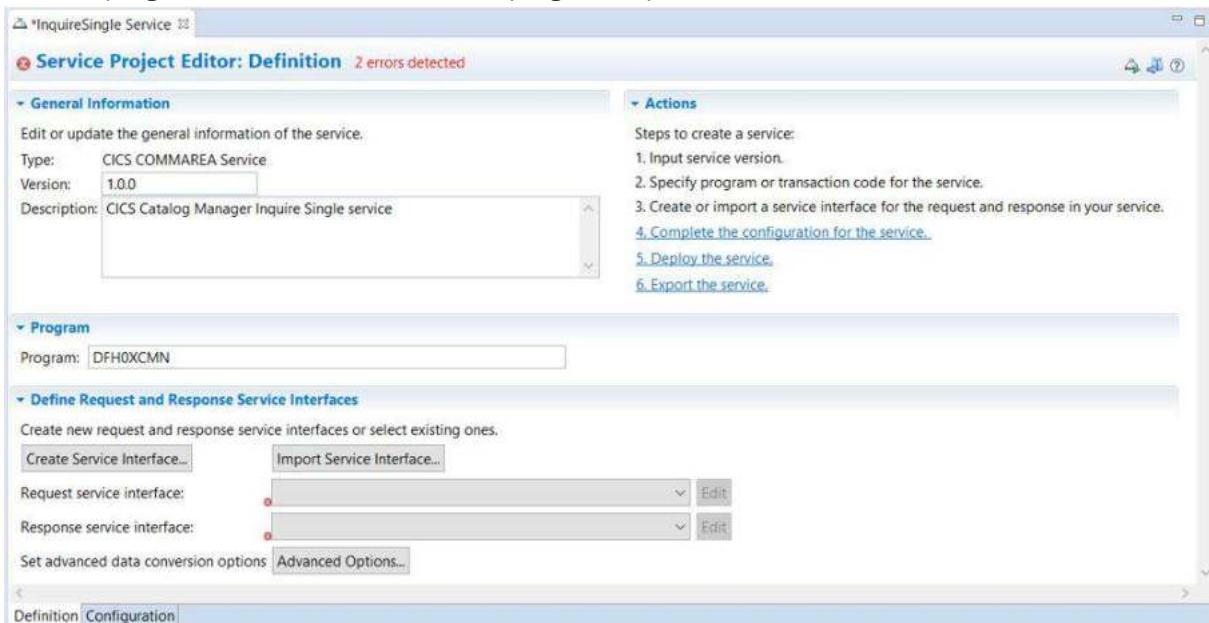
Procedure

1. Start IBM Explorer for z/OS and open the z/OS Connect Enterprise Edition perspective.
2. Right click in the project explorer window and select **File > New... > Project**. Select the **z/OS Connect EE Service Project** wizard.
3. Fill in the sections for the new service project and click **Finish**.



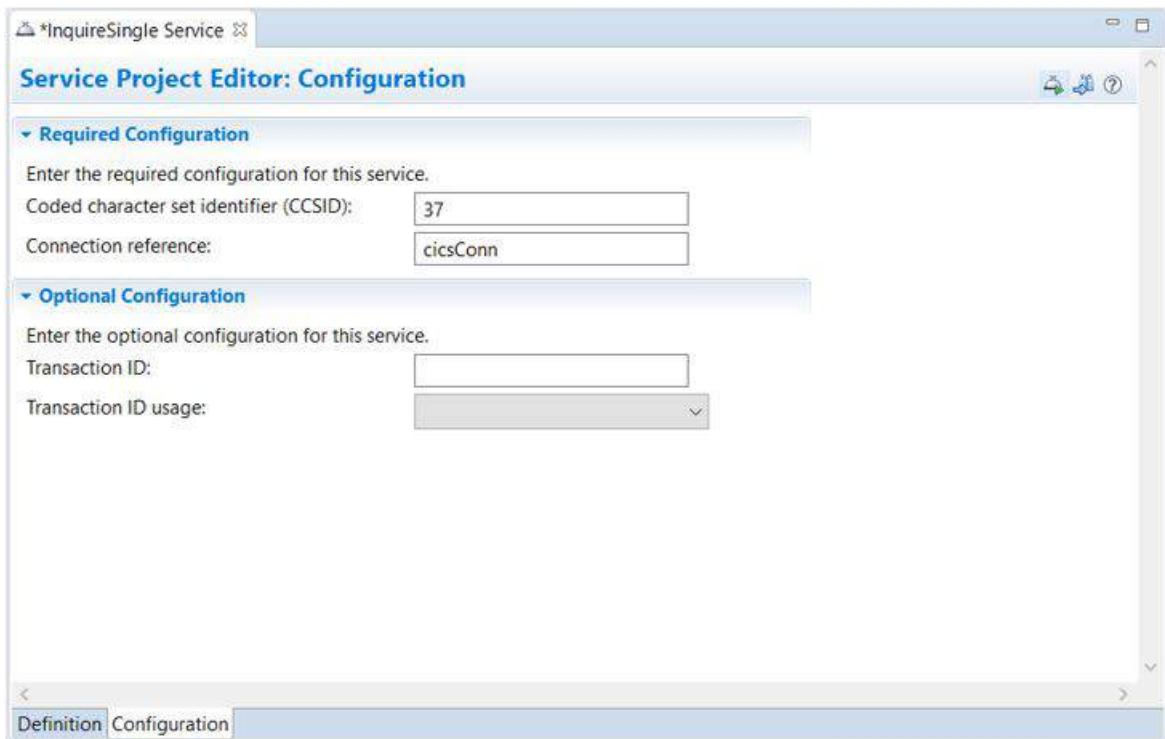
4. The z/OS Connect EE Service Project Editor dialog opens.

Add the program name DFH0XCMN to the program: input area, as shown.



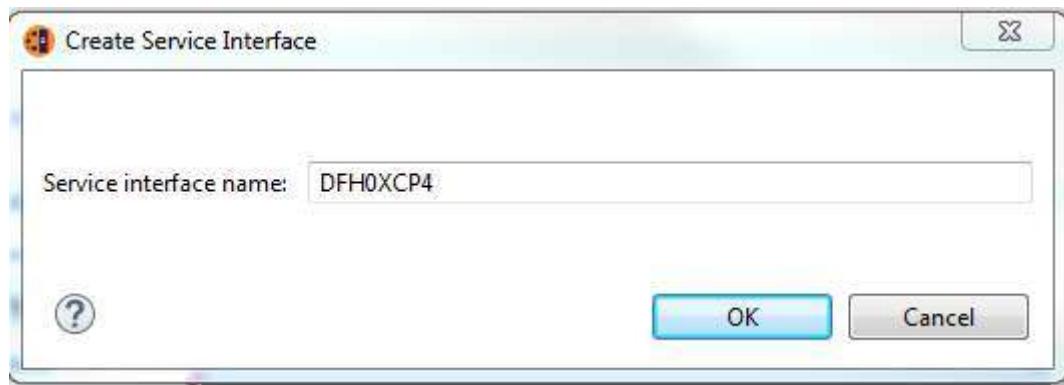
5. Click the Configuration tab at the bottom of the screen.

Add the connection reference cicsConn



6. Click the **Definition** tab at the bottom of the screen.
7. Click **Create Service Interface**.

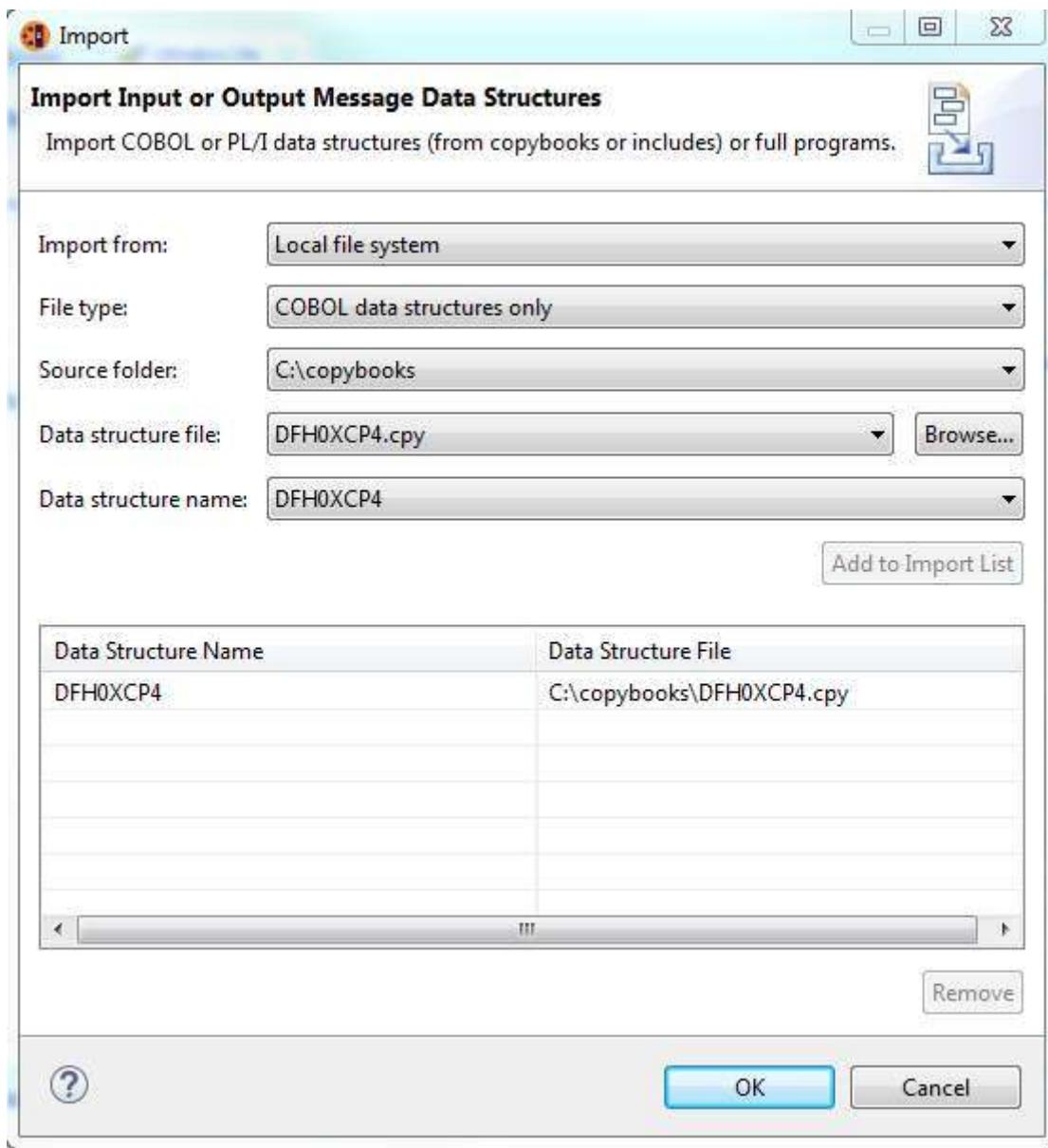
Add the service interface name DFH0XCP4 and click **OK**.



The screenshot shows the 'Service Interface Editor' window with the title bar 'InquireSingle Service DFH0XCP4'. The main area contains a table with the following columns: Fields, Include, Interface rename, Default Field Value, Data Type, Field Length, and Start Byte. A single row is present in the table, labeled 'COMMAREA'. Below the table is a search bar and a set of toolbar icons.

Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA						

8. You have created a new service interface file. In the Service Interface editor, click **Import COBOL or PLI data Structures into the service interface** .
9. Locate the DFH0XCP4 COBOL copybook and import it from the file system. To import the copybook directly from your CICS installation, select **z/OS system** from the **Import from:** list and locate the copybook from the **Source PDS:** list. If the copybook is on your local workstation, select **Local file system** from the **Import from:** list and locate the copybook from the **Source folder:** list.



10. Click **Add to Import List** to import the DFH0XCP4 data structure from the copybook, and click **OK**.

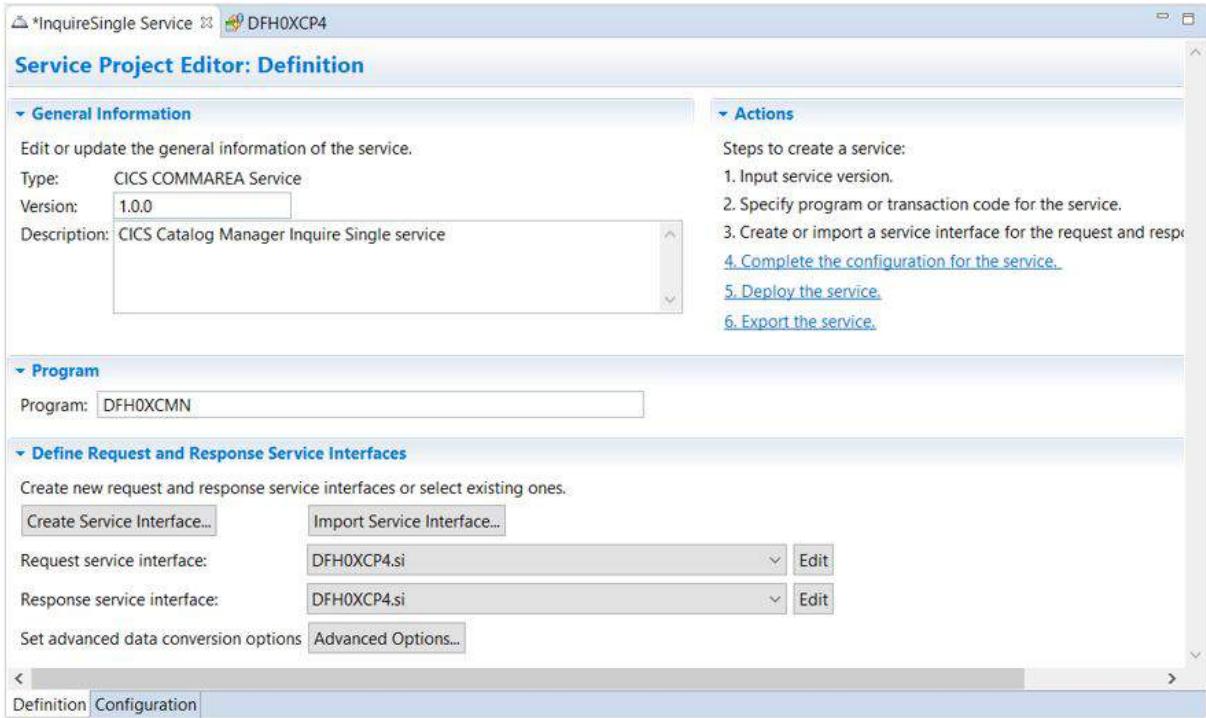
InquireSingle Service *DFH0XCP4

Service Interface Editor

Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

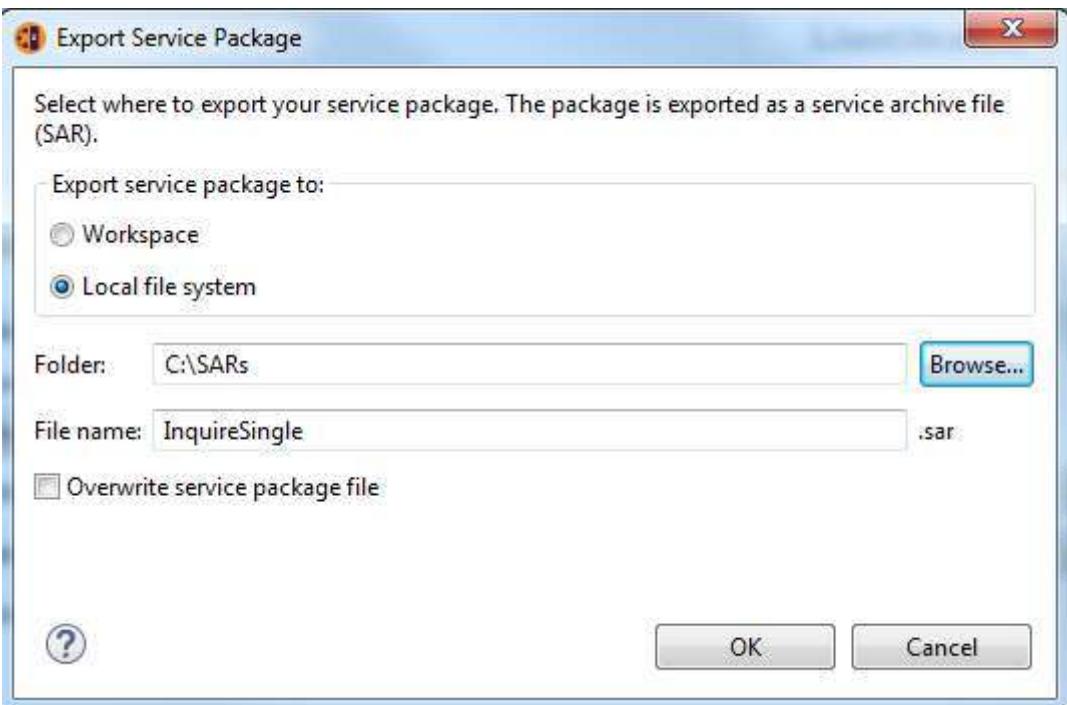
Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA						
DFH0XCP4						
CA_REQUEST_ID	<input checked="" type="checkbox"/>	CA_REQUEST_ID		CHAR	6	1
CA_RETURN_CODE	<input checked="" type="checkbox"/>	CA_RETURN_CODE		DECIMAL	2	7
CA_RESPONSE_MESSAGE	<input checked="" type="checkbox"/>	CA_RESPONSE_MESSAGE		CHAR	79	9
CA_INQUIRE_SINGLE	<input checked="" type="checkbox"/>	CA_INQUIRE_SINGLE		STRUCT	71	88
CA_ITEM_REF_REQ	<input checked="" type="checkbox"/>	CA_ITEM_REF_REQ		DECIMAL	4	88
FILL_0	<input checked="" type="checkbox"/>	FILL_0		DECIMAL	4	92
FILL_1	<input checked="" type="checkbox"/>	FILL_1		DECIMAL	3	96
CA_SINGLE_ITEM	<input checked="" type="checkbox"/>	CA_SINGLE_ITEM		STRUCT	60	99
CA_SNGL_ITEM_REF	<input checked="" type="checkbox"/>	CA_SNGL_ITEM_REF		DECIMAL	4	99
CA_SNGL_DESCRIPTION	<input checked="" type="checkbox"/>	CA_SNGL_DESCRIPTION		CHAR	40	103
CA_SNGL_DEPARTMENT	<input checked="" type="checkbox"/>	CA_SNGL_DEPARTMENT		DECIMAL	3	143
CA_SNGL_COST	<input checked="" type="checkbox"/>	CA_SNGL_COST		CHAR	6	146
IN_SNGL_STOCK	<input checked="" type="checkbox"/>	IN_SNGL_STOCK		DECIMAL	4	152
ON_SNGL_ORDER	<input checked="" type="checkbox"/>	ON_SNGL_ORDER		DECIMAL	3	156

11. Unselect the Include checkboxes for FILL_0 and FILL_1 to hide these fields from the user of the service as they are for internal use by the CICS application.
12. Save your changes to the service interface DFH0XCP4.
13. Click **Definition** tab for the inquireSingle service. As there is only one service interface file, **DFH0XCP4.si** in the service interfaces folder for this project, it is automatically selected in the **Request service interface** and **Response service interface** fields.



Save the InquireSingle service.

14. Right click inquireSingle in the **Project Explorer** view. Click **z/OS Connect EE** > **Export z/OS Connect EE Service Archive**.



15. Click **Browse** to select the directory where you want to save the service archive file and click **OK**.

Results

The InquireSingle service archive file `inquireSingle.sar` is now ready to be deployed.

What to do next

Deploy and test your newly created InquireSingle service.

Deploy the CICS catalog manager service using the API toolkit

Deploy the CICS catalog manager inquireSingle service directly from z/OS Connect EE API toolkit by first creating a connection to the server.

Before you begin

Create a connection to your z/OS Connect EE server from z/OS Connect EE API toolkit. See [“Connecting to a z/OS Connect EE server” on page 603](#).

About this task

In the **Host Connections** view, you can add connections to z/OS Connect EE servers and credentials to store your user IDs and passwords.

Tip: If you don't see the **Host Connections** view, from the menu bar, click **Window > Manage Connections** to open the **Host Connections** view.

Procedure

In the **Project Explorer** view, right-click the service project and select **z/OS Connect EE > Deploy Service to z/OS Connect EE Server**. If you already have the service installed, perhaps by using the `sampleCicsIpicCatalogManager` template, you can select the Overwrite check box, or Stop and Remove the service before deploying your new one.

Create an IMS service

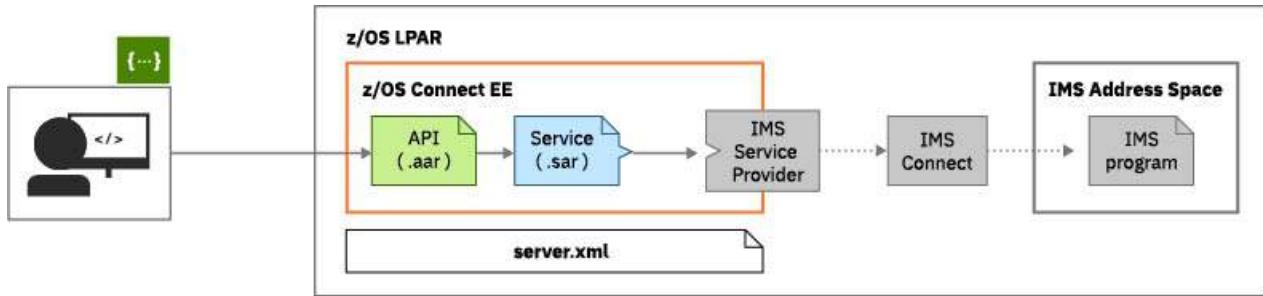
This scenario shows how you can use the API toolkit to create an IMS phonebook service for the phone book application.

Before you begin

Ensure that the following tasks are complete:

1. Install and configure the IMS phone book application on your IMS system. The procedure is described in the scenario [“Prepare the sample IMS application” on page 61](#)
2. Install z/OS Connect EE v3.0 or later. For more information, see [“Installing z/OS Connect EE” on page 196](#).
3. Configure and test your connection to IMS. The procedure is described in the scenario [“Create a server and connect to IMS” on page 72](#) and installs relevant artifacts by using the supplied configuration template `sampleImsPhonebook`, including the phonebook service.
4. Use the z/OS Connect EE API toolkit to Stop the phonebook service, and then Remove the phonebook service.

About this task



In this scenario, the API toolkit, running on a Windows workstation is used to re-create the IMS phonebook service. The service archive file uses an existing IMS COBOL copybook from the IMS phone book application.

This scenario uses the following values:

- IMS region = IMS1
- IMS program = PROG1
- IMS transaction code = IVTNO

Procedure

1. Copy the IMS phone book COBOL copybook structure to a folder on your workstation, for example, C:/COPYBOOKS/IMSPHBK.cpy, making sure that the asterisks are in column 7.

```
*      IMS PhoneBook Transaction (IVTNO) Sample
*
01  IVTNO_INPUT-MSG.
    02  IN-LL          PICTURE S9(3) COMP.
    02  IN-ZZ          PICTURE S9(3) COMP.
    02  IN-TRANCDE    PICTURE X(10).
    02  IN-COMMAND    PICTURE X(8).
    02  IN-LAST-NAME  PICTURE X(10).
    02  IN-FIRST-NAME PICTURE X(10).
    02  IN-EXTENSION   PICTURE X(10).
    02  IN-ZIP-CODE   PICTURE X(7).

01  IVTNO_OUTPUT-MSG.
    02  OUT-LL         PICTURE S9(3) COMP VALUE +0.
    02  OUT-ZZ         PICTURE S9(3) COMP VALUE +0.
    02  OUT-MESSAGE   PICTURE X(40) VALUE SPACES.
    02  OUT-COMMAND   PICTURE X(8) VALUE SPACES.
    02  OUT-LAST-NAME PICTURE X(10) VALUE SPACES.
    02  OUT-FIRST-NAME PICTURE X(10) VALUE SPACES.
    02  OUT-EXTENSION  PICTURE X(10) VALUE SPACES.
    02  OUT-ZIP-CODE  PICTURE X(7) VALUE SPACES.
    02  OUT-SEGNO     PICTURE X(4) VALUE SPACES.
```

2. Switch to the **z/OS Connect EE** perspective.
 - a) From the main menu, select **Window > Open Perspective > Other**
The **Select Perspective** wizard opens.
 - b) Click **z/OS Connect Enterprise Edition**.
 - c) Click **OK**.
The **z/OS Connect EE** perspective opens.
3. Select **File > New > Project**.
The New Project wizard opens.
4. Select **z/OS Connect Enterprise Edition > z/OS Connect EE Service Project**, and click **Next**.
5. Specify a project name, select the project type, and optionally provide a description.
 - a) Specify the project name, phonebook.
 - b) For project type, select **IMS Service**.

c) Click **Finish** to create the project.

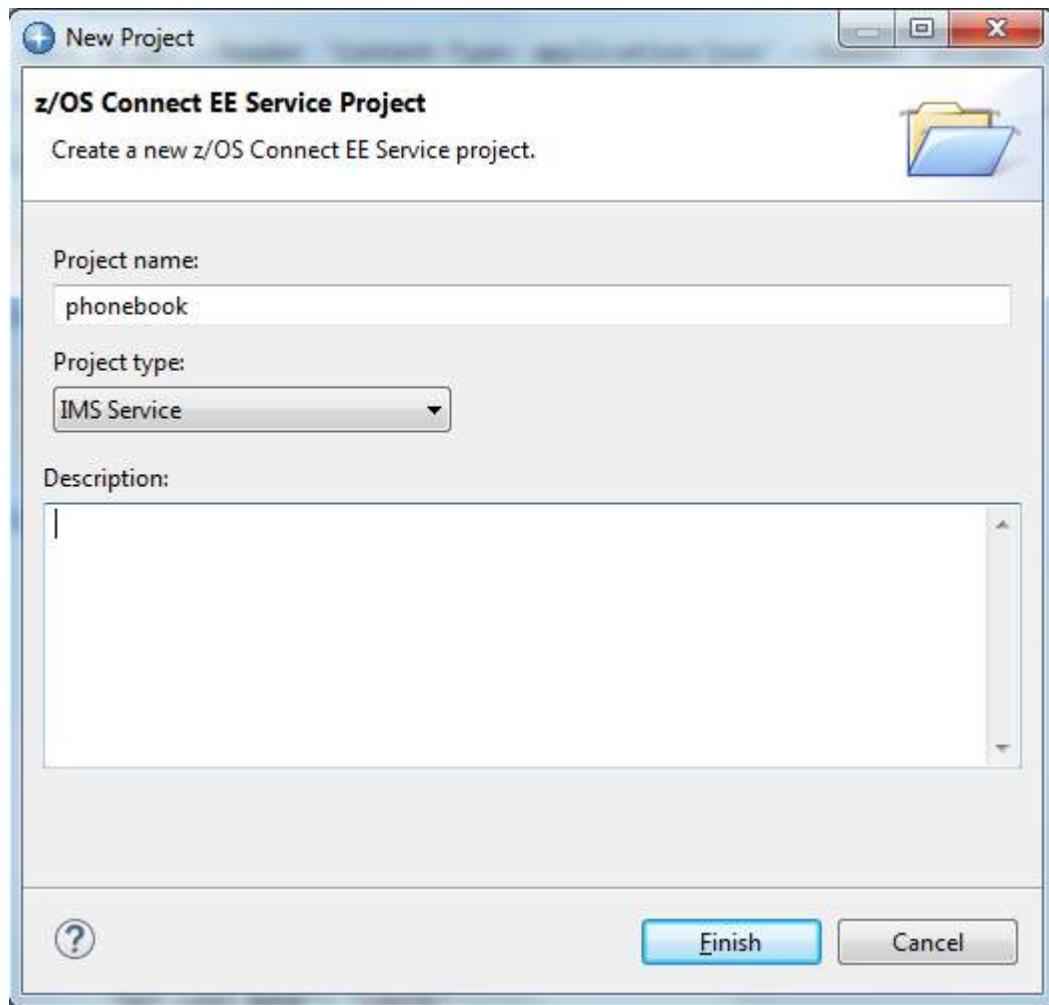


Figure 18. The New project wizard

The service project is created in the **Project Explorer** view. The service definition opens in the service project editor, where you can configure the service and define the service interface. The z/OS Connect EE **service project editor** opens, with some errors initially that indicate some required information is still missing.

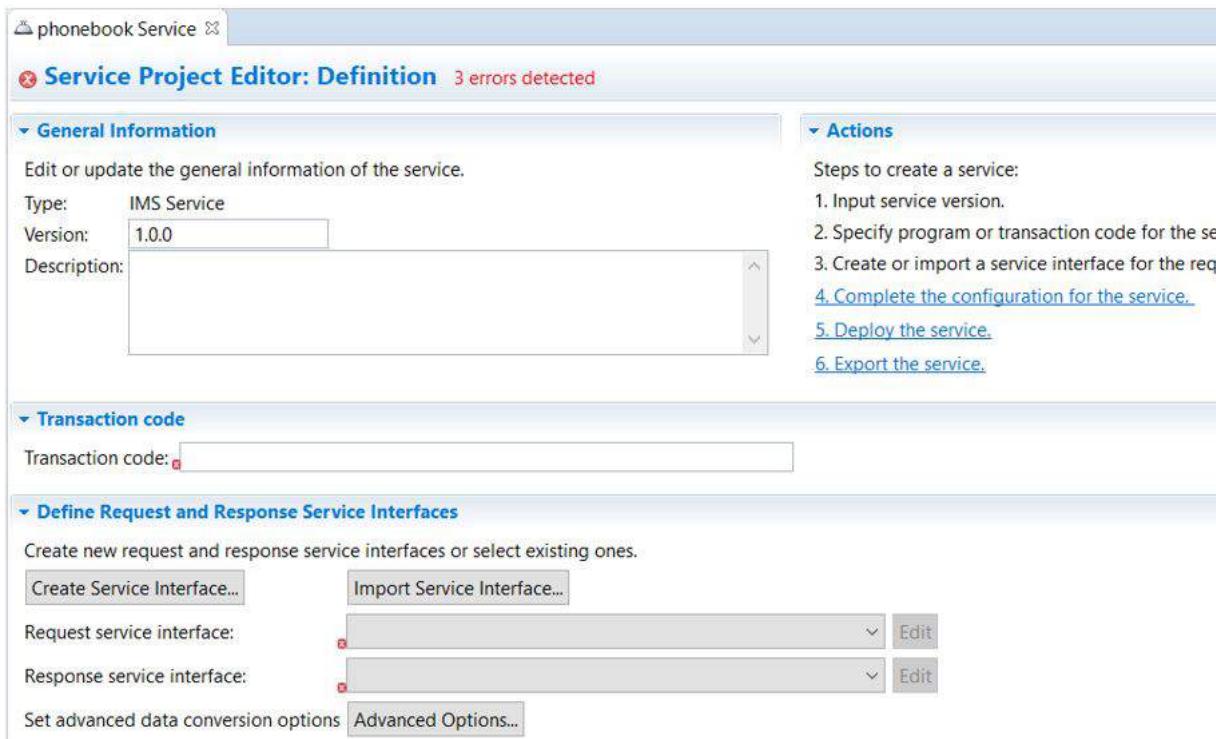


Figure 19. The definition page in the service project editor

6. For the transaction code to invoke, specify IVTNO.

In the following steps, you create the service interface for the request and response message. In this step, you import the COBOL data structure into this project and specify which fields are to be exposed for input and output messages.

7. In the **Define Request and Response Service Interfaces** section, click **Create Service Interface**.

In the following steps, you create a service interface for the request messages.

8. Specify a name such as phonebookRequest for the service interface. This service interface can later be identified as the request service interface for this service project. It can also be reused by other service projects.

9. Click the button to import the phone book copybook.

10. In the **Import** wizard, import the input and output message data structures.

a) From the **Import from:** list, select **Local file system**.

b) For the **Source folder:** field, select the folder where you copied the phonebook copybook to, for example, C:/COPYBOOKS

c) From the **File type:** list, select **COBOL data structures only**.

d) On the **Data structure file:** field, click **Browse**.

e) In the **Select File** dialog, navigate to the phone book copybook file IMSPHBK.cpy on your workstation. For example, C:/COPYBOOKS and click **Open**.

11. Specify the data structure for the input messages:

a) In the **Data structure name** list, select **IVTNO_INPUT_MSG**.

b) For the **Message name** list, <service_interface_name> is automatically selected.

c) For the **Segment name** list, ensure that **Segment 1** is selected.

d) Click **Add to Import List** to add this message data structure to the Import list.

e) Click **OK**.

The service interface editor opens.

12. Expand **Segment 1** and **IVTNO_INPUT_MSG** to see all fields in the data structure.

13. The phone book service requires most of the fields for input messages, except IN_LL and IN_ZZ. These are IMS internal fields.
- Exclude IN_LL and IN_ZZ by clicking the check boxes in the **Include** column to de-select them.

Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
phonebookRequest						
Segment 1						
IVTNO_INPUT_MSG						
IN_LL	<input type="checkbox"/>	IN_LL		SHORT	2	1
IN_ZZ	<input type="checkbox"/>	IN_ZZ		SHORT	2	3
IN_TRANCDE	<input checked="" type="checkbox"/>	IN_TRANCDE		CHAR	10	5
IN_COMMAND	<input checked="" type="checkbox"/>	IN_COMMAND		CHAR	8	15
IN_LAST_NAME	<input checked="" type="checkbox"/>	IN_LAST_NAME		CHAR	10	23
IN_FIRST_NAME	<input checked="" type="checkbox"/>	IN_FIRST_NAME		CHAR	10	33
IN_EXTENSION	<input checked="" type="checkbox"/>	IN_EXTENSION		CHAR	10	43
IN_ZIP_CODE	<input checked="" type="checkbox"/>	IN_ZIP_CODE		CHAR	7	53

Figure 20. The service interface editor with the request service interface for the phonebook service

- Click **File > Save** to save the request service interface.
- To define the service interface for the response message for the phone book application:

 - Click the **service.properties** tab to go to the overview page of your service project.
 - In the **Define Request and Response Service Interfaces** section, click **Create Service Interface**.
 - Specify a name such as phonebookResponse for the service interface.
 - Click the button to import the phone book copybook.
 - Again, select where you are importing the phone book copybook from.
 - Select the file type. For this tutorial, the default of **COBOL data structures only** is used.
 - For the **Data structure file** field, click **Browse**.
 - In the **Select File** dialog, navigate to the phone book copybook file IMSPHBK.cpy on your workstation and click **Open**.
 - Specify the data structure for the output messages:
 - In the **Data structure name** list, select **IVTNO_OUTPUT_MSG**.
 - For the **Message name** list, <**service_interface_name**> is automatically selected.
 - For the **Segment name** list, ensure that **Segment 1** is selected.
 - Click **Add to Import List** to add this message data structure to the Import list.
 - Click **OK**.
 The service interface editor opens.
 - Expand **Segment 1** and **IVTNO_OUTPUT_MSG** to see all fields in the data structure.
 - The phonebook service requires most of the fields for output messages, except OUT_LL and OUT_ZZ. These are IMS internal fields. Exclude OUT_LL and OUT_ZZ by clicking the check boxes in the **Include** column to deselect them.
 - Save your changes.

You have created two service interface files. In the Project Explorer view, two service interface files (.si files) are created in the service-interfaces/ folder in your service project.

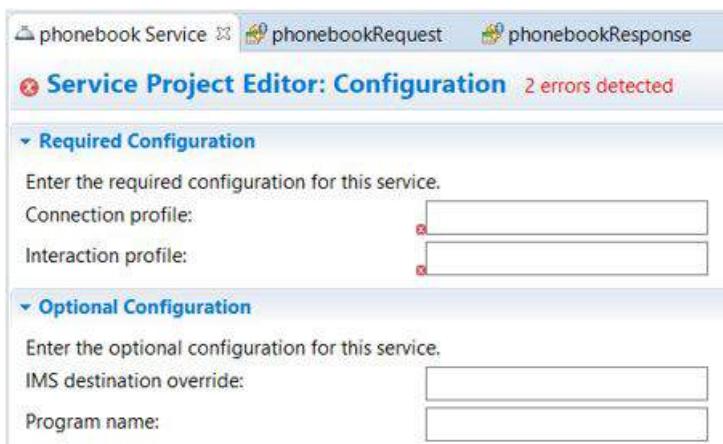
The next step is to go back to the service project editor to select this service interface file as the request and response service interface for this project.

 - Go back to the Overview tab of your service project.

24. Select the service interface files that you created as the interface to use for the request and the response.
 - a) For the **Request service interface** field, select the .si file that you created for the request, for example, phonebookRequest.si.
 - b) For the **Response service interface** field, select the .si file that you created for the response, for example, phonebookResponse.si.
25. Save your changes.

The final step is to enter the details of the required configuration for this service.

26. Select the Configuration tab of your service project.



27. In the **Connection profile:** field, enter the ID value that is associated with your imsmobile_imsConnection XML element in your ims-connections.xml file, for example, Connection1_CF. Your ims-connections.xml file is located under the resources/imsmobile-config/connections directory for your server.
28. In the **Interaction profile:** field, enter the ID value that is associated with your imsmobile_interaction XML element in your ims-interactions.xml file, for example, InteractionProperties1. Your ims-interactions.xml file is located under the resources/imsmobile-config/interactions directory for your server.
29. Save your changes.

Results

The IMS phonebook service is now ready to be deployed.

Deploy the phonebook service using the API toolkit

Deploy the phonebook service directly from z/OS Connect EE API toolkit by first creating a connection to the server.

Before you begin

Create a connection to your z/OS Connect EE server from the z/OS Connect EE API toolkit. See [“Connecting to a z/OS Connect EE server” on page 603](#).

About this task

In the **Host Connections** view, you can add connections to z/OS Connect EE servers and credentials to store your user IDs and passwords.

Tip: If you don't see the **Host Connections** view, from the menu bar, click **Window > Manage Connections** to open the **Host Connections** view.

Procedure

In the **Project Explorer** view, right-click the service project and select **z/OS Connect EE > Deploy Service to z/OS Connect EE Server**. If you already have the service installed, perhaps by using the sampleImsPhonebook template, you can select the Overwrite check box, or Stop and Remove the service before deploying your new one.

What to do next

[“Test the IMS phonebook service” on page 95.](#)

Test the IMS phonebook service

You can test your newly created service is working by invoking the contacts API directly from z/OS Connect EE API toolkit by using the “Try it out!” function in the Swagger UI that is embedded in the editor.

Procedure

To test your newly created and deployed phonebook service:

1. Start IMS and ensure that IMS Connect is listening on the expected port.
2. Start your z/OS Connect EE server. For more information, see [“Starting and stopping z/OS Connect EE” on page 471.](#)
 - a) Check the messages .log file for the following messages that confirm that the service is installed.
BAQR7043I: z/OS Connect EE service archive phonebook installed successfully.
 - b) Check the messages .log file for the following messages that confirm that the API is installed.
BAQR7000I: z/OS Connect API archive file contacts installed successfully.
3. In the **z/OS Connect EE Servers** view under the **APIs** folder, double-click on the contacts API.
The API is opened in the Swagger UI.
4. Click **List Operations** to see available operations in the API.

contacts

This is an API for managing contacts.

The screenshot shows the Swagger UI for the 'contacts' API. At the top, there are buttons for 'Show/Hide', 'List Operations' (which has a red arrow pointing to it), and 'Expand Operations'. Below this, there are four operation entries: a green 'POST' button for '/contacts', a red 'DELETE' button for '/contacts/{lastName}', a blue 'GET' button for '/contacts/{lastName}', and an orange 'PUT' button for '/contacts/{lastName}'. At the bottom left, there is a note: '[BASE URL: /phonebook , API VERSION: 1.0.0]'.

Figure 21. Operations in the API

5. Test adding a contact record by clicking **POST**.
 - a) Click the **Example Value** box to copy the request message format into your contacts POST request.
 - b) Specify the last name, first name, zip code, and extension.

Parameters

Parameter	Value	Description
postPhonebook_request	<input type="button" value="-"/> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>IVTNO_INPUT_MSG <input type="button" value="-"/></p> <p>IN_TRANCODE</p> <p>IVTNO <input type="text" value=""/></p> <p>IN_LAST_NAME <input type="text" value="Smith"/></p> <p>IN_FIRST_NAME <input type="text" value="John"/></p> <p>IN_EXTENSION <input type="text" value="000-0000"/></p> <p>IN_ZIP_CODE <input type="text" value="00000"/></p> </div>	request body

Parameter content type: application/json

Authorization

Figure 22. Testing the POST request

c) Click **Try it out!**.

Information about the request URL, request headers, response body, response code, and response headers are provided. The response body contains the output message. The OUT_MESSAGE would contain one of the following messages:

- ENTRY WAS ADDED
 - ADDITION OF ENTRY HAS FAILED (this message indicates that the IN_LAST_NAME value you specified already exists)

Results

You have verified your newly created phonebook service is working.

Create a two-way IBM MQ service

Create a two-way IBM MQ stock query service that drives the stock query application.

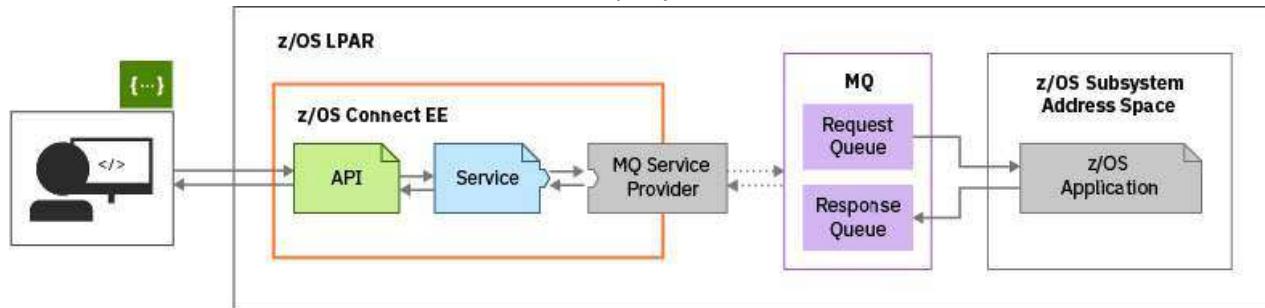
Before you begin

Ensure that the following tasks are complete:

- Compile and test the sample IBM MQ stock query application, and create the necessary IBM MQ queue definitions. For more information, see [“Prepare the sample IBM MQ stock query application” on page 63](#).
 - Install z/OS Connect EE v3.0 or later. For more information, see [“Installing z/OS Connect EE” on page 196](#).
 - Create a z/OS Connect EE server and configure it to connect to IBM MQ. For more information, see [“Create a server to connect to IBM MQ” on page 77](#).
 - Use the z/OS Connect EE API toolkit to Stop the stockQuery service, and then Remove the stockQuery service.

About this task

Use the API toolkit to re-create the IBM MQ stock query service.



Procedure

In this procedure, substitute the Windows example for the operating system of your choice.

1. Create a local directory on your workstation. For example, C:\copybooks.
2. Create a file called STOCKREQ.CPY and save it in C:\copybooks.

This file contains the COBOL copybook that is used for the request payload of the stock query service and contains the following lines:

```
01 SQREQ.  
      05 ITEM-ID  PIC 9(6).
```

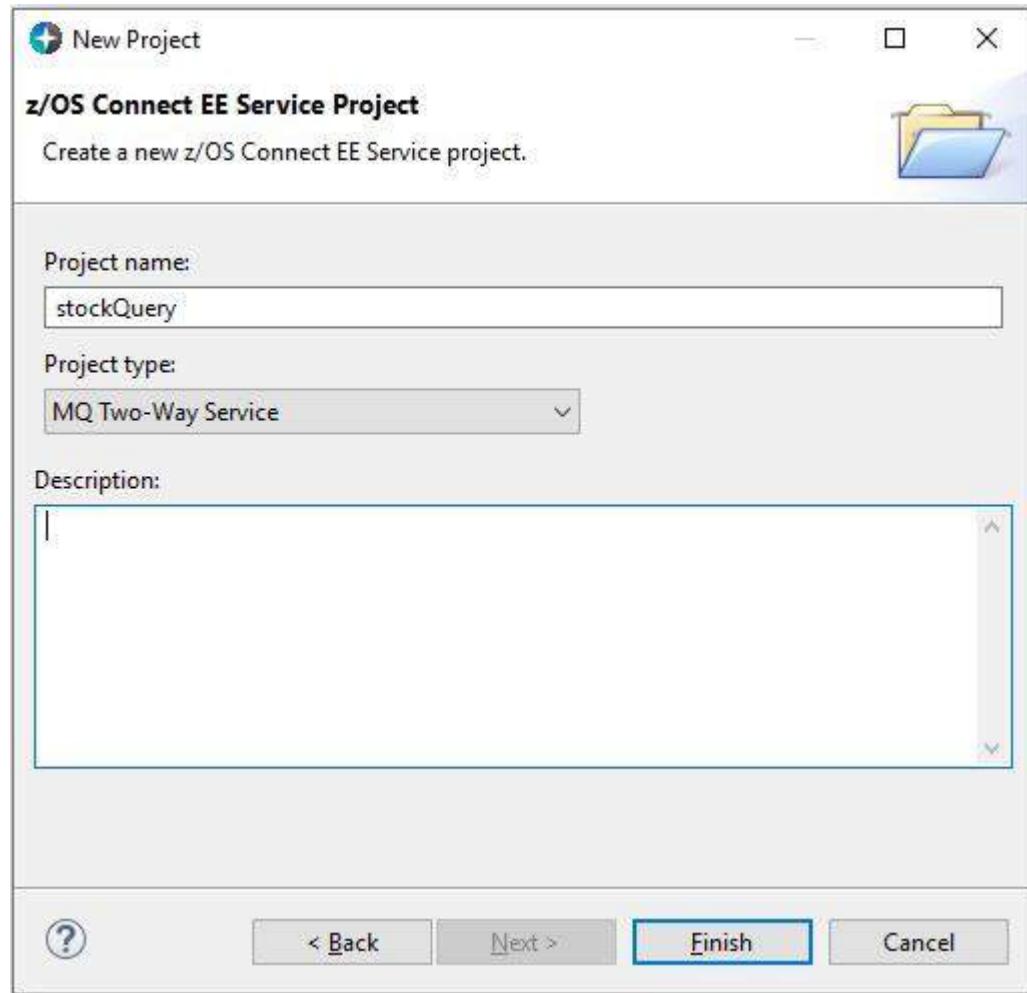
3. Create a file called STOCKRES.CPY and save it in C:\copybooks.

This file contains the COBOL copybook that is used for the response payload of the stock query service and contains the following lines:

```
01 SQRESP.  
      05 ITEM-ID    PIC 9(6).  
      05 ITEM-DESC   PIC X(20).  
      05 ITEM-COUNT  PIC 9(6).  
      05 ITEM-COST   PIC 9(6).
```

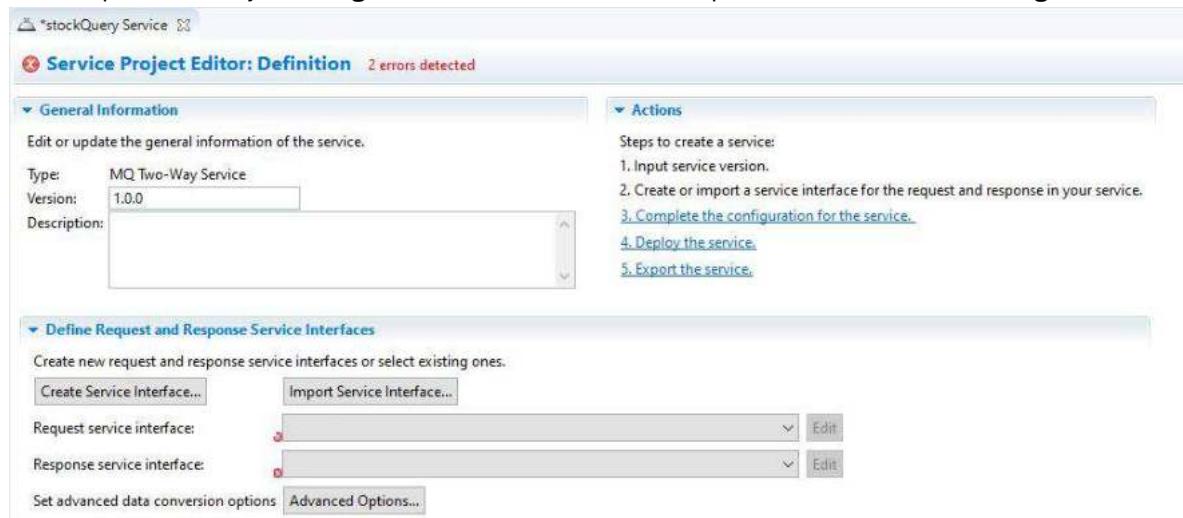
4. Create a new project.
 - a) Start IBM® Explorer for z/OS and open the **z/OS Connect Enterprise Edition** perspective.
 - b) Click **File > New > Project**.
The **New Project** wizard opens.
 - c) Click **z/OS Connect Enterprise Edition > z/OS Connect EE Service Project**, and click **Next**.
 - d) Enter the following information:

Field	Value
Project name	stockQuery
Project type	MQ Two-Way Service
Description	(Optional) Enter a description of your choice.



e) Click **Finish**.

The service project is created in the **Project Explorer** view. The z/OS Connect EE service project editor opens, initially showing some errors that indicate required information is missing.



The service definition opens in the service project editor, where you can configure the service and define the service interface.

In the following steps, you create the service interface for the request and response messages, by importing the COBOL data structure into this project and specifying which fields are to be used by input and output messages.

5. In the Define Request and Response Service Interfaces section, click **Create Service Interface**.

6. Create a service interface for the request messages.

a) Specify a name such as stockQueryRequest for the service interface.

This service interface can later be identified as the request service interface for this service project. It can also be reused by other service projects.

b) Click the import icon  to import the SQREQ copybook.

The Import wizard opens.

c) Enter the following information:

Field	Value
Import from:	Local file system
File type:	COBOL data structures only

d) On the **Data structure file** field, click **Browse**.

The **Select File** dialog opens.

e) Find the SQREQ copybook file STOCKREQ.CPY on your workstation, for example, C:\copybooks and click **Open**.

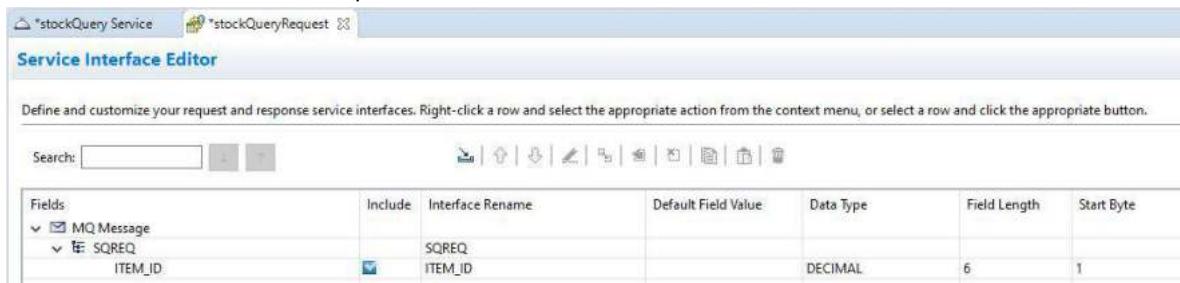
You can now specify a data structure for the input messages.

f) In the **Data structure name list**, select SQREQ.

g) Click **Add to Import List** to add this message data structure to the import list.

h) Click **OK**.

The service interface editor opens.



The screenshot shows the Service Interface Editor window. At the top, there are tabs for *stockQuery Service and *stockQueryRequest, with *stockQueryRequest selected. Below the tabs is a toolbar with various icons. The main area is titled "Service Interface Editor" and contains a table with columns: Fields, Include, Interface Rename, Default Field Value, Data Type, Field Length, and Start Byte. Under the "Fields" column, there is a tree view showing "MQ Message" expanded, with "SQREQ" under it, and "ITEM_ID" listed. The "Include" column has a checkmark next to "ITEM_ID". The "Interface Rename" column has "SQREQ" and "ITEM_ID" listed. The "Default Field Value" column is empty. The "Data Type" column has "DECIMAL" listed. The "Field Length" column has "6" listed. The "Start Byte" column has "1" listed. A search bar and filter icons are at the top left of the table area.

i) Expand **MQ Message** and **SQREQ** to see all fields in the data structure.

j) Click **File > Save** to save the request service interface.

7. Create a service interface for the response messages.

a) Click the **stockQuery service** tab.

The overview page of your service project opens.

b) In the **Define Request and Response Service Interfaces** section, click **Create Service Interface**.

c) Specify a name such as stockQueryResponse for the service interface.

d) Click the import button  to import the SQRESP copybook.

The Import wizard opens.

e) Enter the following information:

Field	Value
Import from:	Local file system
File type:	COBOL data structures only

f) On the **Data structure file** field, click **Browse**.

g) In the **Select File** window, find the SQRESP copybook file STOCKRES.CPY on your workstation.

For example, C:\copybooks and click **Open**.

h) Specify the data structure for the output messages:

In the **Data structure name** list, select SQRESP.

Click **Add to Import List** to add this message data structure to the Import list.

Click **OK**.

The service interface editor opens.

i) Expand **MQ Message** and **SQRESP** to see all fields in the data structure.

j) Click **File > Save** to save the response service interface.

You have created two service interface files, (.si files). In the Project Explorer view, these files, are stored in the **service-interfaces/** folder in your service project.

8. Assign these service interface files as the request and response service interfaces for this project.

a) In the service project editor, click the **Definition** tab of your service project.

b) Select the service interface files that you created as the interface to use for the request and the response.

i) For the **Request service interface** field, select the .si file that you just created for the request. For example, stockQueryRequest.si.

ii) For the **Response service interface** field, select the .si file that you just created for the response. For example, stockQueryResponse.si.

c) Save your changes.

9. Enter the details of the configuration for this service.

a) Select the **Configuration** tab of your service project.

The screenshot shows the 'Service Project Editor: Configuration' window. At the top, there are tabs for 'stockQuery Service', 'stockQueryRequest', and 'stockQueryResponse'. The 'stockQueryRequest' tab is selected. Below the tabs, the title bar says 'Service Project Editor: Configuration' with '4 errors detected'. A section titled 'Required Configuration' is expanded, containing the following fields:

Setting	Value
Connection factory JNDI name:	[Empty]
Request destination JNDI name:	[Empty]
Reply destination JNDI name:	[Empty]
Wait interval:	[Empty]
MQMD format:	MQSTR
Coded character set identifier (CCSID):	37
Is message persistent:	<input type="checkbox"/>
Reply selection:	msgIDToCorrelID
Expiry:	-1

b) Enter the following information:

Field	Value
Connection factory JNDI name	Enter the JNDI name of the jmsConnectionFactory element in your server.xml file. For example, jms/cf1.

Field	Value
Request destination JNDI name	Enter the JNDI name of the request jmsQueue element in your server.xml file. For example, jms/stockRequestQueue.
Reply destination JNDI name	Enter the JNDI name of the reply jmsQueue element in your server.xml file. For example, jms/stockResponseQueue.
Wait interval	10000. This value indicates that the service waits 10 seconds for a reply message.
Expiry	10000. This value indicates that the request message sent by the service will expire after 10 seconds.

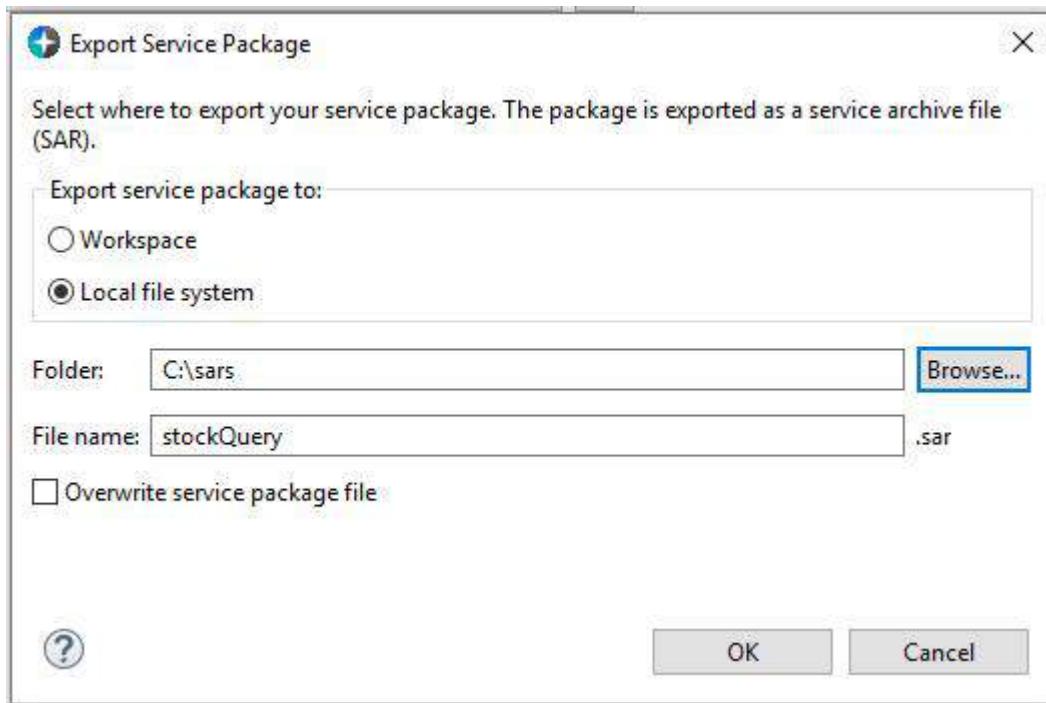
- c) Save your changes.

The screenshot shows the 'Service Project Editor: Configuration' window for the 'stockQuery Service'. The window has tabs at the top: 'stockQuery Service', 'stockQueryRequest', and 'stockQueryResponse'. The main area is titled 'Service Project Editor: Configuration' and contains a section titled 'Required Configuration'. The configuration fields and their values are:

- Connection factory JNDI name: jms/cf1
- Request destination JNDI name: jms/stockRequestQueue
- Reply destination JNDI name: jms/stockResponseQueue
- Wait interval: 10000
- MQMD format: MQSTR
- Coded character set identifier (CCSID): 37
- Is message persistent:
- Reply selection: msgIDToCorrelID
- Expiry: 10000

10. Export the stock query service to a service archive (.sar) file.

- Right-click the **stockQuery** service and select **z/OS Connect EE > Export z/OS Connect EE Service Archive**.
- In the **Export Service Package** window, select **Local file system** and choose an appropriate folder name. For example: C:\sars.
- Click **OK**.



Results

You created a service archive file that defines the stock query service, and exported it to the file system.

What to do next

Deploy the stock query service to the z/OS Connect EE server by using the z/OS Connect EE API toolkit.

Deploy the stock query service

Deploy the stock query service directly from z/OS Connect EE API Toolkit.

Before you begin

Create a connection to your z/OS Connect EE server from the z/OS Connect EE API toolkit. For more information, see [“Connecting to a z/OS Connect EE server” on page 603](#)

About this task

In the **Host Connections** view, you can add connections to z/OS Connect EE servers and credentials to store your user IDs and passwords.

Tip: If you don't see the **Host Connections** view, from the menu bar, click **Window > Manage Connections** to open the **Host Connections** view.

Procedure

In the Project Explorer view, right-click the service project and select **z/OS Connect EE > Deploy Service to z/OS Connect EE Server**.

If the service is already installed, you can select the Overwrite checkbox, or stop and remove the service before you deploy your new one.

Results

The stock query service is deployed to the z/OS Connect EE server.

What to do next

Create an API to invoke the stock query service. Follow the steps in “[Create an API to invoke the IBM MQ stock query service](#)” on page 149.

Create an IMS database service

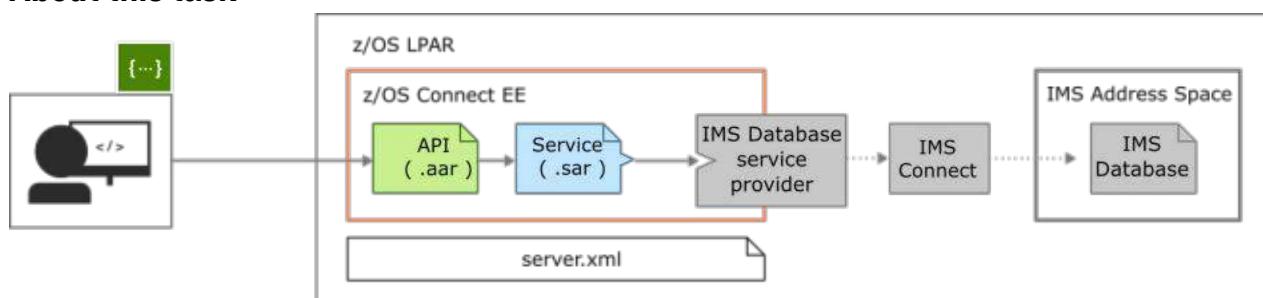
This scenario shows how you can create an IMS database service for the IMS sample application database by using the API toolkit.

Before you begin

To create and test the IVP database service, ensure that the following tasks are complete:

1. Install and configure the IMS IVP sample database. The procedure is described in the scenario “[Prepare the sample IMS database](#)” on page 62.
2. Install z/OS Connect EE v3.0.30 or later. For more information, see “[Installing z/OS Connect EE](#)” on page 196.
3. Configure and test your z/OS Connect EE server connection to an IMS database. The procedure is described in the scenario [Create a server and connect to an IMS database](#), and installs relevant artifacts by using the supplied configuration template sampleImsDatabase, including the dbcontacts service.
4. Configure and test your host connections for an IMS database. This procedure is described in “[Creating an IMS database host connection](#)” on page 265. Host connection are used to create service interfaces from SQL queries. For this example, the IMS database host connection is named IMSIVPDBConnection.

About this task



In this scenario, the API toolkit is running on a Windows workstation to recreate the IMS IVP database service. The service archive file uses an existing IMS IVP database that is configured on IMS in previous steps.

This scenario uses the following values and names:

- IMSIVPCF = name of the IMS database connection profile created in previous steps.
- IMSIVPDBConnection = the name of the IMS database host connection specified in pre-requisite step 4.

Procedure

1. Switch to the **z/OS Connect EE** perspective.
 - a) From the main menu, select **Window > Open Perspective > Other**
The **Select Perspective** wizard opens.
 - b) Click **z/OS Connect Enterprise Edition**.
 - c) Click **OK**.
The **z/OS Connect EE** perspective opens.
2. Select **File > New > Project**.

The New Project wizard opens.

3. Select **z/OS Connect Enterprise Edition** > **z/OS Connect EE Service Project**, and click **Next**.
4. Specify a project name, select the project type, and optionally provide a description.
 - a) Specify the project name, dbPhonebook.
 - b) For project type, select **IMS Database Service**.
 - c) Click **Finish** to create the project.

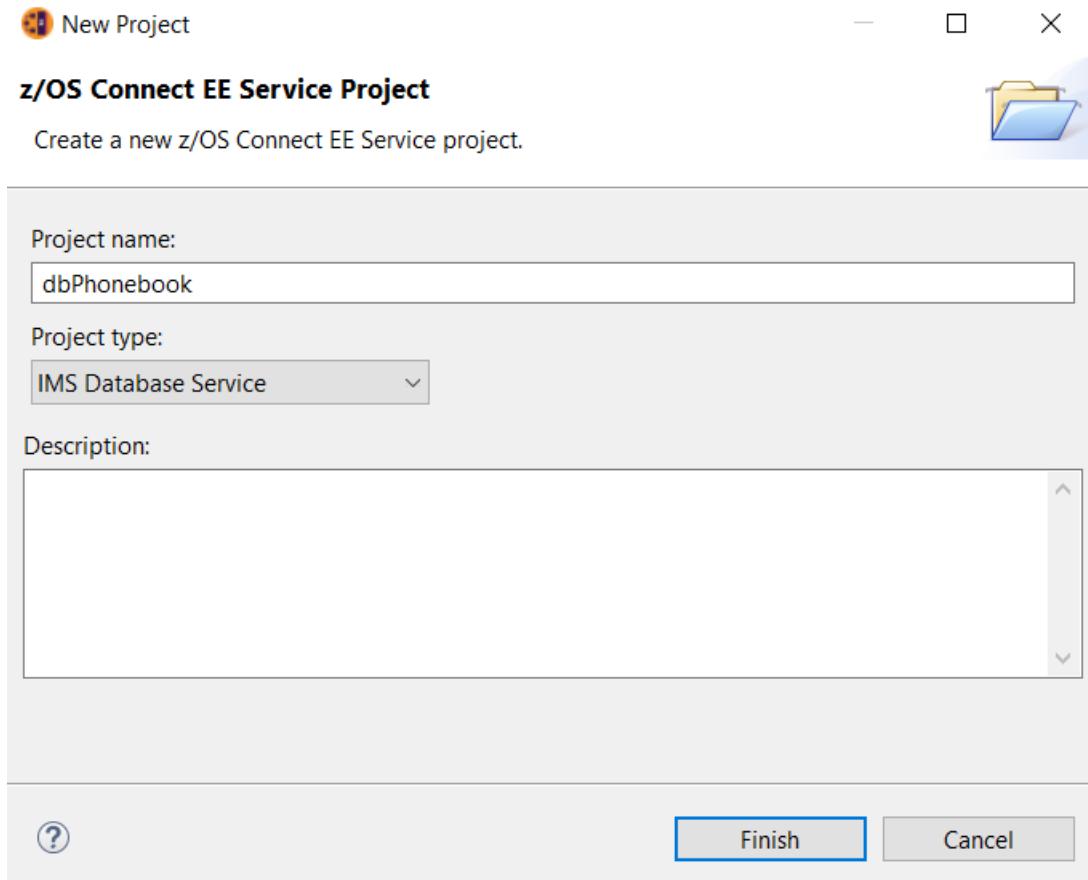


Figure 23. The New project wizard

The service project is created in the **Project Explorer** view. The service definition opens in the service project editor, where you can configure the service and define the service interface. The **z/OS Connect EE service project editor** opens, with some errors initially that indicate some required information is still missing.

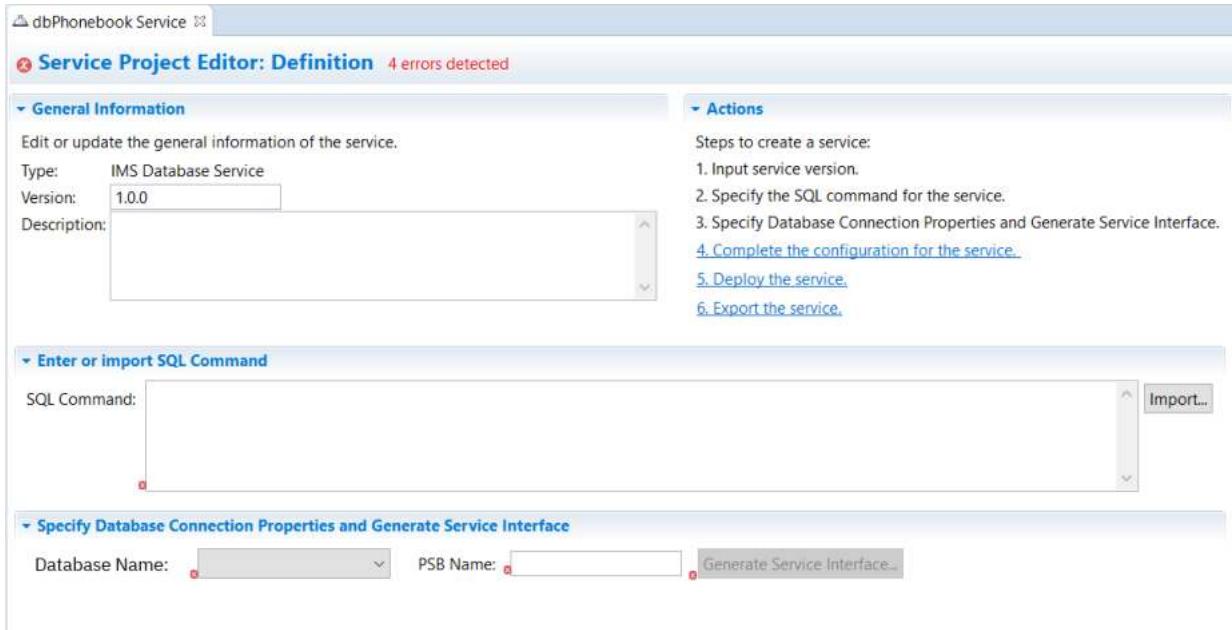


Figure 24. The definition page in the service project editor

5. In the **Service Project Editor**, specify the following information.
 - a) For the **Database Connection** field, specify the database connection you configured pre-requisite step 3. In this example, the connection name is IMSIVPDBConnection.
 - b) For the **Database Name** field, specify DFSIVP37.
 - c) For the SQL command field, specify the following SQL statement: `SELECT LASTNAME, FIRSTNAME, EXTENSION, ZIPCODE FROM PHONEAP.PHONEBOOK WHERE LASTNAME=?`
6. Click **Generate Service Interface**.

A dialog box will appear and inform you that your service interface has successfully been created.

Important: You must click the **Generate Service Interface** button any time changes are made to your SQL command.
7. In the lower left area of the **Service Project Editor**, click the **Configuration** tab.
8. In the **Connection profile** field, specify IMSIVPCF.
9. Save your changes by clicking **File > Save**, or the windows shortcut **Ctrl+S**.

Results

You have created an IMS IVP database service and it is now ready to be deployed.

What to do next

[“Deploy the IMS database service using the API toolkit” on page 106.](#)

Related tasks

[“Create a server and connect to an IMS database” on page 74](#)

This task shows you how to create a z/OS Connect EE server and configure a connection to an IMS database.

[“Deploy the IMS database service using the API toolkit” on page 106](#)

Deploy the IVP database service directly from z/OS Connect EE API toolkit by first creating a connection to the server.

Deploy the IMS database service using the API toolkit

Deploy the IVP database service directly from z/OS Connect EE API toolkit by first creating a connection to the server.

Before you begin

Create a connection to your z/OS Connect EE server from the z/OS Connect EE API toolkit. See [“Connecting to a z/OS Connect EE server” on page 603](#).

About this task

In the **Host Connections** view, you can add connections to z/OS Connect EE servers and credentials to store your user IDs and passwords.

Tip: If you don't see the **Host Connections** view, from the menu bar, click **Window > Manage Connections** to open the **Host Connections** view.

Procedure

In the **Project Explorer** view, right-click the service project and select **z/OS Connect EE > Deploy Service to z/OS Connect EE Server**. If you already have the service installed, you can select the **Overwrite** check box, or Stop and Remove the service before deploying your new one.

What to do next

[“Test the IMS database service” on page 106](#)

Related tasks

[“Create an IMS database service” on page 103](#)

This scenario shows how you can create an IMS database service for the IMS sample application database by using the API toolkit.

[“Test the IMS database service” on page 106](#)

You can test your newly created IVP database service is working in z/OS Connect EE API toolkit by using the “Try it out!” function in the Swagger UI, or alternatively with a REST client.

Test the IMS database service

You can test your newly created IVP database service is working in z/OS Connect EE API toolkit by using the “Try it out!” function in the Swagger UI, or alternatively with a REST client.

Procedure

To test your newly created and deployed IVP database API from within the API Toolkit, take the following steps:

1. Start IMS and ensure that IMS Connect is listening on the expected port.
2. Start your z/OS Connect EE server. For more information, see [“Starting and stopping z/OS Connect EE” on page 471](#).
 - a) Check the messages.log file for the following messages that confirm that the service is installed.
BAQR7043I: z/OS Connect EE service archive dbPhonebook installed successfully.
 - b) Check the messages.log file for the following messages that confirm that the API is installed.
BAQR7000I: z/OS Connect API archive file dbcontacts installed successfully.
3. In the **z/OS Connect EE Servers** view under the **APIs** folder, double-click on the IMS database API. The API is opened in the Swagger UI.

4. Click **List Operations** to see available operations in the API.

dbcontacts

The screenshot shows the dbcontacts API operations page. At the top right are links for 'Show/Hide', 'List Operations', and 'Expand Operations'. Below this, a blue button labeled 'GET' is followed by a '/' separator. A note at the bottom left says '[BASE URL: /dbcontacts , API VERSION: 1.0.0]'.

Figure 25. Operations in the API

5. Test querying a contact record by clicking **GET**.

- a) The **Example Value** box opens, demonstrating the response message payload format from your database GET request.

This screenshot shows the details for the GET / operation. At the top right are links for 'Show/Hide', 'List Operations', and 'Expand Operations'. Below this, a blue button labeled 'GET' is followed by a '/' separator. The main area displays the 'Response Class (Status 200)' as 'OK'. Under the 'Model' tab, the 'Example Value' shows a JSON response structure:

```

{
  "result": [
    {
      "LASTNAME": "string",
      "FIRSTNAME": "string",
      "EXTENSION": "string",
      "ZIPCODE": "string"
    }
  ]
}

```

Below the example value, the 'Response Content Type' is set to 'application/json'. The 'Parameters' section includes two entries: 'Authorization' (header, string) and 'lastName' (query, string). At the bottom are 'Try it out!' and 'Hide Response' buttons.

- b) Under the **Parameters** list, specify a **lastName** value of LAST1

- c) Click **Try it out!**.

Information about the request URL, request headers, response body, response code, and response headers are provided.

You will receive the following response body.

```
{
  "response": {
    "result": [
      {
        "LASTNAME": "LAST1",
        "ZIPCODE": "D01/r01",
        "FIRSTNAME": "FIRST1",
        "EXTENSION": "8-111-1111"
      }
    ]
  }
}
```

The following steps describe how you can test your database service using a REST client:

6. Start IMS and ensure that IMS Connect is listening on the expected port.
7. Start your z/OS Connect EE server if it is not already running. For more information, see [“Starting and stopping z/OS Connect EE” on page 471](#).

a) Check the messages.log file for the following messages that confirm that the API is installed.

BAQR7000I: z/OS Connect API archive file dbcontacts installed successfully.

b) Check the messages.log file for the following messages that confirm that the service is installed.

BAQR7043I: z/OS Connect EE service archive dbPhonebook installed successfully.

8. In a REST client, issue the following HTTP POST, or PUT, command:

`https://<yourServerName>:<port>/zosConnect/services/dbPhonebook?action=invoke`

Where <yourServerName> is the name of your server, and <port> is the port.

Submit the following payload with your POST or PUT command:

```
{  
  "request": {  
    "LASTNAME": "LAST1"  
  }  
}
```

If your service is working, you will get the following response body in your REST client.

```
{  
  "response": {  
    "result": [  
      {  
        "LASTNAME": "LAST1",  
        "ZIPCODE": "D01/r01",  
        "FIRSTNAME": "FIRST1",  
        "EXTENSION": "8-111-1111"  
      }  
    ]  
  }  
}
```

Results

You have verified your newly created database service is working.

Create a Db2 service

This scenario shows how you can recreate the Db2 employeeList service using the Db2 service manager in the API toolkit.

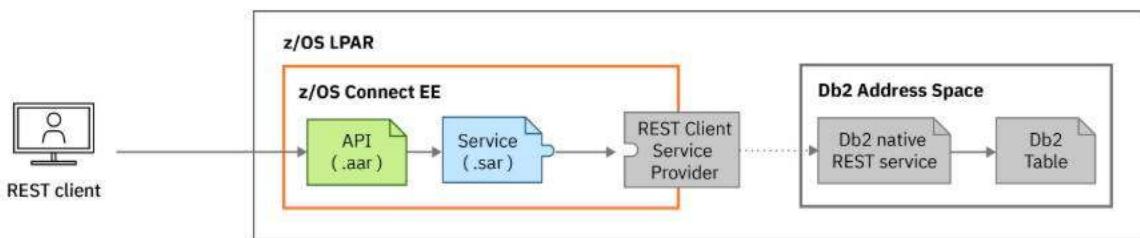
Before you begin

To create and test the employeeList service ensure that the following tasks are complete:

1. Create and enable Db2 Native REST services. The procedure is described in the scenario “[Prepare the sample Db2 native REST services](#)” on page 66
2. Install z/OS Connect EE v3.0.29 or later. For more information, see “[Installing z/OS Connect EE](#)” on page 196.
3. Install the API Toolkit, see “[Installing z/OS Explorer and the z/OS Connect EE API toolkit](#)” on page 201.
4. Configure and test your connection to Db2. The procedure is described in the scenario “[Create a server to connect to Db2](#)” on page 80 and installs relevant artifacts using the supplied configuration template `sampleDb2ProjectManager`. This includes the employeeList service.
5. Use the z/OS Connect EE API toolkit to **Stop** and **Remove** the employeeList service.

About this task

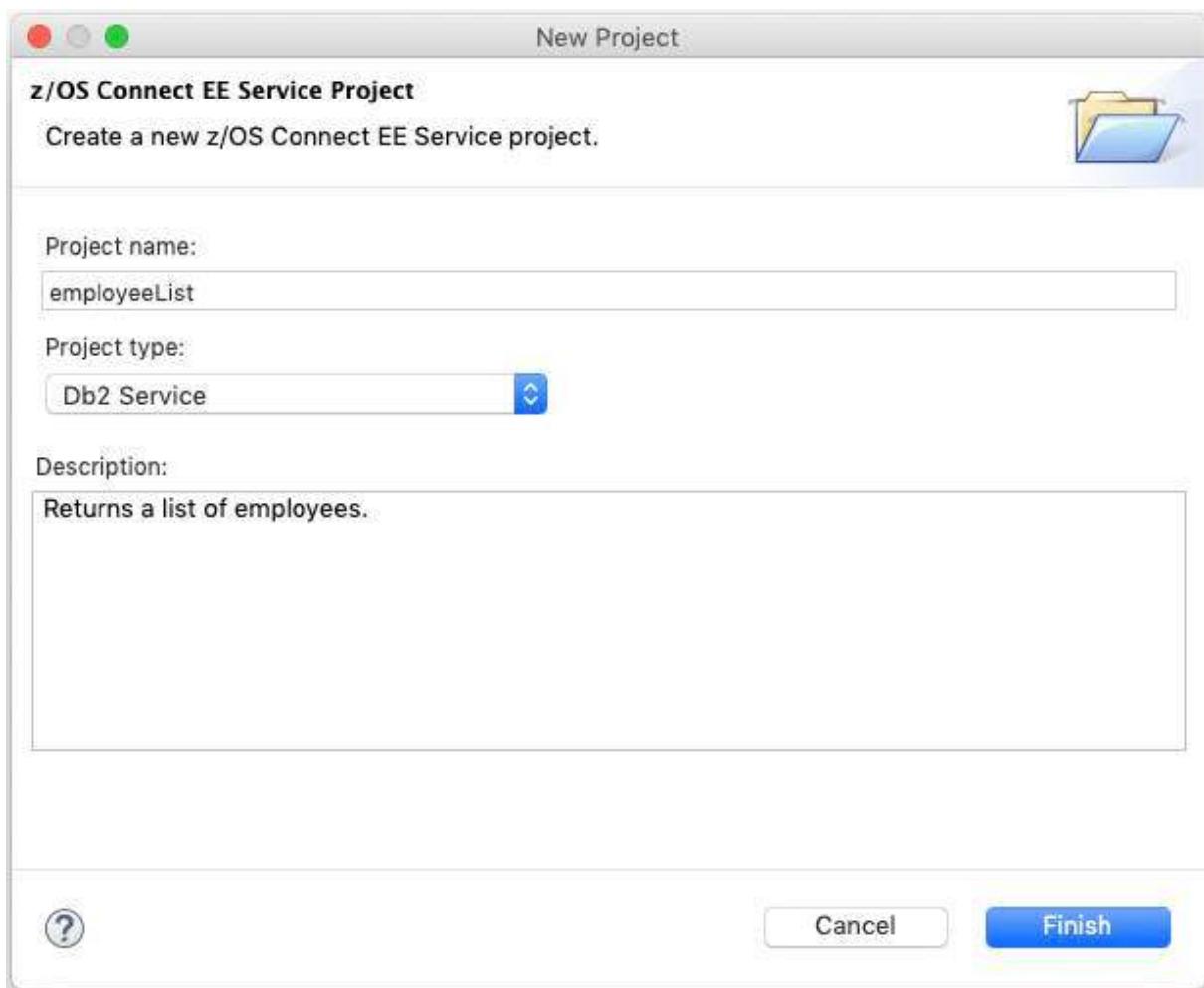
In this case we use screenshots of the API toolkit running on a MacOS workstation to recreate the Db2 employeeList service. This scenario also supports Windows and Linux workstations.



Procedure

1. Start IBM Explorer for z/OS and open the z/OS Connect Enterprise Edition perspective.
2. Right click in the project explorer window and select **New > Project....**. Select the **z/OS Connect EE Service Project** wizard.
3. Enter employeeList for the project name, select Db2 Service for the project type and enter Returns a list of employees. for the description of the new service project.

Click **Finish**.



The **Service Project Editor** opens.

*employeeList Service

Service Project Editor: Definition 4 errors detected

General Information

Edit or update the general information of the service.

Type: Db2 Service
Version: 1.0.0
Description: Returns a list of employees.

Actions

Steps to create a service:

1. Input service version.
2. Import JSON schemas from a Db2 service manager or your local machine.
- [3. Complete the configuration for the service.](#)
- [4. Deploy the service.](#)
- [5. Export the service.](#)

Define Db2 service

Import a Db2 native REST service from a Db2 service manager. Alternatively, enter your Db2 service details and import the JSON schemas from your local machine.

Import from Db2 service manager...

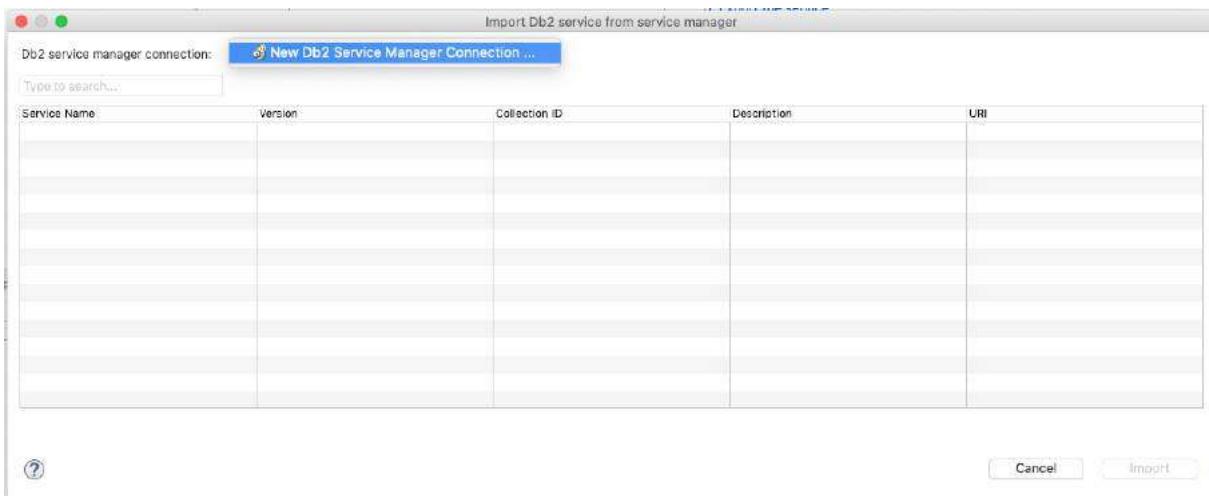
Collection ID: SYSIBMSERVICE
Db2 native REST service name: myService
Db2 native REST service version: V1
Request JSON schema:
Response JSON schema:

Definition Configuration

4. Click **Import from Db2 service manager...**

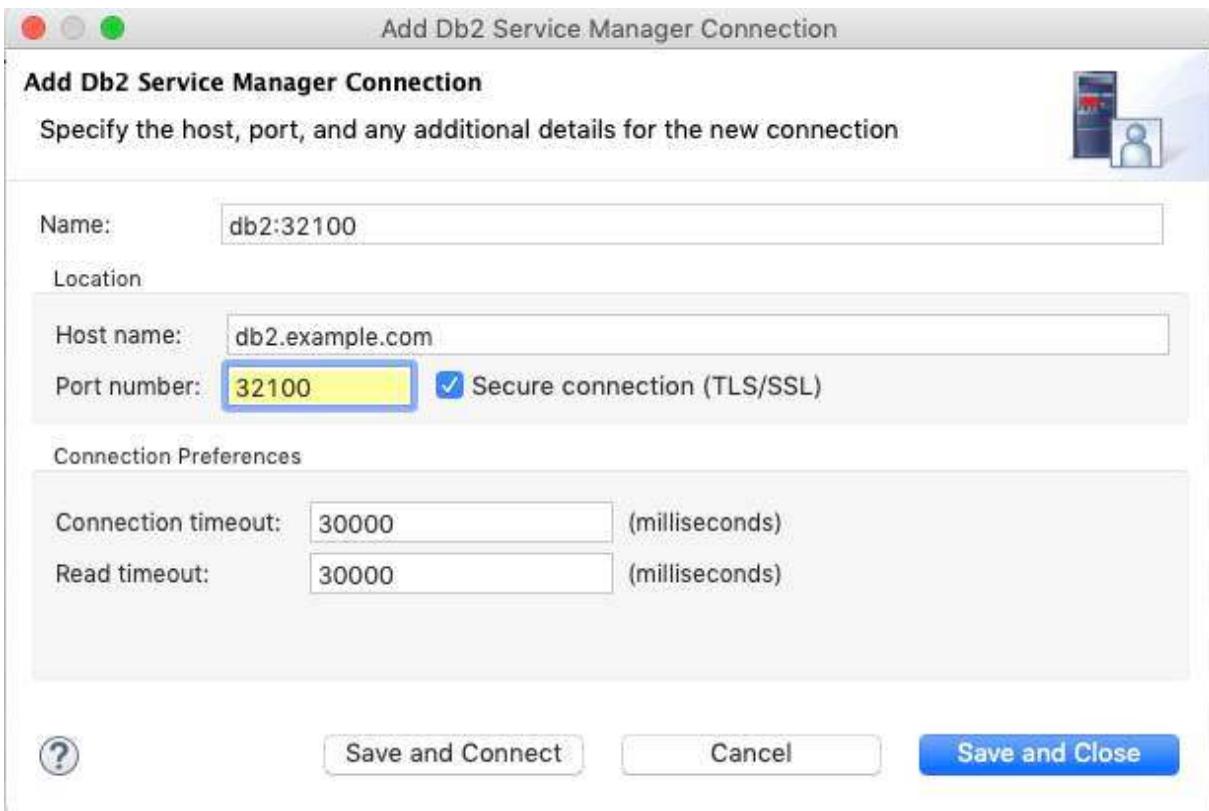
The Import Db2 service from service manager dialog opens.

5. Select **New Db2 service manager connection...**



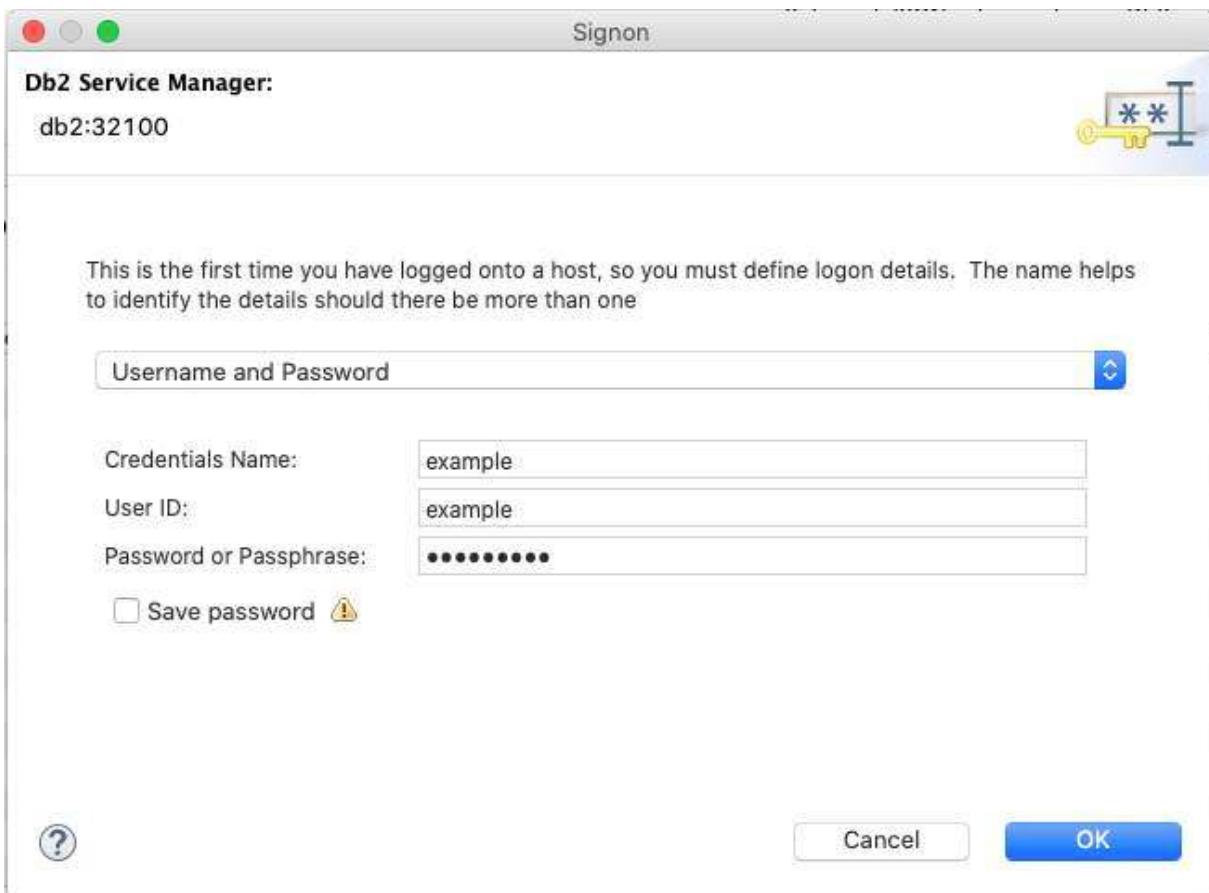
6. Enter the host name and port number for Db2. If Db2 uses a secure connection, select **secure connection (TLS/SSL)**.

Click **Save and Connect**.



7. Enter your credentials to connect to Db2.

Click **OK**.



A green circle indicates that the connection was successful.

Import Db2 service from service manager					
Db2 service manager connection:	db2	Type to search...	3 services total		
Service Name	Version	Collection ID	Description	URI	
employeeDetails	V1	SYSIBMSERVICE	Returns the details of an individual e...	/services/SYSIBMSERVICE/employe...	
employeeList	V1	SYSIBMSERVICE	Returns a list of employees.	/services/SYSIBMSERVICE/employe...	
employeeUpdate	V1	SYSIBMSERVICE	Updates the details of an individual...	/services/SYSIBMSERVICE/employe...	

The Db2 native REST services for the connection are shown in the table.

8. Use the search tool in the top left corner to find the Db2 native REST service **employeeList**.
Select the **employeeList** service in the table and click **Import**.

Import Db2 service from service manager					
Db2 service manager connection:	db2	Type to search...	3 services total		
Service Name	Version	Collection ID	Description	URI	
employeeList	V1	SYSIBMSERVICE	Returns a list of employees.	/services/SYSIBMSERVICE/employe...	

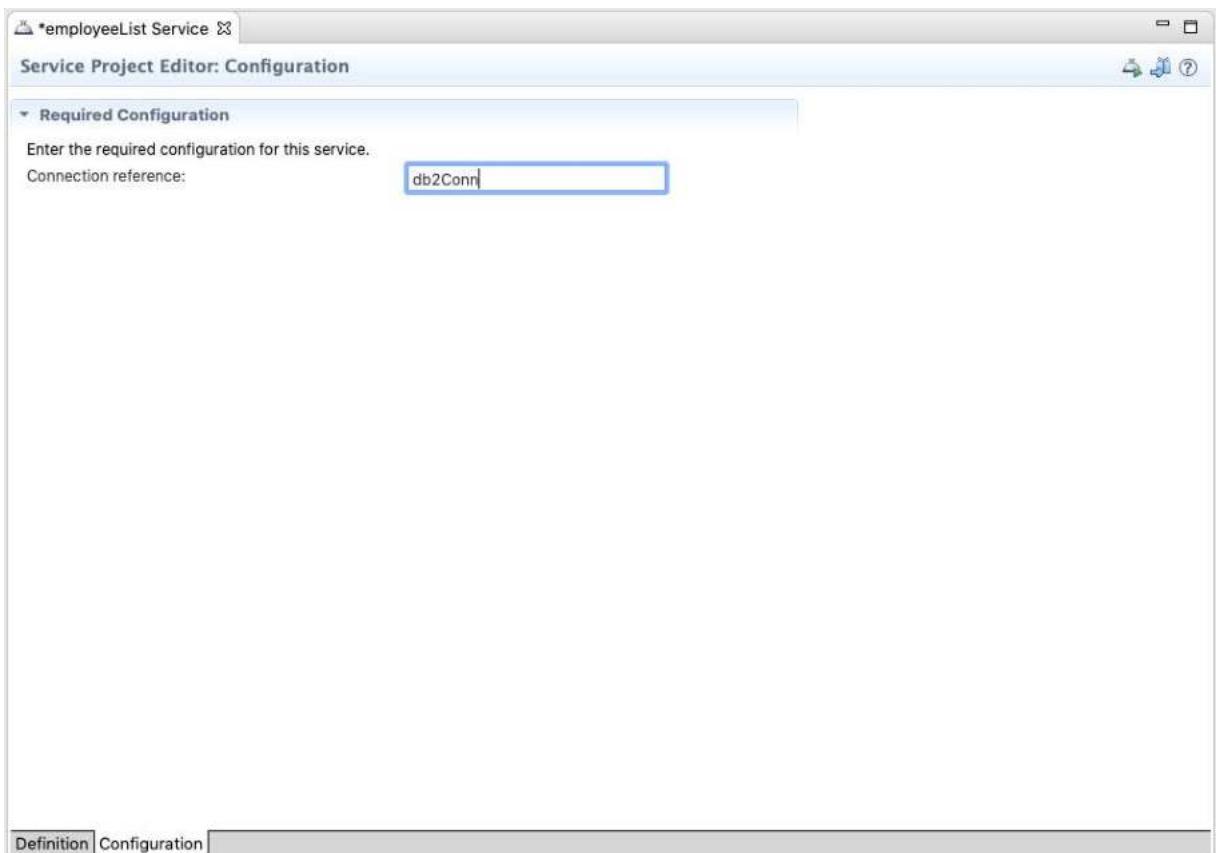
9. The fields Collection ID, Db2 native REST service name, Db2 native REST service version, Request JSON schema and Response JSON schema have been filled.

The screenshot shows the Service Project Editor interface. On the left, the Project Explorer displays a project named 'employeeList' containing 'request-schema.json' and 'response-schema.json'. The main area is titled 'employeeList Service' and contains the 'Service Project Editor: Definition' tab. Under 'General Information', the Type is set to 'Db2 Service', Version to '1.0.0', and Description to 'Returns a list of employees.'. The 'Actions' section lists steps: 1. Input service version, 2. Import JSON schemas from a Db2 service manager or your local machine, 3. Complete the configuration for the service, 4. Deploy the service, and 5. Export the service. The 'Define Db2 service' section allows importing from a Db2 service manager, setting the Collection ID to 'SYSIBMSERVICE', and specifying the Db2 native REST service name as 'employeeList' with version 'V1'. It also provides options to import Request and Response JSON schemas from a local machine.

You can view the contents of the selected `request-schema.json` and `response-schema.json` schemas in the **Project Explorer view** as shown on the left hand side.

10. Click the **Configuration** tab at the bottom of the screen.

Add the connection reference db2Conn.



11. Save the employeeList service.

Results

The employeeList service is now ready to be deployed.

What to do next

Deploy and test your newly created employeeList service.

Deploy the list employees service

Deploy the employeeList service directly from z/OS® Connect EE API toolkit by first creating a connection to the server.

Before you begin

Create a connection to your z/OS Connect EE server from z/OS Connect EE API toolkit. See [“Connecting to a z/OS Connect EE server” on page 603](#).

About this task

In the **Host Connections** view, you can add connections to z/OS Connect EE servers and credentials to store your user IDs and passwords.

Tip: If you don't see the **Host Connections** view, from the menu bar, click **Window > Manage Connections** to open the **Host Connections** view.

Procedure

In the **Project Explorer** view, right-click the service project and select **z/OS Connect EE > Deploy Service to z/OS Connect EE Server**.

Test the list employees service

Test the created Db2 native REST service `employeeList` to ensure it has installed correctly.

About this task

Use a REST client to test the Db2 native REST service. One API request is made for the created Db2 native REST service.

For the request, replace `<zos_connect_server_hostname>` with the hostname of your z/OS Connect EE server and `<zos_connect_server_port>` with the port of your z/OS Connect EE server. You can also include authentication credentials such as the Authorization HTTP header.

Procedure

1. Use a REST client to test the `employeeList` service.

Send an HTTP POST request to `https://<zos_connect_server_hostname>:<zos_connect_server_port>/zosConnect/services/employeeList?action=invoke` with header `Content-Type: application/json` and the following body:

```
{}
```

A response code HTTP OK (200) indicates success. No JSON properties are required for the `employeeList` Db2 native REST service, so an empty JSON object payload is used.

2. Information about the request URL, request headers, response body, response code, and response headers are provided. The response body starts with:

```
{
    "StatusDescription": "Execution Successful",
    "ResultSet Output": [
        {
            "firstName": "BRUCE",
            "lastName": "ADAMSON",
            "department": "D11",
            "employeeNumber": "000150"
        },
        ...
    ],
    "StatusCode": 200
}
```

Results

The Db2 native REST service `employeeList` is working correctly.

What to do next

Create an API for the `employeeList` service, see [“Create a Db2 API to invoke the Db2 employee services” on page 154](#).

Create APIs

These tasks show you how to create APIs.

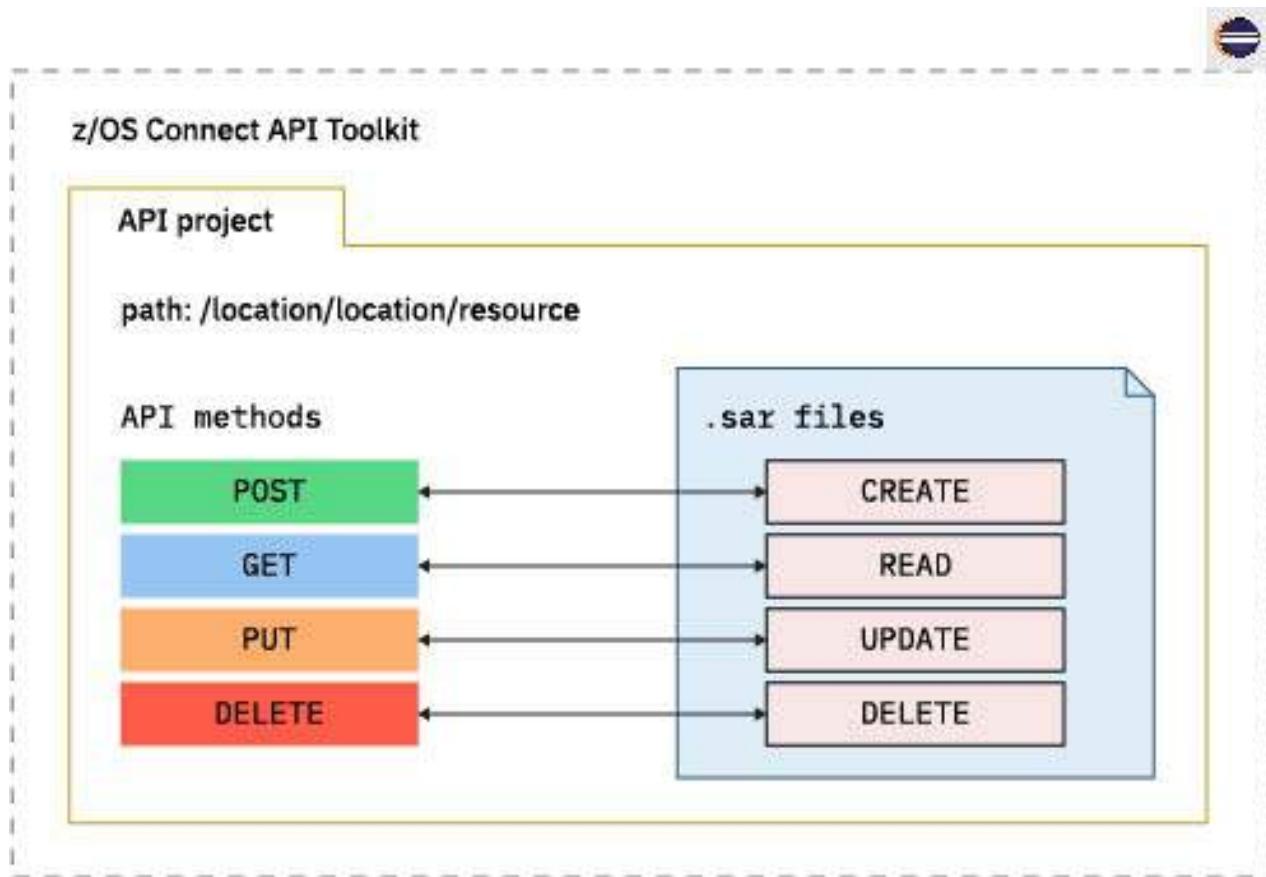
Create an API to invoke the CICS catalog manager services

This scenario shows you how to recreate the CICS catalog API for the three services used to call the catalog manager example application.

Before you begin

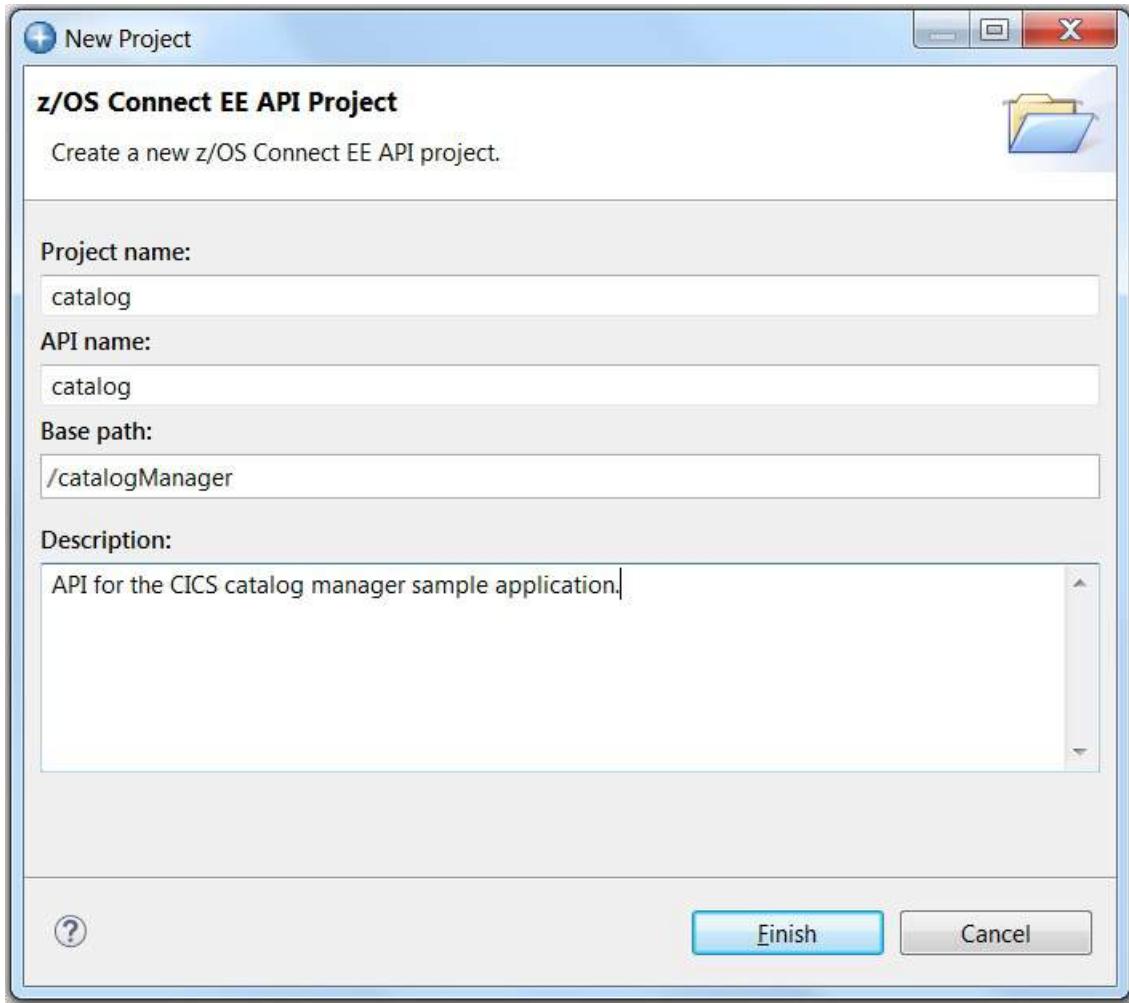
To create and test the catalog API ensure that the following tasks are complete:

1. Install and configure the CICS catalog manager example application. The procedure is described in the scenario [“Prepare the sample CICS application” on page 59](#)
2. Configure and test your IPIC connection to CICS. The procedure is described in the scenario [“Create a server and connect to CICS” on page 69](#) and installs relevant artifacts using the supplied configuration template `sampleCicsIpicCatalogManager`. This includes the catalog API.
3. Use the z/OS Connect EE API toolkit to Stop the catalog API, and then Remove the catalog API.



Procedure

1. Start IBM Explorer for z/OS and open the **z/OS Connect Enterprise Edition** perspective.
2. From the menu bar, select **File > New > z/OS Connect EE API Project**.
3. Complete the sections for the new API project and click **Finish**.

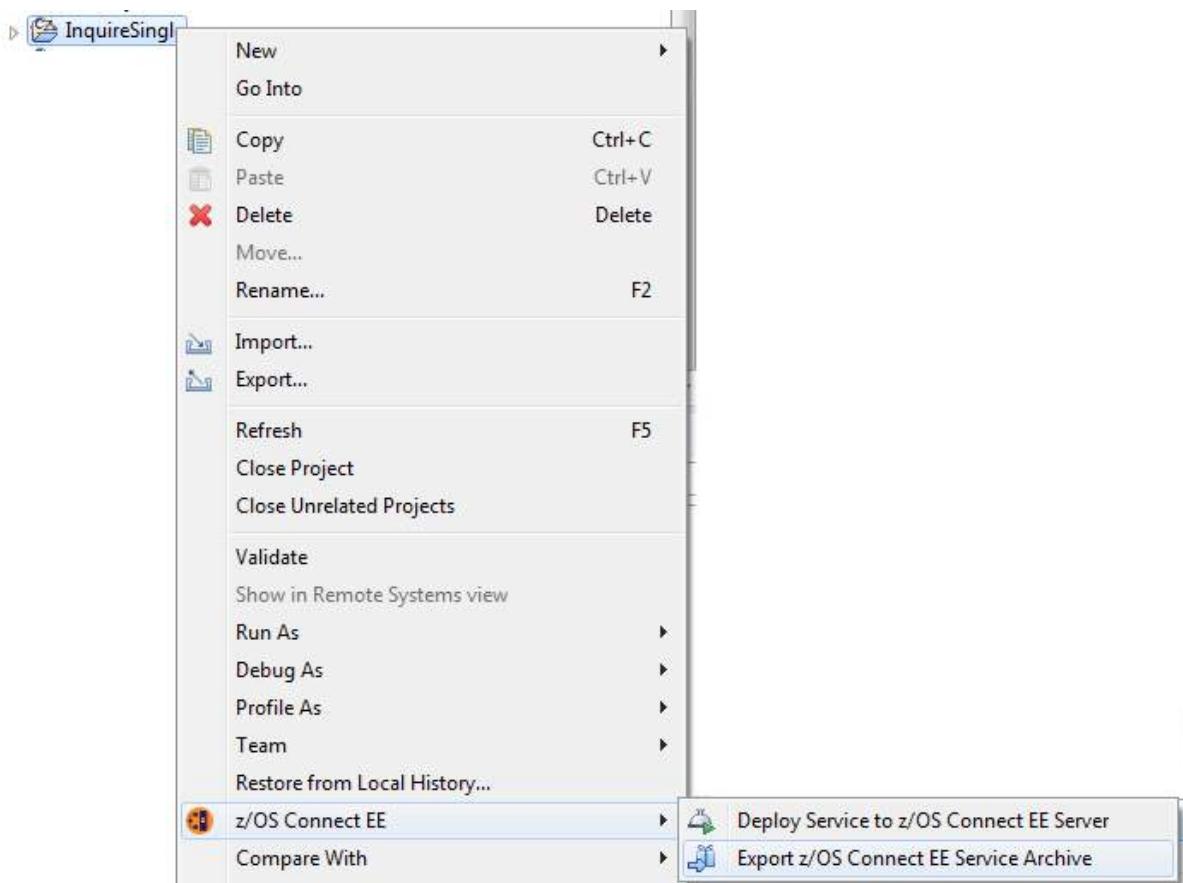


The z/OS Connect EE API Editor opens.

The API Editor window shows a new API named 'catalog' with a base path of '/catalogManager' and a version of '1.0.0'. The description is 'API for the CICS catalog manager sample application.' A 'Path' section contains a single entry '/newPath1'. The 'Methods' section lists four methods: POST, GET, PUT, and DELETE, each associated with a service and mapping configuration area.

The service archive files must now be exported into your catalog API project. There are two methods to achieve this.

4. If you created the `inquireSingle.sar` file in the “[Create a CICS service](#)” on page 83 scenario, use the z/OS Connect EE API toolkit to export the service archive to your catalog API project.
 - a) From the z/OS Connect EE perspective, right click your InquireSingle service project
 - b) Select **z/OS Connect EE > Export z/OS Connect EE Service Archive**.



- c) In the Export Service Package dialog, select **Workspace**.
 - d) Click **Browse...** to locate your catalog API.
 - e) Select your catalog API and click **OK**.
5. The service archive files provided with the `sampleCicsIpicCatalogManager` template need to be transferred in binary mode to the machine where IBM Explorer for z/OS is installed.
 - a) Transfer the `inquireCatalog.sar` and `placeOrder.sar` in binary to a directory on your workstation, for example, `C:/SARFILES`. You need to also export the `inquireSingle.sar` file if you did not export it with the z/OS Connect EE API toolkit as described in step “[4](#)” on page 117.
 - b) To import the service archive files into your catalog API project, right click on your catalog API project
 - c) Select **Import...** and locate the files on your file system, for example, `C:/SARFILES`.
 - d) Select the service archive files to import and click **Finish**.

What to do next

In the following tasks, you create operations for a catalog manager API.

Create an operation to inquire about an item

The following steps create an operation to inquire about an item in the catalog.

Before you begin

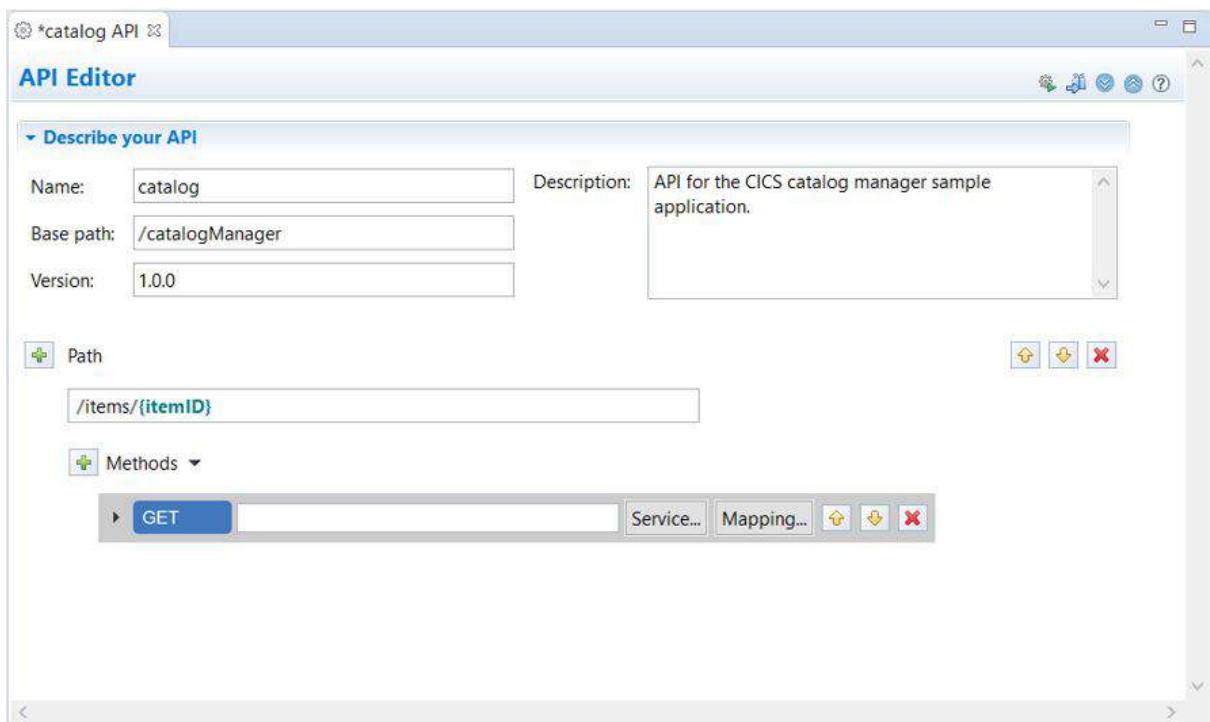
If the `inquireSingle.sar` file was generated on a different machine, transfer the file, in binary mode, to the workstation where IBM Explorer for z/OS is installed.

About this task

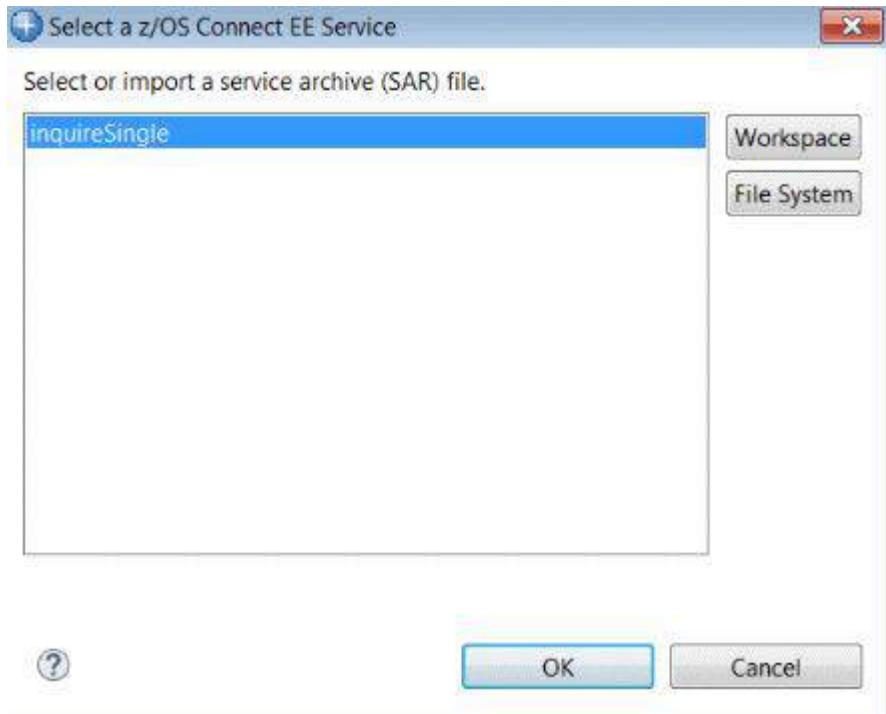
The operation is implemented as an HTTP GET request and demonstrates exposing a resource using a path parameter, with assign, move and remove transforms to map to the `inquireSingle` service.

Procedure

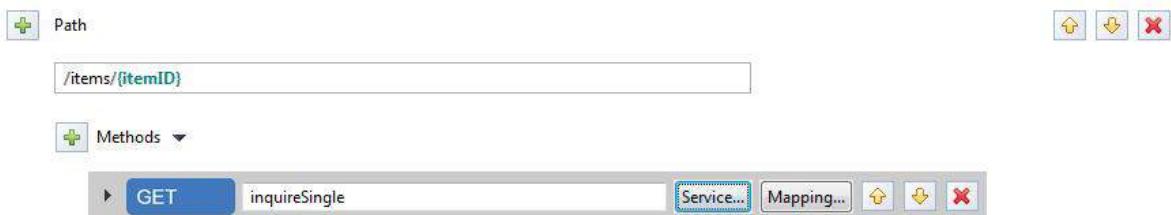
1. Rename the default **Path** value to be `/items/{itemID}` by typing over the existing value, where `{itemID}` indicates a path parameter whose value will be substituted at runtime.
2. Remove the POST, PUT and DELETE methods by clicking on the  at the end of the line. This should leave only the GET method allowed for the path `/items/{itemID}`.



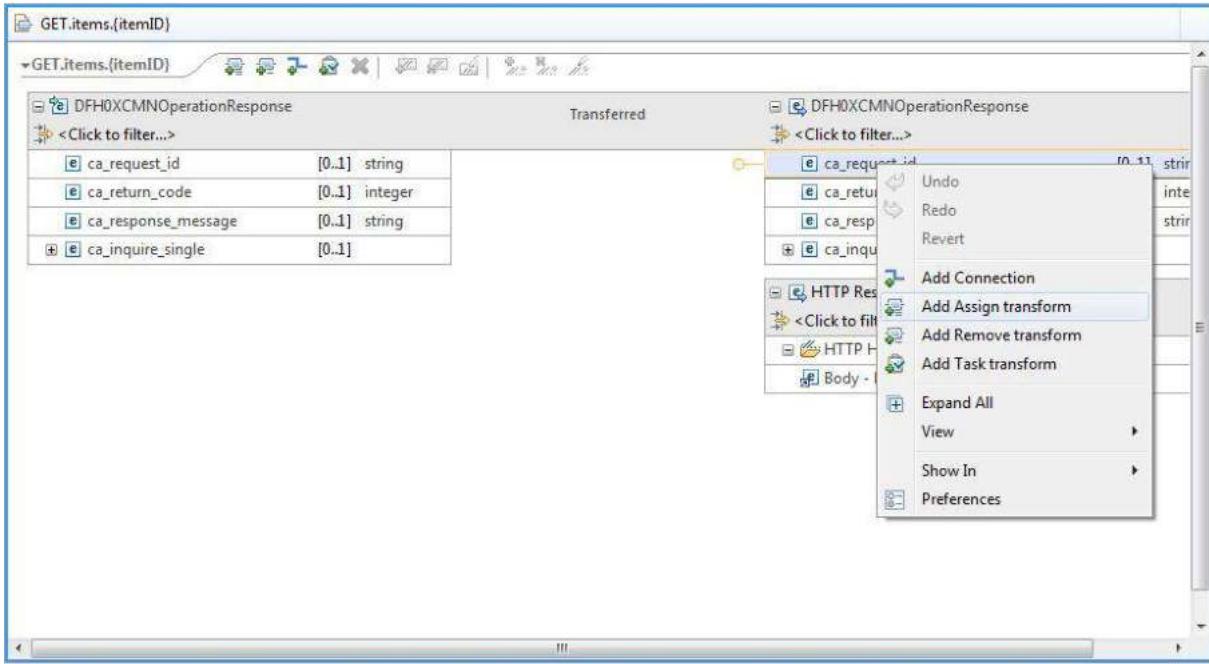
3. Click **Service...** for the GET method to select the service archive file that defines the service on this API path that will be called by an HTTP GET request.
The **Select a z/OS Connect EE Service** dialog opens.
4. Click **File System** and navigate to the location of the `inquireSingle.sar` file. Select the file and click **Open**.
The **Import Services** dialog opens. Click **OK**. The `inquireSingle` service now appears in the Service dialog.



Click **OK**. This operation now maps to the `inquireSingle` service.



5. From the menu bar, select **File > Save**.
6. Click **Mapping...** for the GET method, then click **Open Both Mappings**.
The request and response mapping editors open so that you can define the mapping between the content of the API's HTTP request and response, and the JSON content passed to and from the z/OS Connect Enterprise Edition service.
7. In the request mapping, right click on `ca_request_id` and click **Add Assign transform**.
This assigns a static value to the property in the request JSON payload passed to the service.



8. Ensure **Assign** has focus, so that the **Transform - Assign** dialog is displayed in the **Properties** view. On the **General** tab, set **Value** to **01INQS**.

This is the value required by the CICS Catalog Manager application to perform an inquiry on a single catalog item.

9. Ensure **Omit from interface** is checked.

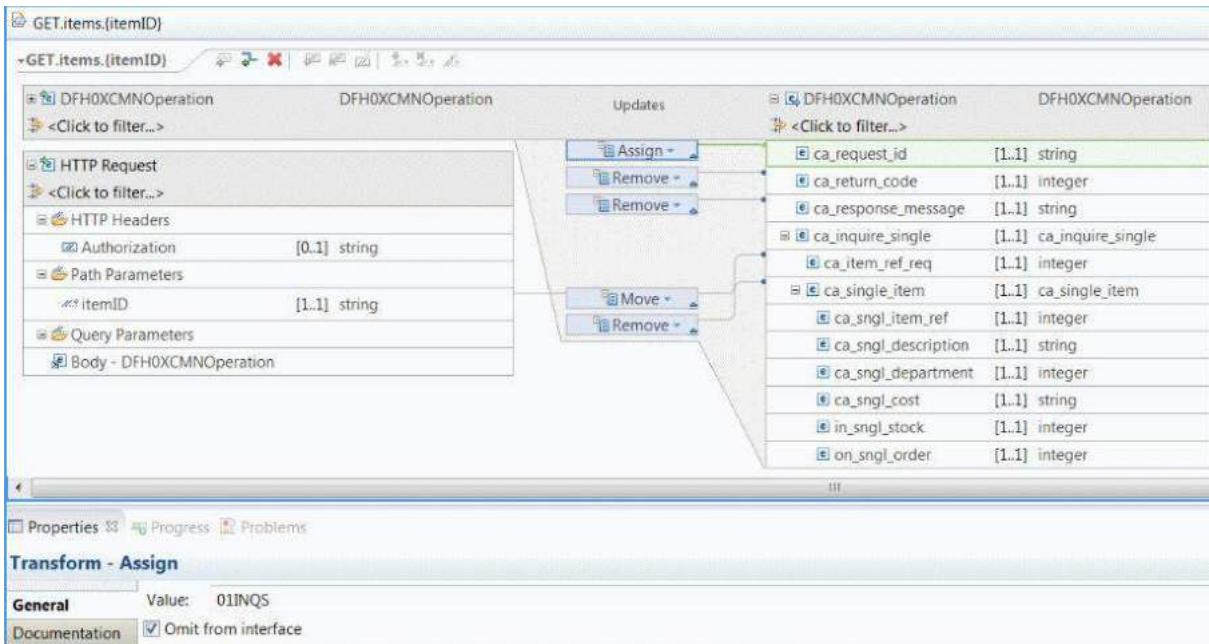
This excludes this property from the HTTP request body for this operation. This value must be set for the CICS Catalog Manager application, but because its value is the same for all requests, it does not need to be exposed to users of the API.

10. In the request mapping, expand **ca_inquire_single**. For each of **ca_return_code**, **ca_response_message**, and **ca_single_item** right click the property and click **Add Remove transform**.

This excludes these properties from the HTTP request for this operation. These values are not required on the request. They will be populated by the Catalog Manager application and returned in the response.

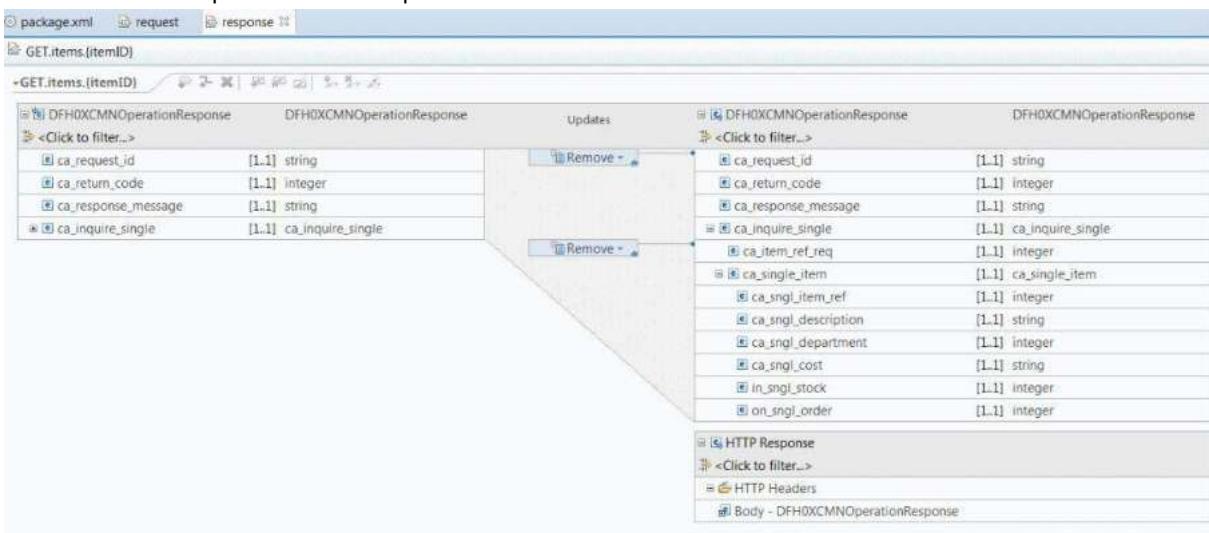
11. Connect the path parameter **itemID** to **ca_item_ref_req** by dragging from the path parameter **itemID** to the JSON property **ca_item_ref_req**.

This creates a **Move** transform that copies the value from the path parameter in the HTTP request to the property in the JSON payload passed to the z/OS Connect EE service.



12. In the response mapping, for each of `ca_request_id` and `ca_item_ref_req`, right click the property and click **Add Remove transform**.

This excludes these properties from the HTTP response body for this operation. These properties do not need to be exposed in the response to the API user.



13. Save the request and response mapping files.

Create an operation to inquire about multiple items

The following steps create an operation to inquire about multiple items in the catalog.

Before you begin

If the `inquireCatalog.sar` file was generated on a different machine, transfer the file, in binary mode, to the workstation where IBM Explorer for z/OS is installed.

About this task

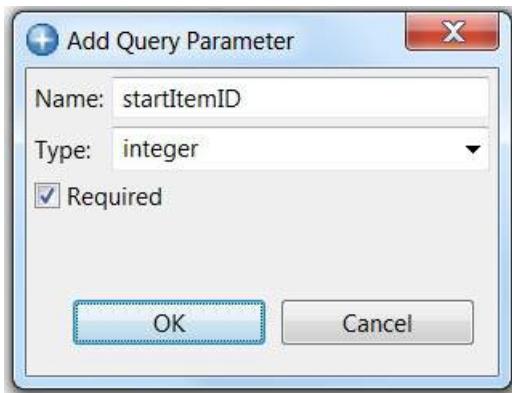
The operation is implemented as an HTTP GET request and shows how to expose a resource by using assign, move, and remove transforms that map to the `inquireCatalog` service.

Procedure

1. Select the catalog API tab and click  next to **Path** to add a new path. Enter /items into the path and delete the POST, PUT and DELETE methods.
2. Click **Service...** for the GET method to select the service archive file that defines the service on this API path that is called by an HTTP GET request.
The **Select a z/OS Connect EE Service** dialog opens.
3. Click **File System** and navigate to the location of the inquireCatalog.sar file. Select the file and click **Open**.
The **Import Services** dialog opens.
4. Click **OK**.
The inquireCatalog service now appears in the Service dialog. Click **OK**. This operation now maps to the inquireCatalog service.

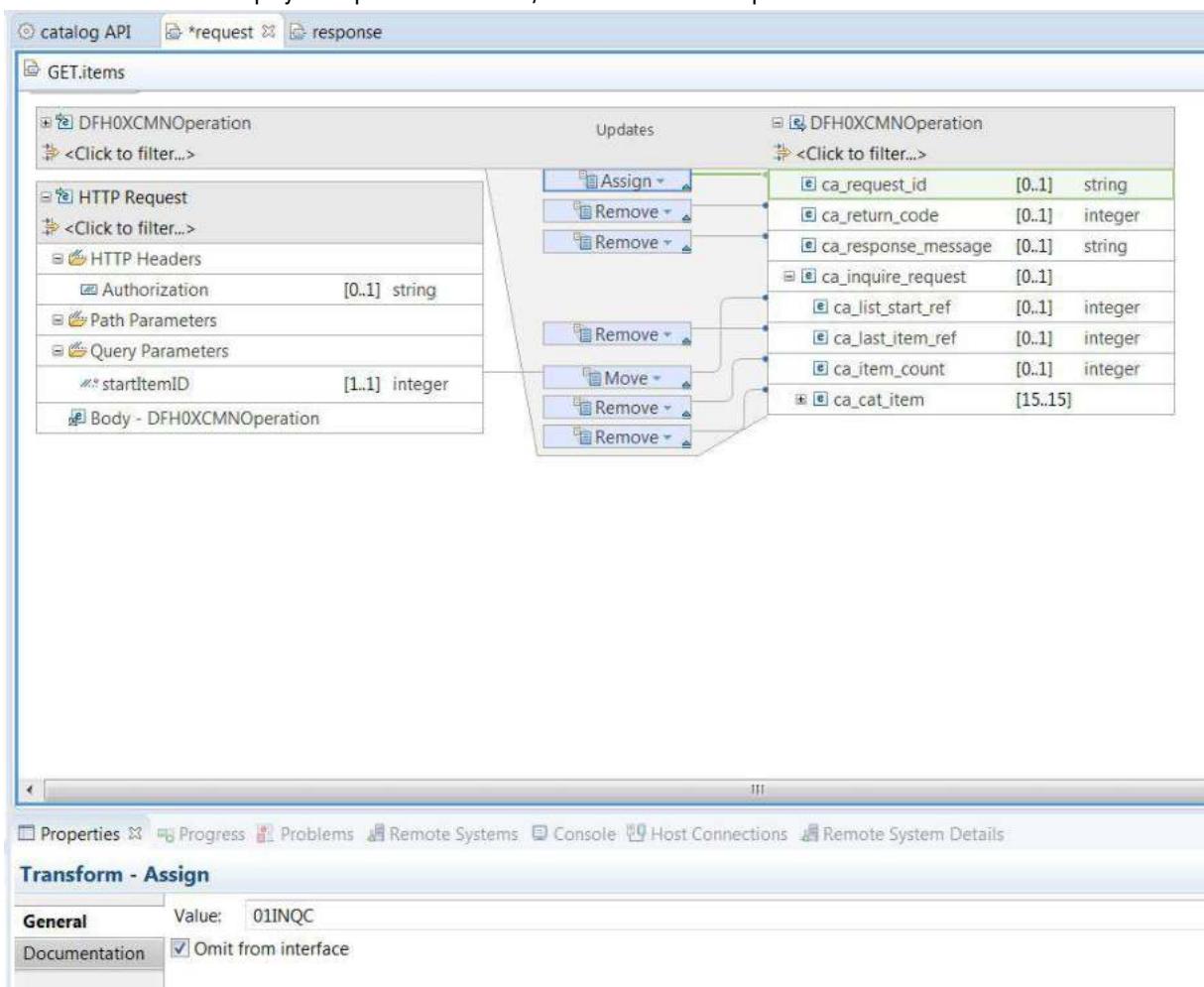


5. From the menu bar, select **File > Save**.
6. Click **Mapping...** for the GET method using the inquireCatalog service. Click **Open Both Mappings**.
The request and response mapping editors open so that you can define the mapping between the content of the API's HTTP request and response, and the JSON content passed to and from the z/OS Connect Enterprise Edition service.
7. In the request mapping, right click on `ca_request_id` and click **Add Assign transform**.
This assigns a static value to the property in the request JSON payload passed to the service.
8. Ensure **Assign** has focus, so that the **Transform - Assign** dialog is displayed in the **Properties** view. On the **General** tab, set **Value** to `01INQC`. This is the value required by the CICS Catalog Manager application to perform an inquiry on multiple catalog items.
9. Ensure **Omit from interface** is checked.
This excludes this property from the HTTP request body for this operation. This value must be set for the CICS Catalog Manager application, but because its value is the same for all requests, it does not need to be exposed to users of the API.
10. In the request mapping, expand `ca_inquire_request`. For each of `ca_return_code`, `ca_response_message`, `ca_last_item_ref`, `ca_item_count` and `ca_cat_item` right click the property and click **Add Remove transform**.
This excludes these properties from the HTTP request for this operation. These values are not required on the request. They will be populated by the Catalog Manager application and returned in the response.
11. Right click **Query Parameters** and click **Add Query Parameter**.
Fill in the name field with the name that you want to use for the query parameter. For example, `startItemID`. From the drop down menu, select **integer** for the type and check **Required**.



12. Click **OK**. Connect the path parameter **startItemID** to **ca_list_start_ref** by dragging from the path parameter **startItemID** to the JSON property **ca_list_start_ref**.

This creates a **Move** transform that copies the value from the query parameter in the HTTP request to the field in the JSON payload passed to the z/OS Connect Enterprise Edition service.



13. In the response mapping, add a **Remove** for **ca_request_id**.

14. Save the request and response mapping files.

Create an operation to place an order for an item

The following steps create an operation to place an order for an item in the catalog.

Before you begin

If the placeOrder.sar file was generated on a different machine, transfer the file, in binary mode, to the workstation where IBM Explorer for z/OS is installed.

About this task

The operation is implemented as an HTTP POST request and demonstrates exposing a resource with assign, move and remove transforms to map to the placeOrder service.

Procedure

1. Select the catalog API tab and click next to **Path** to add a new Path. Enter /orders in the input field and delete the GET, PUT and DELETE methods.
2. Click **Service...** for the POST method to select the service archive file that defines the service on this API path that will be called by an HTTP POST request.
The **Select a z/OS Connect EE Service** dialog opens.
3. Click **File System** and navigate to the location of the placeOrder.sar file. Select the file and click **Open**.
The **Import Services** dialog opens.
4. Click **OK**.

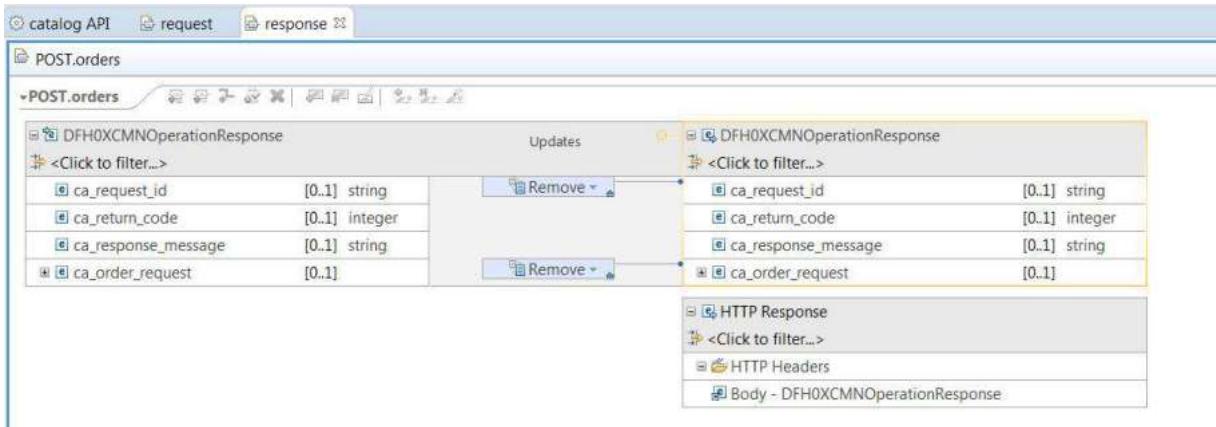
The placeOrder service now appears in the Service dialog. Click **OK**. This operation now maps to the placeOrder service.

5. From the menu bar, select **File > Save**.
6. Click **Mapping...** for the POST method and Click **Open Both Mappings**.
The request and response mapping editors open so that you can define the mapping between the content of the API's HTTP request and response, and the JSON content passed to and from the z/OS Connect Enterprise Edition service.

7. In the request mapping, right click `ca_request_id` and click **Add Assign transform**.
This assigns a static value to the property in the request JSON payload passed to the service.
8. Ensure **Assign** has focus, so that the **Transform - Assign** dialog is displayed in the **Properties** view.
On the **General** tab, set **Value** to `010RDR`.
This is the value required by the CICS Catalog Manager application to order items from the catalog.
9. Ensure **Omit from interface** is checked.
This excludes this property from the HTTP request body for this operation. This value must be set for the CICS Catalog Manager application, but because its value is the same for all requests, it does not need to be exposed to users of the API.
10. In the request mapping, expand `ca_order_request`. Add an **Assign transform** to `ca_userid` and `ca_charge_dept`.
Both these JSON properties will use the default value of an empty string, so no value needs to be specified in the Properties tab.
11. For both `ca_return_code` and `ca_response_message` right click the property and click **Add Remove transform**.
This excludes these properties from the HTTP request for this operation. These values are not required on the request. They will be populated by the Catalog Manager application and returned in the response.

The screenshot shows the IBM API Connect interface for defining a request mapping. On the left, the 'catalog API' tab is selected, and the 'request' tab is active. The 'POST.orders' operation is being edited. The 'DFH0XCMNOperation' section contains various parameters and headers. The 'Updates' section on the right lists several properties with 'Assign' transforms applied to them. A specific 'Assign' transform for `ca_request_id` is highlighted. At the bottom, a 'Transform - Assign' dialog is open, showing the 'General' tab with the value `010RDR` and the 'Omit from interface' checkbox checked.

12. In the response mapping, for each of `ca_request_id` and `ca_order_request`, right click the property and click **Add Remove transform**.
This excludes these properties from the HTTP response body for this operation. These properties do not need to be exposed in the response to the API user.



13. Save the request and response mapping files.

Define catalog API response codes

The following topic demonstrates how to define multiple response codes for the CICS catalog API methods. The steps in this topic are optional.

Before you begin

Ensure your API project is open in the z/OS Connect EE API editor. You can do this by double-clicking the package .xml file for your CICS catalog API project in the **Project Explorer** view.

Ensure that your CICS catalog service is assigned to your GET method.

About this task

In this step, you will define response codes with unique response data mapping for each of the GET and POST API methods.

Procedure

1. In the API editor, for the **GET** method, click **Mapping... > Define Response Codes** to open the response details for the GET API method.
2. Click **Add Response**
The Add Response window opens.
3. Specify a response code of **404**, and specify the following rules:
 - a) For Rule 1, specify **.../CA_RETURN_CODE** as the service field. Select **=** for the comparison operator. Enter **20** for the comparison value.
 - b) Click **OK**. Save your changes, click **File > Save** (Ctrl-S).

Rule 1	.../CA_RETURN_CODE	=	20		
--------	--------------------	---	----	--	--

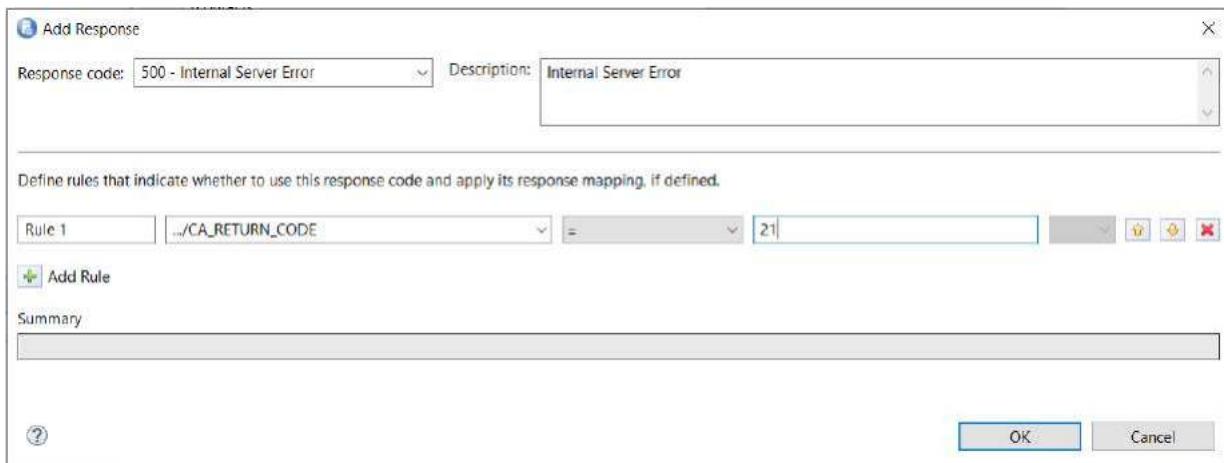
4. In the API editor, click **Mapping...** > **Define Response Codes** for the **GET** method. Click the **Add Response** button.

The Add Response window opens.

5. Specify a response code of **500**, and specify the following rule:

a) For Rule 1, specify `.../CA_RETURN_CODE` as the service field. Select `=` for the comparison operator. Enter `21` for the comparison value.

b) Click **OK**.



6. Ensure response code 200 is set as the default response code for the POST method if it is not already set.

a) Click the **Response** button for the 200 response code.

b) Click **Set Response 200 As Default**

7. Save your changes to your API project by Clicking **File** > **Save** (Ctrl-S).

Results

You have configured multiple response codes for your CICS catalog API GET and POST methods. You may optionally define unique response mappings for your response codes. Your API is now ready to deploy.

What to do next

Deploy the catalog API

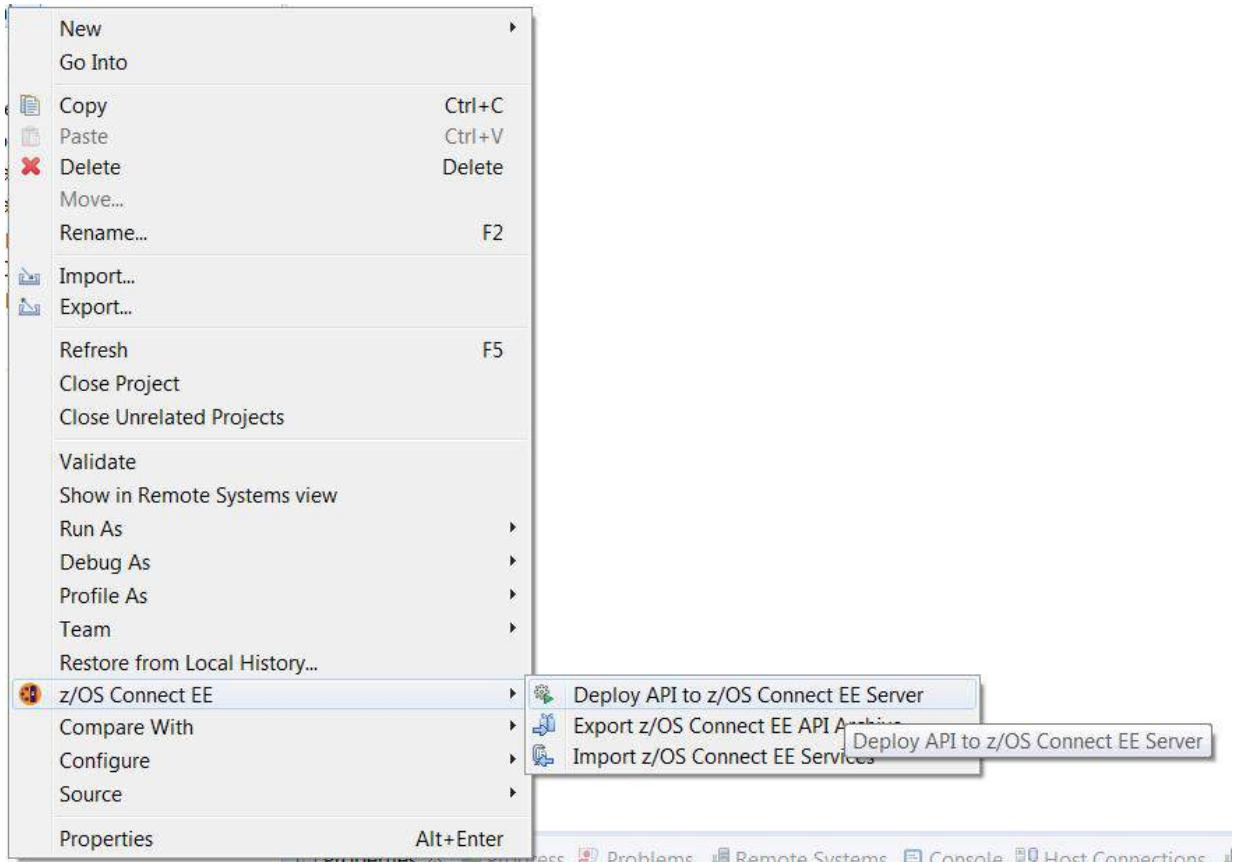
Deploy the CICS catalog manager catalog API directly from z/OS Connect EE API toolkit by first creating a connection to the server.

About this task

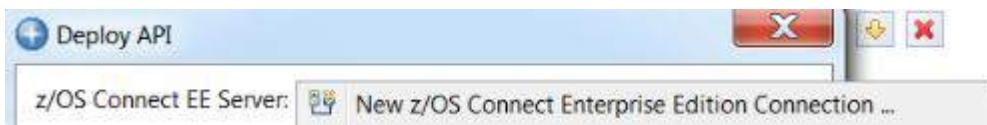
Transfer the API archive file (.aar file) to the UNIX System Services directory on the z/OS LPAR where z/OS Connect Enterprise Edition is installed.

Procedure

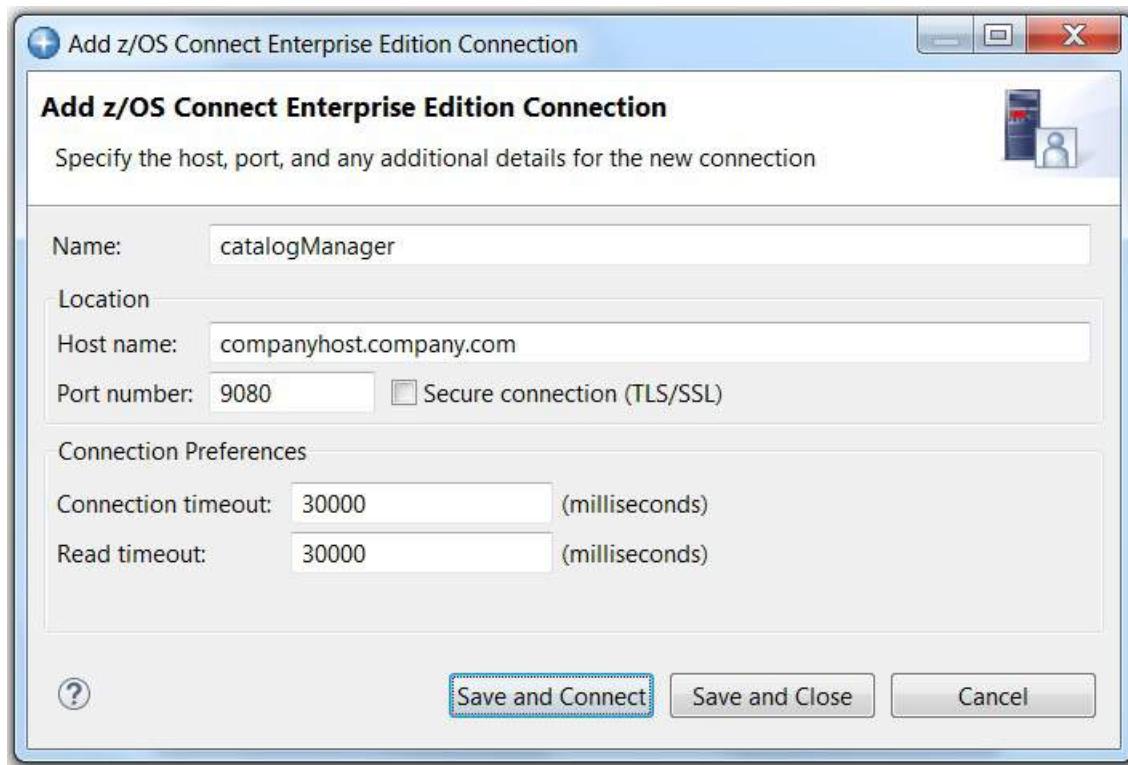
1. In the **Project Explorer** view, right click **catalog**. Click **z/OS Connect EE** > **Deploy API to z/OS Connect EE Server**.



2. In the **Deploy API** dialog, click on the server selection drop-down icon and select **New z/OS Connect Enterprise Edition Connection**.

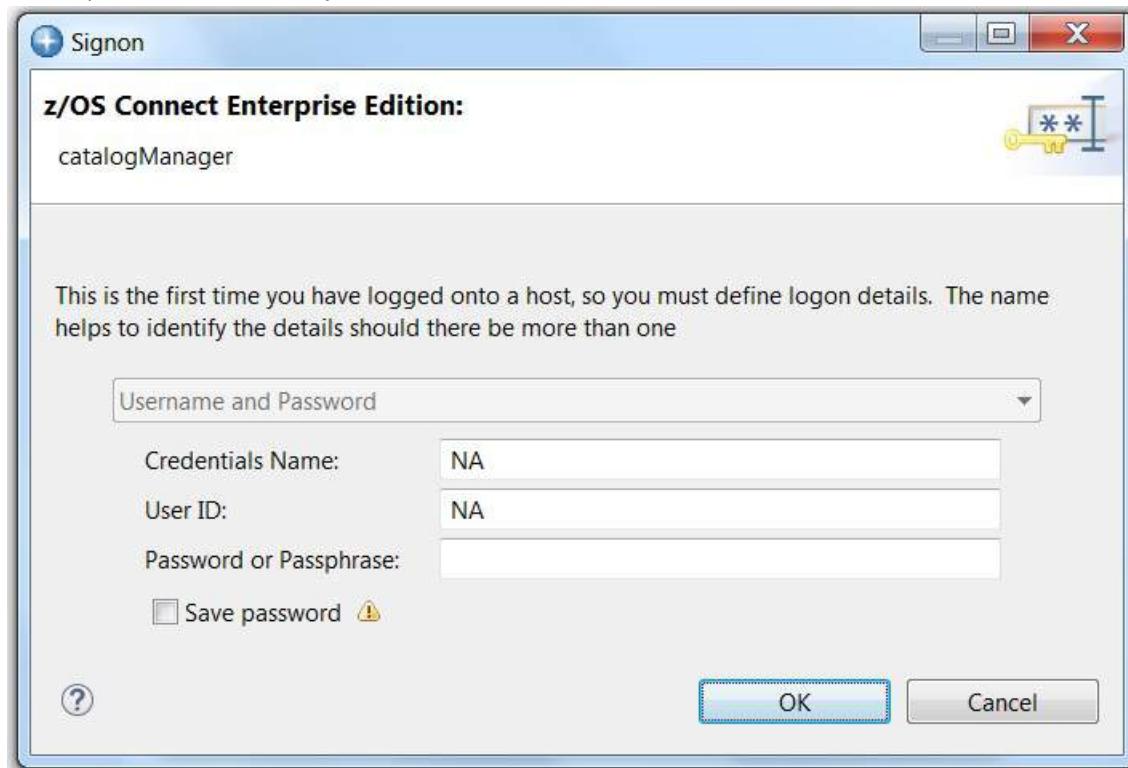


3. Enter the *host name* and *port number* for the z/OS Connect server and a name for the connection.



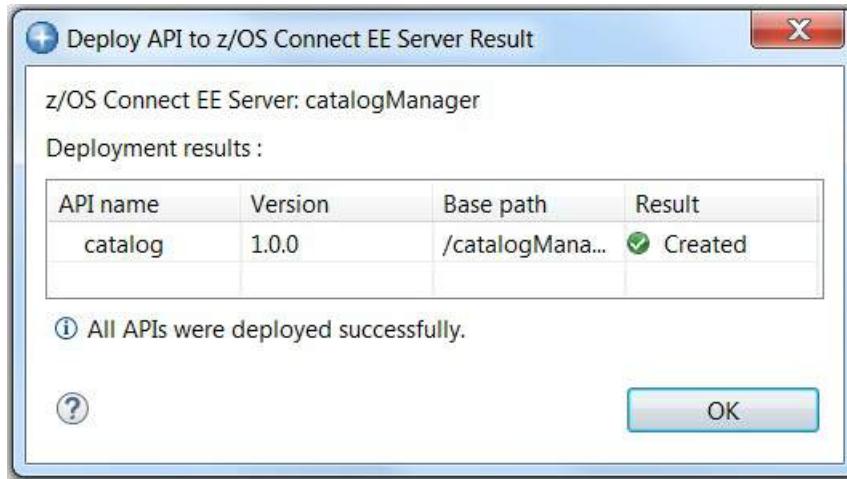
4. Click **Save and Connect**

- The **Signon** window appears because IBM Explorer for z/OS needs credentials to connect to the server.
5. Enter *NA* for the **User ID** field and click **OK**.
- The z/OS Connect EE server is configured with `requireAuth` set to `false`, so valid credentials are not required or validated by the z/OS Connect EE server.



6. Click **OK**.

The **Deploy API to z/OS Connect EE Server Result** dialog shows the results of the deployment.



7. Click **OK** to close the dialog.

Test the CICS catalog manager API

You can test your newly created API is working by invoking the catalog API directly from the z/OS Connect EE API toolkit by using the "Try it out!" function in the Swagger UI that is embedded in the editor.

Procedure

1. Start CICS and ensure that the TCPIPSERVICE is open.
2. Start your z/OS Connect EE server. For more information, see ["Starting and stopping z/OS Connect EE" on page 471](#).
 - a) Check the messages.log file for the following messages to confirm the services are installed.

BAQR7043I: z/OS Connect EE service archive placeOrder installed successfully.
BAQR7043I: z/OS Connect EE service archive inquireSingle installed successfully.
BAQR7043I: z/OS Connect EE service archive inquireCatalog installed successfully.

- b) Check the messages.log file for the following messages to confirm the API is installed.
- BAQR7000I: z/OS Connect API archive file catalog installed successfully.
3. In the **z/OS Connect EE Servers** view of the API toolkit, under the **APIs** folder, double-click on the catalog API.

The API is opened in the Swagger UI.

4. Click **List Operations** to see available operations in the API.

catalog

The screenshot shows the Swagger UI for the "catalog" API. It lists three operations: GET /items, GET /items/{itemID}, and POST /orders. The "POST /orders" row is highlighted with a green background. At the top right are "Show/Hide", "List Operations" (which is highlighted with a red box), and "Expand Operations" buttons. Below the operations is a note: "[BASE URL: /catalogManager , API VERSION: 1.0.0]".

Figure 26. Operations in the API

5. Test inquiring on an item by clicking **GET /items/{itemID}**.
 - a) Specify a value of 10 for the itemID.

The screenshot shows a REST API testing interface. At the top, it says "GET /items/{itemID} Response Class (Status 200) normal response". Below this, there are tabs for "Model" and "Example Value". The "Example Value" tab is active, displaying a JSON snippet:

```
{
  "DFH0XCP4": {
    "CA_RETURN_CODE": 0,
    "CA_RESPONSE_MESSAGE": "string",
    "CA_INQUIRE_SINGLE": {
      "FILL_0": 0,
      "FILL_1": 0,
      "CA_SINGLE_ITEM": {
        "CA_SNGL_ITEM_REF": 0,
        "CA_SNGL_DESCRIPTION": "string"
      }
    }
  }
}
```

Below the JSON, there's a "Response Content Type" dropdown set to "application/json". Under "Parameters", there are two entries:

Parameter	Value	Description	Parameter Type	Data Type
itemID	10		path	string
Authorization			header	string

At the bottom left is a "Try it out!" button, and at the bottom right is a "Hide Response" link.

Figure 27. Testing the GET request

b) Click **Try it out!**.

Information about the request URL, request headers, response body, response code, and response headers are provided. The response body contains the output message:

- "CA_SNGL_DESCRIPTION": "Ball Pens Black 24pk"

Results

You have verified that your newly created catalog API works to CICS.

Create an API to invoke the IMS phone book service

Use the z/OS Connect EE API toolkit to develop a REST API to invoke the IMS phone book service.

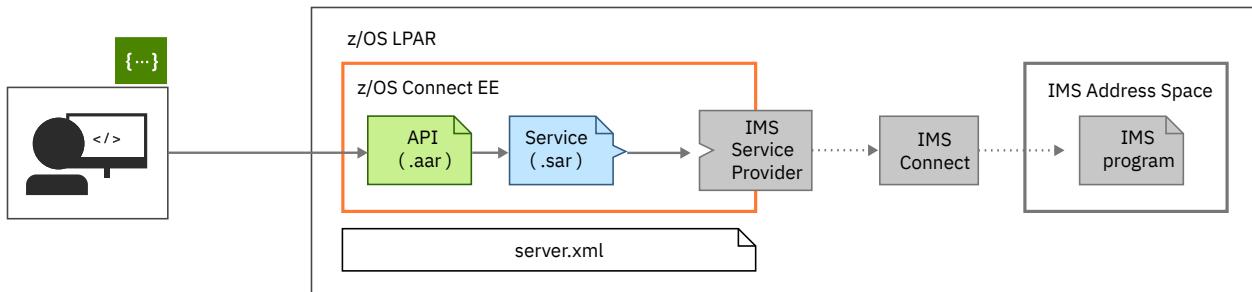
Before you begin

To create and test the catalog API ensure that the following tasks are complete:

1. Install and configure the IMS phone book sample application. The procedure is described in the scenario ["Prepare the sample IMS application"](#) on page 61
2. Configure and test your connection to IMS. The procedure is described in the scenario ["Create a server and connect to IMS"](#) on page 72 and installs relevant artifacts using the supplied configuration template `sampleImsIpicCatalogManager`. This includes the phonebook API.
3. Use the z/OS Connect EE API toolkit to Stop the phonebook API, and then Remove the phonebook API.

About this task

This task shows you how to create and test an API to access the IMS sample phone book application.



This scenario uses the following values:

- IMS region = IMS1
- IMS program = PROG1
- IMS transaction code = IVTNO

The JSON schema of an IMS mobile service is obtained from the service archive (.sar) file. You then map an HTTP method to a service, assign a value to a field in the service, and remove unrequired fields.

After the mapping is done, the API can be deployed to the server. After the service archive file is deployed to the server by putting the file in the defined services directory, you can immediately test the API, all from within the API toolkit.

Procedure

Perform the following tasks in the given sequence to create the API.

Design the API for the service

When composing the API, first identify the resources that can be acted upon by the service. Then plan the URI paths, the use of path or query parameters, and any requirements for input.

About this task

For the phonebook service, the resource is the contact record. The actions against the contact record are to add, display, update, and delete a contact record.

Procedure

1. Identify which fields need to be available to accept input (request).

The following table identifies the fields in the phonebook service that need to be exposed, hidden, or assigned a value. It also demonstrates the API design thought process.

Table 4. Input fields for the phonebook service

Input field	Action	Description
IVTNO-INPUT-MSG	Hide	An "01" level element for the record.
IN-COMMAND	Expose	The API must pass in this value. The values must be either ADD, DISPLAY, DELETE, or UPDATE.
IN-LAST-NAME	Expose	The key field for the database. This value must be unique. Therefore, this field must be exposed.
IN-FIRST-NAME	Expose	This field is needed for ADD and UPDATE actions.
IN-EXTENSION	Expose	Same as for IN-FIRST-NAME.
IN-ZIP-CODE	Expose	Same as for IN-FIRST-NAME.

2. Identify the fields that need to be available for output (response).

The following table identifies the fields that need to be made available in the response.

Table 5. Output fields for the phonebook service

Output field	Action	Description
IVTNO-OUTPUT-MSG	Hide	An "01" level element for the record.
OUT-MESSAGE	Expose	The message that indicates the result of the action. This field needs to be exposed to the API.
OUT-COMMAND	Expose	This field will be made available to the API to be exposed to the client application if the API developers intend to do so. A potential use for this field is a simple verification of the command used to achieve the result. This can be hidden if you don't see a use case for it.
OUT-LAST-NAME	Expose	This field will be made available to the API to be exposed to the end users. For all four actions, this field should be displayed for the end users.
OUT-FIRST-NAME	Expose	This field will be made available to the API to be exposed to the client applications if the API developers intend to do so. For ADD, UPDATE and DISPLAY, this information can be useful for end user validation purposes. For DELETE, this field has less value and can be hidden by the API.
OUT-EXTENSION	Expose	Same as for OUT-FIRST-NAME.
OUT-ZIP-CODE	Expose	Same as for OUT-FIRST-NAME.
OUT-SEGNO	Hide	Internal to the IMS transaction. No need to expose this field to the API.

3. Identify the resource and actions that are allowed on the resource.

The resource that is being exposed by this service is the phonebook entry, or contact record. The goal of the API we are creating is to allow users to add, update, display, and delete a contact record.

Table 6. Goals of the API

Action	IN-COMMAND value	HTTP Verb
Add a contact	ADD	POST
Update a contact	UPDATE	PUT
Display a contact	DISPLAY	GET
Delete a contact	DELETE	DELETE

4. Identify URI paths and query parameters.

The next step is to plan the URI paths, the use of path or query parameters, and input JSON requirements:

Table 7. URI path planning

Action	HTTP verb	Path
Add a contact	POST	/phonebook/contacts + JSON body The HTTP verb POST implies the action ("add"), so in the API mapping we will assign the value ADD to the IN-COMMAND field. The JSON will carry in the four contact record values: last name, first name, extension and zip code.
Update a contact	PUT	/phonebook/contacts/{lastName} + JSON body The HTTP verb PUT implies the action ("update"), so in the API mapping we will assign the value UPDATE to the IN-COMMAND field. The resource to act upon is specified as a path parameter, which will get mapped to the LAST-NAME field. The JSON will carry in the three other contact record values: first name, extension and zip code.
Display a contact	GET	/phonebook/contacts/{lastName} The HTTP verb GET implies the action ("retrieve" or "display"), so in the API mapping we will assign the value DISPLAY to the IN-COMMAND field. The transaction supports the display of a single contact record, not a range.
Delete a contact	DELETE	/phonebook/contacts/{lastName} The HTTP verb DELETE implies the action ("delete"), so in the API mapping we will assign the value DELETE to the IN-COMMAND field. The transaction supports the delete of a single contact record, not a range. Therefore, we use a path parameter. The API will map this path parameter value to the last name field.

5. Identify output for each action.

The output for each action is as follows:

Table 8. Output for each action

Action	Fields to return
Add a contact	<ul style="list-style-type: none">• OUT-MESSAGE: To indicate success or failure of the action.• LAST-NAME, FIRST-NAME, EXTENSION, ZIP-CODE: To allow for validation of the added contact record.
Update a contact	The same as adding a contact.
Display a contact	The same as adding a contact.
Delete a contact	<ul style="list-style-type: none">• OUT-MESSAGE: To indicate success or failure of the action.• LAST-NAME: To allow for validation of the deleted record.

Results

With the API design identified, you are ready to create the API.

Create the API project

Use the z/OS Connect EE API toolkit to create an API project and define the API name and base path.

About this task

You can create an API project and define the API in the **z/OS Connect Enterprise Edition** perspective.

Procedure

1. Open the Eclipse tool in which you installed the z/OS Connect EE API toolkit.
2. Switch to the z/OS Connect Enterprise Edition perspective.
 - a) From menu bar, select **Window > Open Perspective > Other**.
 - b) In the list of perspectives, select **z/OS Connect Enterprise Edition** and click **OK**.
You are now in the **z/OS Connect Enterprise Edition** perspective, with related views and resources readily available.
3. Create an API project.
 - a) From the menu bar, select **File > New > z/OS Connect EE API Project**.
The z/OS Connect EE API Project wizard opens.
 - b) Enter the project properties:

Table 9. Input fields for the phonebook service

Project property	Description	Sample value to specify
Project name	Unique alphanumeric name for your project. This is the name of the project in Eclipse.	phonebook
API name	The name of your API. This is the name by which the z/OS Connect EE server knows about this API.	contacts
Base path	The unique basePath attribute that specifies the root of all the resources in this API. This path is used by REST clients in the URI they send in to invoke the API.	/phonebook
Description	Optional field to provide a description of this API for documentation purposes.	This is an API for managing contacts.

- c) Click **Finish**.

Results

The API project is created in the Project Explorer view. The API package .xml file opens in a tab that is named after your project. You can now model your API.

Create the contacts API

Create an API to add, display, update and delete contacts from the IMS phonebook.

About this task

In this step, you will import the service archive (.sar) file for the phonebook service into the API project you created in z/OS Connect EE API toolkit, and map the input and output messages for each HTTP verb (action).

For the API design, our goal is to create the following APIs for adding, updating, displaying, and deleting a contact record, as defined in “[Design the API for the service](#)” on page 132. The path of the API is /<base_path>/<api_name>, and in this case, the path is /phonebook/contacts

Table 10. URI path planning

Action	HTTP verb	Path
Add a contact	POST	/phonebook/contacts
Display a contact	GET	/phonebook/contacts/{lastName}
Update a contact	PUT	/phonebook/contacts/{lastName}
Delete a contact	DELETE	/phonebook/contacts/{lastName}

Two paths are needed:

- /phonebook/contacts: This path is needed only by the POST action (ADD).
- /phonebook/contacts/{lastName}: This path is needed by the GET, PUT, and DELETE methods, and you will need to define last name as a parameter.

Procedure

Define a /phonebook/contacts path and an associated POST method:

1. In the **Project Explorer** view, right-click the **phoneBook** folder (or your project name, if you use a different value), and select **z/OS Connect EE > Import z/OS Connect EE Services**.
The **Import z/OS Connect EE Services** wizard opens.
2. Depending on where you exported the service archive file, click **Workspace** or **File System** and select the .sar file to import it into your API project.
3. Click **OK**.

For the POST verb to add contact information for a given last name, in the next step you will:

- a. Define the path value.
 - b. Map the request message for the action.
 - c. Map the response message for the action.
4. In the z/OS Connect EE API toolkit where the package.xml is displayed, set the path to /contacts.
The default base path for this API is /phonebook, so it does not need to be included.

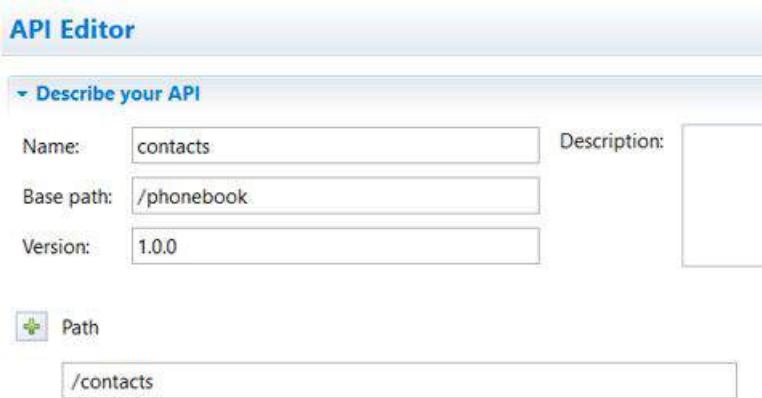


Figure 28. Adding a path for the API

For the POST action, we want to ADD a contact record, and therefore:

- IN_COMMAND needs to be assigned a value of ADD. This field should be hidden from end users whenever this API is called.
- The other input fields can be left alone with their default mapping. These fields should be allowed to be exposed to end users for them to specify the values.

5. Remove the GET, PUT, and DELETE methods by clicking the remove button (next to these methods.

You have created the front end of your API. Now the API needs to be connected to the back-end application in IMS.

To associate a service with the API:

6. For the POST method, click **Service** next to the method.

The **Select a z/OS Connect EE Service** window opens.

7. Select the phonebook service archive file you imported in an earlier step and click **OK**.

8. Save your project by clicking **File > Save** (or press Ctrl-S).

Now map your API request to fields in the service:

9. Click the **Mapping** button next to the POST method and select **Open Request Mapping**.

If you have not saved your project or latest changes, a dialog opens to prompt you to save the package.xml file. Click **OK** to save the changes, and the request mapping editor opens.

Transferred	
	IVTNO_INPUT_MSG
	<Click to filter...>
	IN_TRANCODE [0..1] string
	IN_COMMAND [0..1] string
	IN_LAST_NAME [0..1] string
	IN_FIRST_NAME [0..1] string
	IN_EXTENSION [0..1] string
	IN_ZIP_CODE [0..1] string

Figure 29. Request mapping for the POST method

10. Expand the IVTNO_INPUT_MSG section on the left by clicking the expand button (+).

The following image shows the expanded IVTNO_INPUT_MSG section.

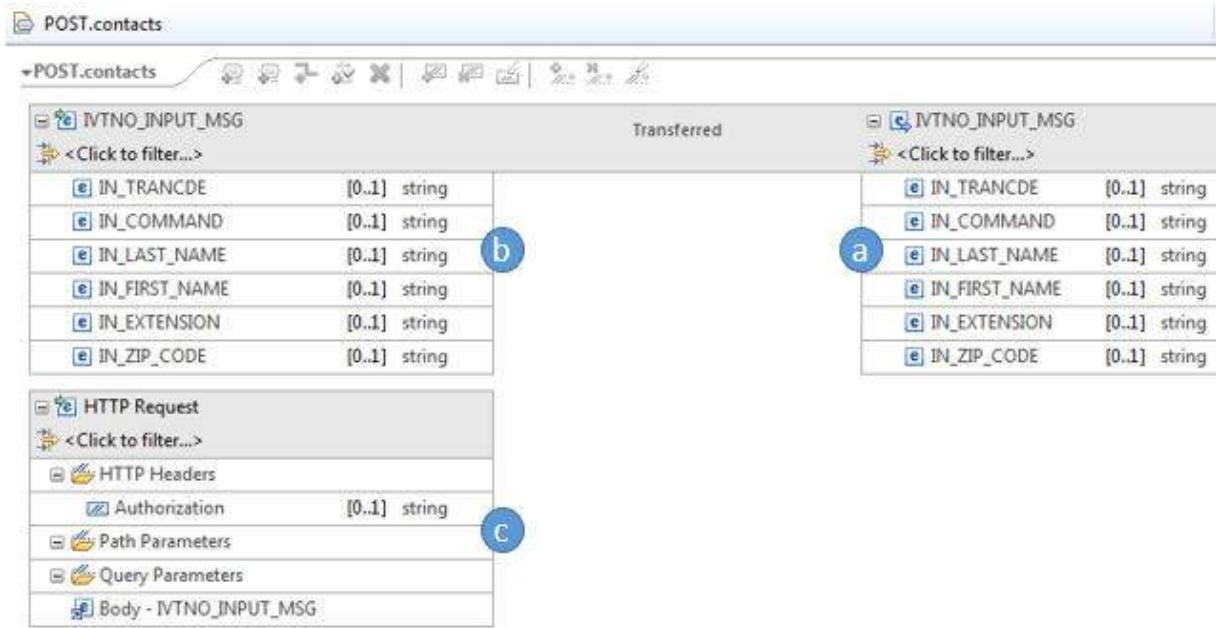


Figure 30. Request mapping for the POST method

- The right side of the interface (labeled with the letter "a") shows the fields that are exposed by the service for the request.
- The upper left of the interface (labeled with the letter "b") shows fields that will be exposed through the API and made available to the REST clients. By default, it is a one-to-one mapping to the fields that are exposed by the service unless you change the mapping.
- The lower left of the interface (labeled with the letter "c") shows the HTTP request, which can consist of one or more headers, path parameters, query parameters, and the request body.

For the POST action, we want to ADD a contact record with the following goals:

- IN_COMMAND needs to be assigned a value of ADD. This field should be hidden from end users whenever this API is called.
- The other input fields can be left alone with their default mapping. These fields should be allowed to be exposed to end users for them to specify the values.

We will achieve these goals in the next few steps.

- Right-click the IN_COMMAND field in the service definition (section A in Figure 3, and select **Add Assign transform**. An Assign box shows up in the middle.
- Click the **Assign** box, and the **Transform - Assign** properties view opens.
- In the Value field, enter ADD, and leave the **Omit from interface** checkbox checked to exclude this value from the Swagger document so it is not exposed in the API.

The screenshot shows the Mule Studio interface for defining a request mapping. At the top, there's a toolbar with various icons. Below it, the main workspace shows two message structures side-by-side:

- Left Message Structure (INVTNO_INPUT_MSG):**
 - Fields: IN_TRANCODE, IN_COMMAND, IN_LAST_NAME, IN_FIRST_NAME, IN_EXTENSION, IN_ZIP_CODE.
 - Type: [0..1] string.
- Right Message Structure (OUTVTNO_INPUT_MSG):**
 - Fields: IN_TRANCODE, IN_COMMAND, IN_LAST_NAME, IN_FIRST_NAME, IN_EXTENSION, IN_ZIP_CODE.
 - Type: [0..1] string.

In the center, there's a green 'Assign' node with a small dropdown arrow. Two red arrows point from this node towards the 'Value' field in the 'Transform - Assign' dialog below. The dialog has tabs for 'General' (selected), 'Value' (set to 'ADD'), and 'Documentation' (with a checked 'Omit from interface' checkbox). At the bottom of the dialog, there's a red arrow pointing to the 'Value' field.

Figure 31. Assigning a fixed value to the IN_COMMAND field

14. Click **File > Save** (or press Ctrl-S) to save your mapping.
 15. Close the request mapping tab.
- You have defined how the request message should be mapped. You will now define what will be in the response message.
16. Click the **Mapping** button next to the POST method and select **Open Response Mapping**.
 17. For output fields, we don't need to send OUT_COMMAND back to the user. Remove it by right-clicking OUT_COMMAND and select **Add Remove transform**.

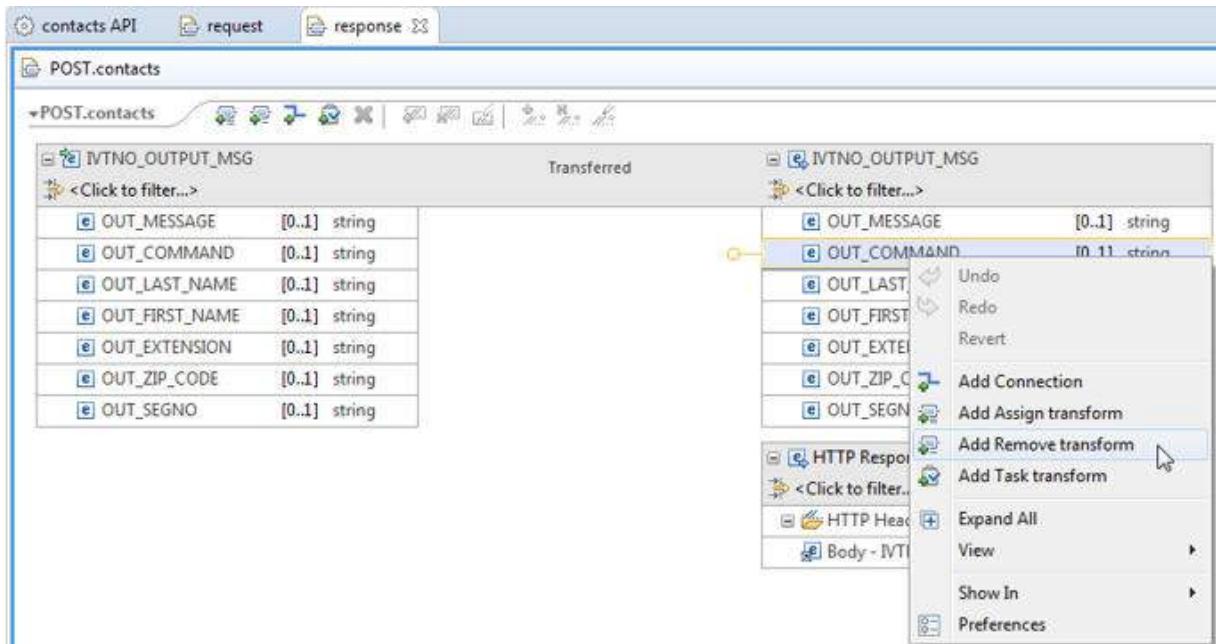


Figure 32. Removing a field from the response message

18. Click **File > Save** (or press Ctrl-S) to save your mapping.
19. Close the response mapping tab.

Results

You have defined the POST method to add a contact record and how the request and response messages are mapped to the data fields in the service. The next step is to define a new path that will take `{lastName}` as a parameter for the GET, PUT, and DELETE methods. You can skip this step and proceed directly to deploy and test the API if you already understand how the API design and development process works.

Create additional phonebook API methods

This topic shows you how to define another path and data mapping for the PUT, GET, and DELETE verbs. The steps in this topic are optional.

About this task

In this step, you will define a new path, `/phonebook/contacts/{lastName}`, and the data mapping for the associated PUT, GET, and DELETE verbs.

Table 11. Output for the display and delete actions

Action	HTTP Verb	Fields to return
Display a contact	GET	<ul style="list-style-type: none"> • IN-COMMAND to be assigned a value of DISPLAY. • Fields to return: <ul style="list-style-type: none"> – OUT-MESSAGE: To indicate success or failure of the action. – LAST-NAME, FIRST-NAME, EXTENSION, ZIP-CODE

Table 11. Output for the display and delete actions (continued)

Action	HTTP Verb	Fields to return
Update a contact	PUT	<ul style="list-style-type: none"> • IN-COMMAND to be assigned a value of UPDATE. • Fields to return: <ul style="list-style-type: none"> – OUT-MESSAGE: To indicate success or failure of the action. – LAST-NAME, FIRST-NAME, EXTENSION, ZIP-CODE: To allow for validation of the updated contact record.
Delete a contact	DELETE	<ul style="list-style-type: none"> • IN-COMMAND to be assigned a value of DELETE. • Fields to return: <ul style="list-style-type: none"> – OUT-MESSAGE: To indicate success or failure of the action. – LAST-NAME: To allow for validation of the deleted record.

Procedure

To display, update, or delete a contact record for a given last name, define the path value and the map the request and response message for the action:

1. In the z/OS Connect EE API toolkit where the package .xml is displayed, add a path by clicking the **Add a new path** button () next to the "Path" label.

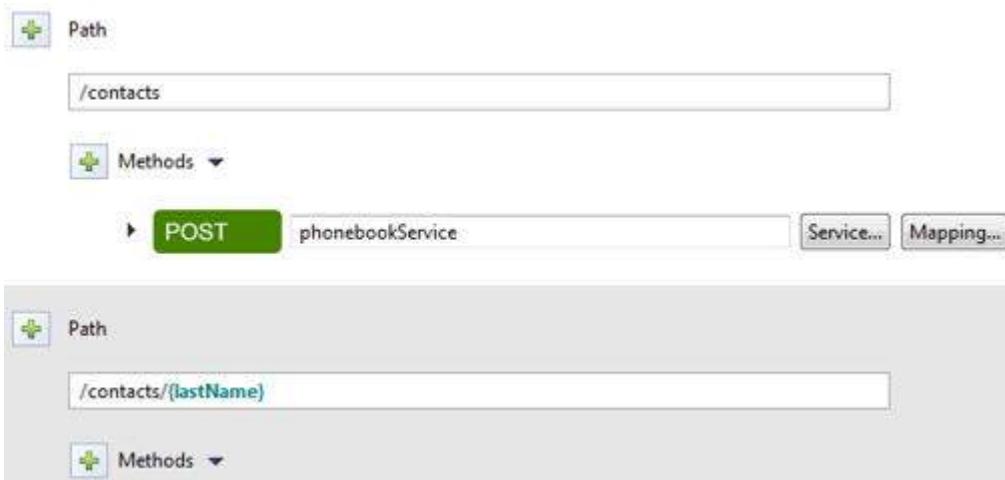


Figure 33. Adding another path for the API

2. Enter the API path and the path parameter, lastName, enclosed in curly brackets. In this case, it is / contacts/{lastName}

3. Remove the POST method by clicking the remove button () next to the method.

The GET method

For the GET method (querying a contact record by last name), map the data structure between the service JSON schema and the HTTP request and response messages.

4. For the GET method, click **Service** next to the method, select the phonebook service archive file, and click **OK**.
5. Click **File > Save** or press Ctrl-S to save the service association.
6. Click **Mapping** next to the GET method and select **Open Request Mapping**.
7. Right-click the IN_COMMAND field in the service definition on the right, and select **Add Assign transform**.

An Assign box shows up in the middle.

8. Click the **Assign** box, and the **Transform - Assign** properties view opens.
9. For the **Value** field, enter DISPLAY.
10. Because only the last name is needed for the request and the other fields are not needed, remove those fields so they are hidden from the client.
 - a) Hold down the Ctrl key to select multiple fields: IN_EXTENSION, IN_FIRST_NAME, and IN_ZIP_CODE.
 - b) Right-click and select **Add Remove transform**.

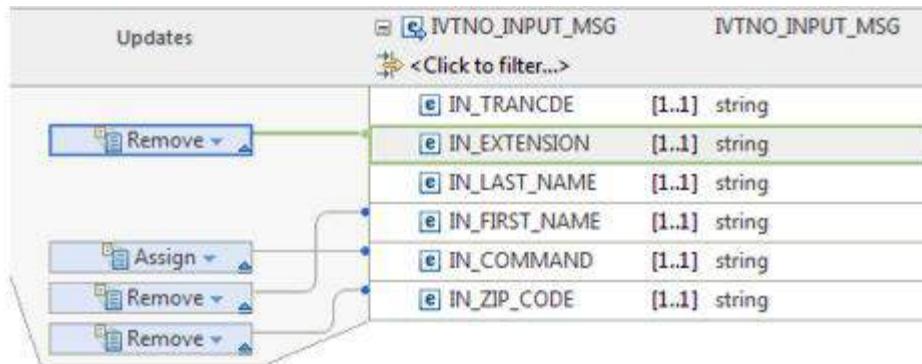


Figure 34. Removing unneeded fields from the requests

The last step is to map the path parameter **lastName** from the HTTP Request section to the IN_LAST_NAME field.

Tip: You can right-click the field to undo an action, or right-click the added action button to delete the action.

11. Click and drag the **lastName** path parameter field in the HTTP Request section to the IN_LAST_NAME field on the right.

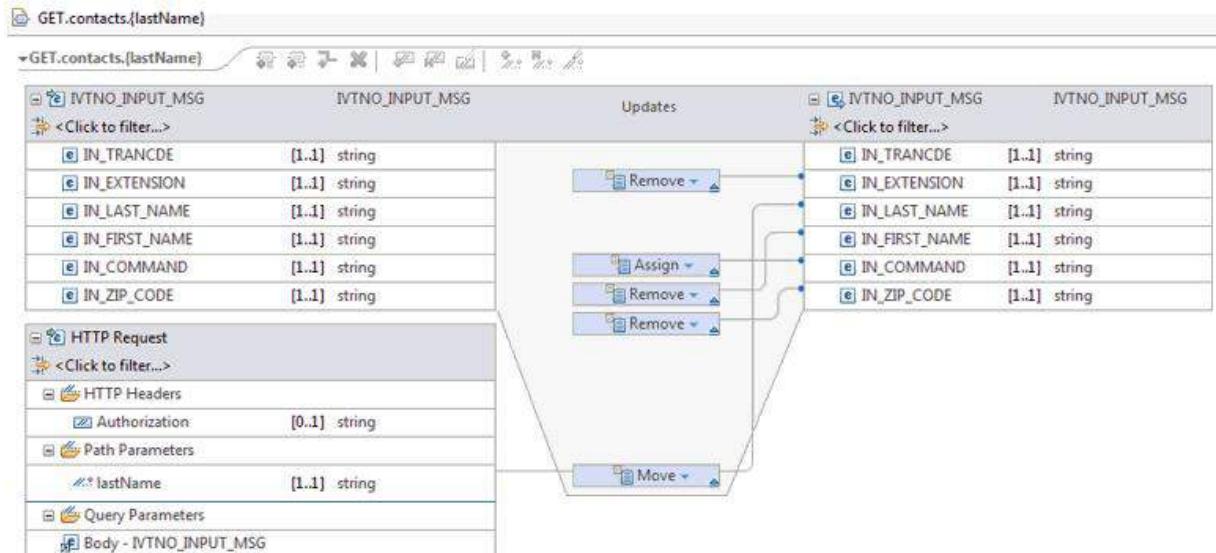


Figure 35. Mapping the lastName path parameter to the IN_LAST_NAME field

No JSON body is sent in with this request. All the information needed to display a contact record is carried on the URI with the path parameter.

12. Click **File > Save** (Ctrl-S) to save your request mapping.
13. Close the request mapping tab.

The next step is to map the response. For the given last name, we want to return all fields except the OUT_COMMAND field, so we need to remove this field from the response.

14. Click **Mapping** next to the GET method and select **Open Response Mapping**.

15. Right-click OUT_COMMAND on the right, and select **Add Remove transform**.

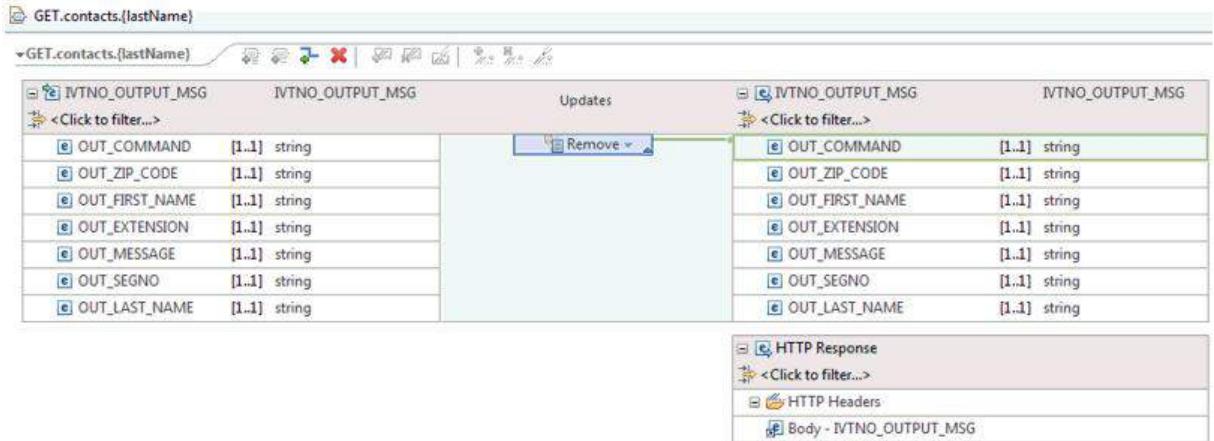


Figure 36. Removing the OUT_COMMAND field from the response

16. Click **File > Save** (Ctrl-S) to save your response mapping and close the response tab.

The PUT method

For the PUT method (updating a record), the request and response mapping is similar to the GET method. The key tasks are as follows:

- Assign a service that is associated with this method.
- Assign a static value to the IN_COMMAND field. The command to update a record in the phonebook application is UPDATE, so set the static value to UPDATE.
- The **lastName** path parameter is the key and its value needs to be moved to the IN_LAST_NAME field.
- The IN_FIRST_NAME, IN_EXTENSION and IN_ZIP_CODE fields need to be exposed in the JSON body for this request for users to update.

17. Map the request for the PUT method so the mapping looks as follows:

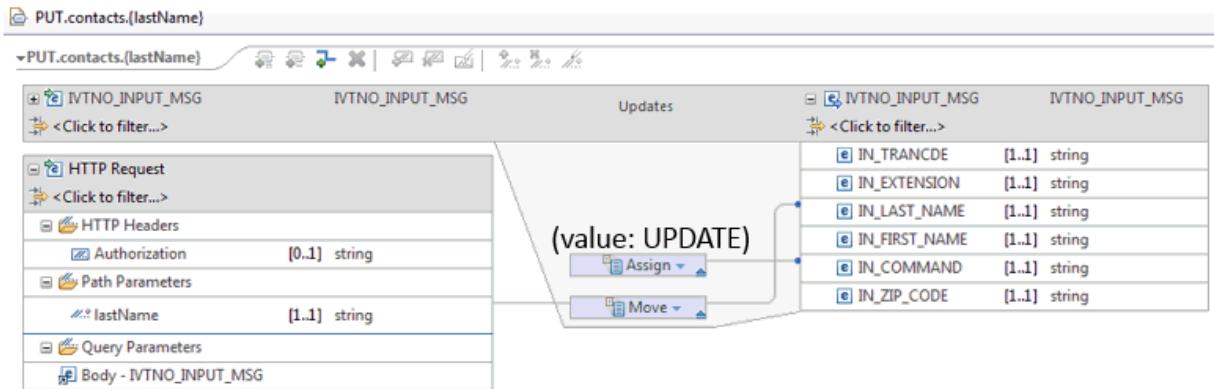


Figure 37. Request mapping for the PUT method

18. Save your changes.

19. Close the request tab.

Next map the response message. All fields should be returned to allow users to validate their updates.

20. Map the response for the PUT method as follows:

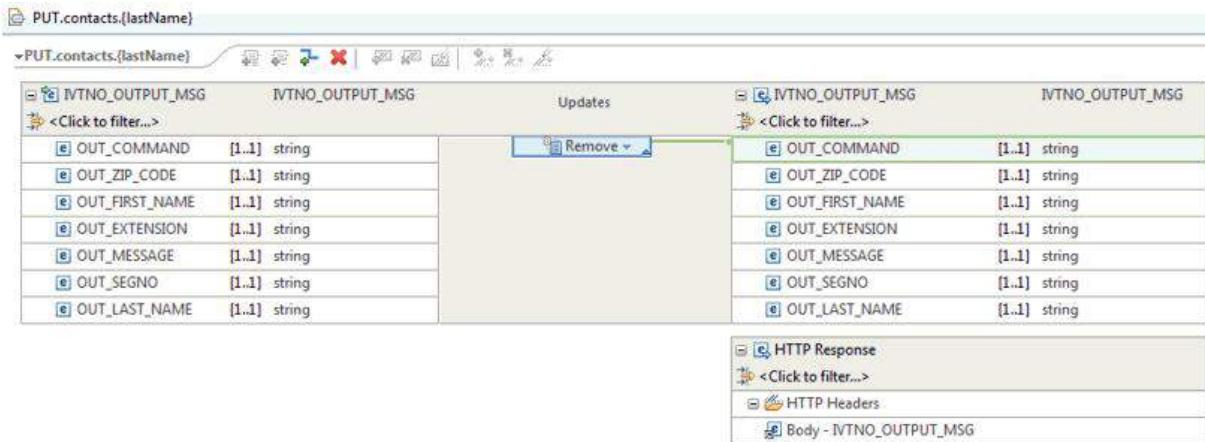


Figure 38. Response mapping for the PUT method

21. Save your changes and close the response tab.

Tip: With this mapping design, if only the extension for a specified last name is provided in the request message, the first name and zip code fields will be updated as blank. Ensuring that field values are not incorrectly wiped out would be the responsibility of the mobile application developer.

The DELETE method

For the DELETE method (deleting a record), the key mapping tasks are as follows:

- Assign the service that is associated with this method and save
- Assign a static value to the IN_COMMAND field. In this case, the command to delete a record in the phonebook application is DELETE, so set the value to DELETE.
- The **lastName** path parameter is the key and its value needs to be moved to the IN_LAST_NAME field.
- No fields need to be exposed in the request, so IN_EXTENSION, IN_FIRST_NAME, and IN_ZIP_CODE should be removed.

22. Map the request for the DELETE method so the mapping looks as follows:

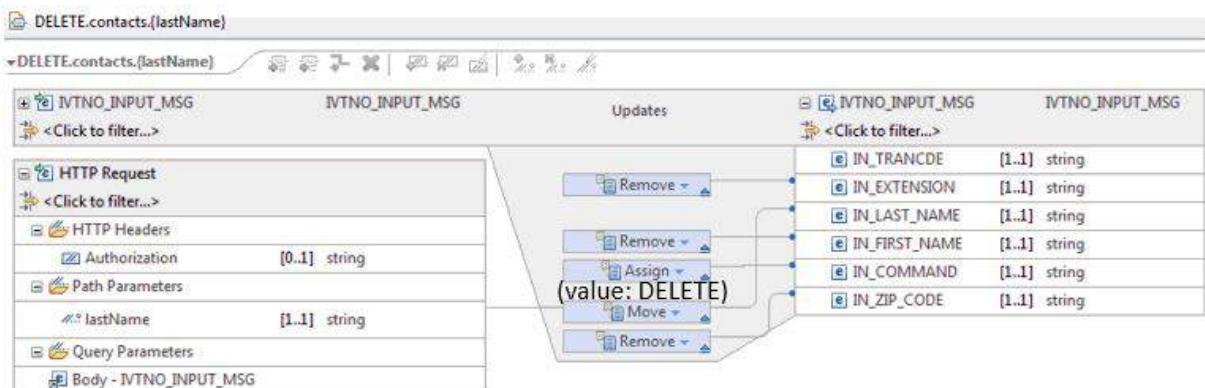


Figure 39. Request mapping for the DELETE method

23. Save your changes and close the request tab.

Next map the response message. For a delete operation, all we are interested in is whether the action succeeded. This information is indicated in OUT_MESSAGE. Therefore all fields, except OUT_MESSAGE, should be removed.

24. Map the response for the DELETE request as follows:

Figure 40. Response mapping for the *DELETE* method

25. Save your changes and close the response tab.

Results

The API is now completed.

Define contacts API response codes

The following topic demonstrates how to define multiple response codes for the IMS phone book API methods. The following steps in this topic are optional.

Before you begin

Ensure your API project is open in the z/OS Connect EE API editor. You can do this by double-clicking the package.xml file for your IMS contacts API project in the **Project Explorer** view.

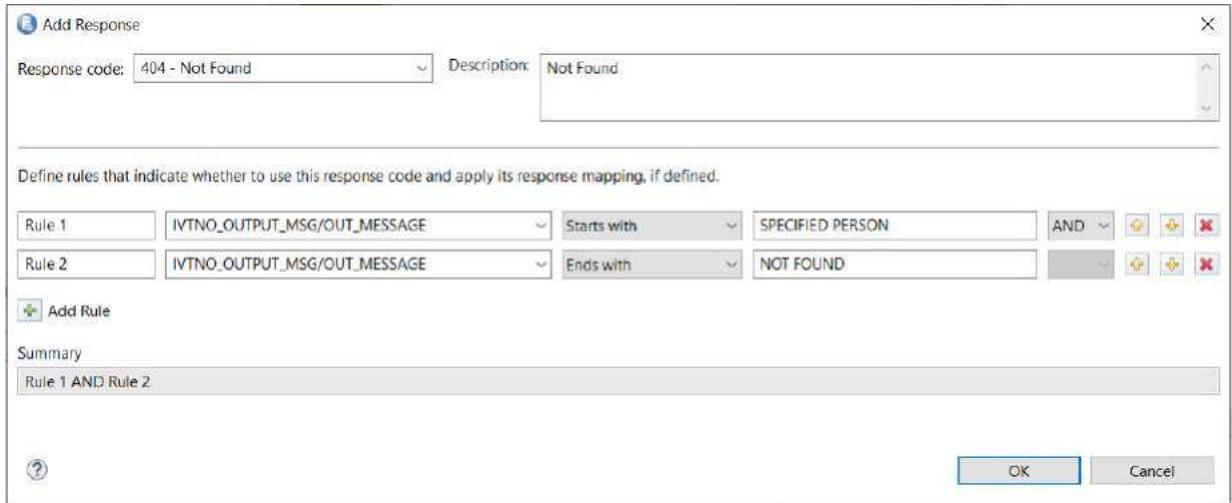
Ensure that your phone book service is assigned to your GET and POST API methods.

About this task

In this step, you will define response codes with unique response data mapping for each of the GET and POST API methods.

Procedure

1. In the API editor, for the **GET** method, click **Mapping... > Define Response Codes** to open the response details for the GET API method.
2. Click **Add Response**
The Add Response window opens.
3. Specify a response code of **404**, and specify the following rules:
 - a) For Rule 1, specify **IVTNO_OUTPUT_MSG/OUT_MESSAGE** as the service field. Select **Starts with** for the comparison operator. Enter **SPECIFIED PERSON** for the comparison value. Click the **Add Rule** () button to add a new rule, and specify the logical operator **AND** next to Rule 1.
 - b) For Rule 2, specify **IVTNO_OUTPUT_MSG/OUT_MESSAGE** as the service field. Select **Ends with** for the comparison operator. Enter **NOT FOUND** for the comparison value.
 - c) Click **OK**. Save your changes, click **File > Save** (Ctrl-S).



4. Define a unique response mapping for the 404 response.

a) Click **Mapping...** > **Open response 404 mapping** next to the 404 response.

The field mapping editor opens.

b) Under the **Body - response** section, right-click each of the following fields and select **Add Remove transform**:

- lastName
- firstName
- extension
- zipcode

c) Save your response mapping changes, click **File > Save** (Ctrl-S). Close the field mapping editor.

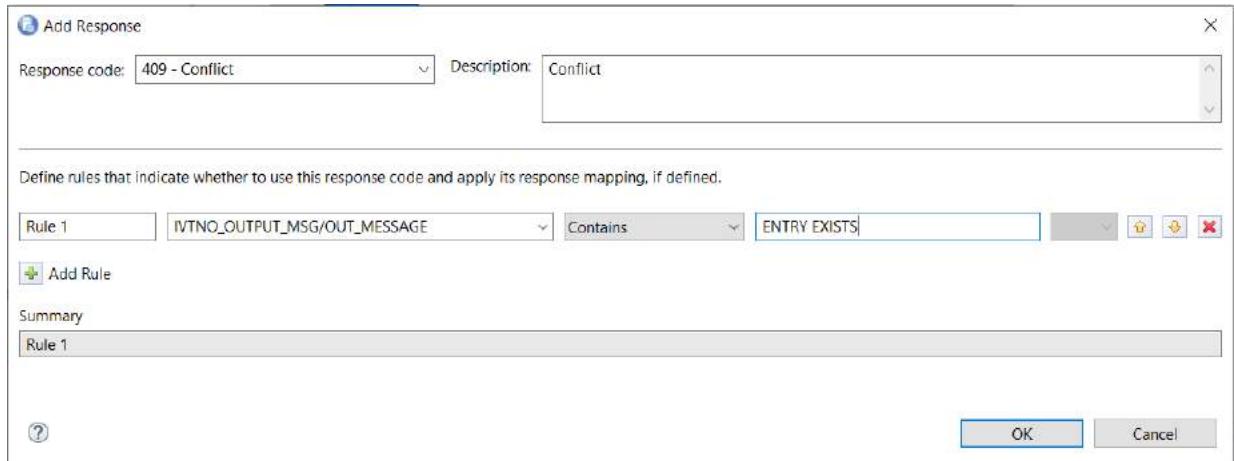
5. In the API editor, under the **Responses** section for the **POST** method, click the **Add Response** button.

The Add Response window opens.

6. Specify a response code of **409**, and specify the following rule:

a) For Rule 1, specify **IVTNO_OUTPUT_MSG/OUT_MESSAGE** as the service field. Select **Contains** for the comparison operator. Enter **ENTRY EXISTS** for the comparison value.

b) Click **OK**.



7. Ensure response code 200 is set as the default response code for the POST method if it is not already set.
 - a) Click the **Response** button for the 200 response code.
 - b) Click **Set Response 200 As Default**
8. Save your changes to your API project by Clicking **File > Save** (Ctrl-S).

Results

You have configured multiple response codes for your phone book API GET and POST methods, your GET method has also been configured with a unique response mapping. Your API is now ready to deploy.

Method	Operation ID	Responses
POST	postPhonebookService	409 Conflict 200 OK
GET	getPhonebookService	404 Not Found 200 OK

Deploy the contacts API

Deploy the contacts API directly from z/OS Connect EE API toolkit.

Before you begin

A server connection is required to deploy the API.

Procedure

1. In the **Project Explorer** view, right-click your API project and select **z/OS Connect EE > Deploy API to z/OS Connect EE Server**.
2. In the **Deploy API** window, select the server to which to deploy the API.
3. Click **OK**.

Deployment result is reported. The deployed API is automatically started unless specified otherwise in the z/OS Connect EE preferences window.

Results

Your contacts API is deployed and ready to be tested.

Test the IMS contacts API

You can test your newly created contacts API is working by invoking the API directly from z/OS Connect EE API toolkit by using the "Try it out!" function in the Swagger UI that is embedded in the editor.

Procedure

To test your newly created and deployed contacts API:

1. Start IMS and ensure that IMS Connect is listening on the expected port.
2. Start your z/OS Connect EE server. For more information, see [“Starting and stopping z/OS Connect EE” on page 471](#).
 - a) Check the messages.log file for the following message to confirm the service is installed.

BAQR7043I: z/OS Connect EE service archive phonebook installed successfully.

- b) Check the messages.log file for the following message to confirm the API is installed.

BAQR7000I: z/OS Connect API archive file contacts installed successfully.

3. In the **z/OS Connect EE Servers** view of the API toolkit, under the **APIs** folder, double-click on the contacts API. Click **OK** on the Try it out! dialog.

The API is opened in the Swagger UI.

4. Click **List Operations** to see available operations in the API.

contacts

This is an API for managing contacts.

The screenshot shows the 'contacts' API endpoint in the z/OS Connect EE API toolkit. At the top, there's a header with 'default' and buttons for 'Show/Hide', 'List Operations' (which has a red arrow pointing to it), and 'Expand Operations'. Below this, there are four operation entries: a green 'POST' button for '/contacts', a red 'DELETE' button for '/contacts/{lastName}', a blue 'GET' button for '/contacts/{lastName}', and an orange 'PUT' button for '/contacts/{lastName}'. At the bottom, a note says '[BASE URL: /phonebook , API VERSION: 1.0.0]'.

Figure 41. Operations in the API

5. Test adding a contact record by clicking **POST**.
 - a) Click the **Example Value** box to copy the request message format into your contacts POST request.
 - b) Specify the last name, first name, zip code, and extension.

The screenshot shows the z/OS Connect API Explorer interface. At the top, there's a header with the text "Parameters". Below it is a table with three columns: "Parameter", "Value", and "Description". A single row is visible, labeled "postPhonebookService_request" under "Parameter" and "request body" under "Description". The "Value" column contains a dropdown menu with the option "IVTNO_INPUT_MSG" selected. A modal window titled "IVTNO_INPUT_MSG" is displayed, showing a list of input fields with their values: IN_TRANCODE (IN_TRANCODE), IVTNO (IVTNO), IN_LAST_NAME (Smith), IN_FIRST_NAME (John), IN_EXTENSION (000-0000), IN_ZIP_CODE (00000). Below the modal, the "Parameter content type" is set to "application/json". At the bottom of the main area, there are buttons for "Try it out!" and "Hide Response".

Figure 42. Testing the POST request

- c) Click **Try it out!**.

Information about the request URL, request headers, response body, response code, and response headers are provided. The response body contains the output message. The OUT_MESSAGE would contain one of the following messages:

- ENTRY WAS ADDED
- ADDITION OF ENTRY HAS FAILED (this message indicates that the IN_LAST_NAME value you specified already exists)

Results

You have verified your newly created phonebook service is working.

Create an API to invoke the IBM MQ stock query service

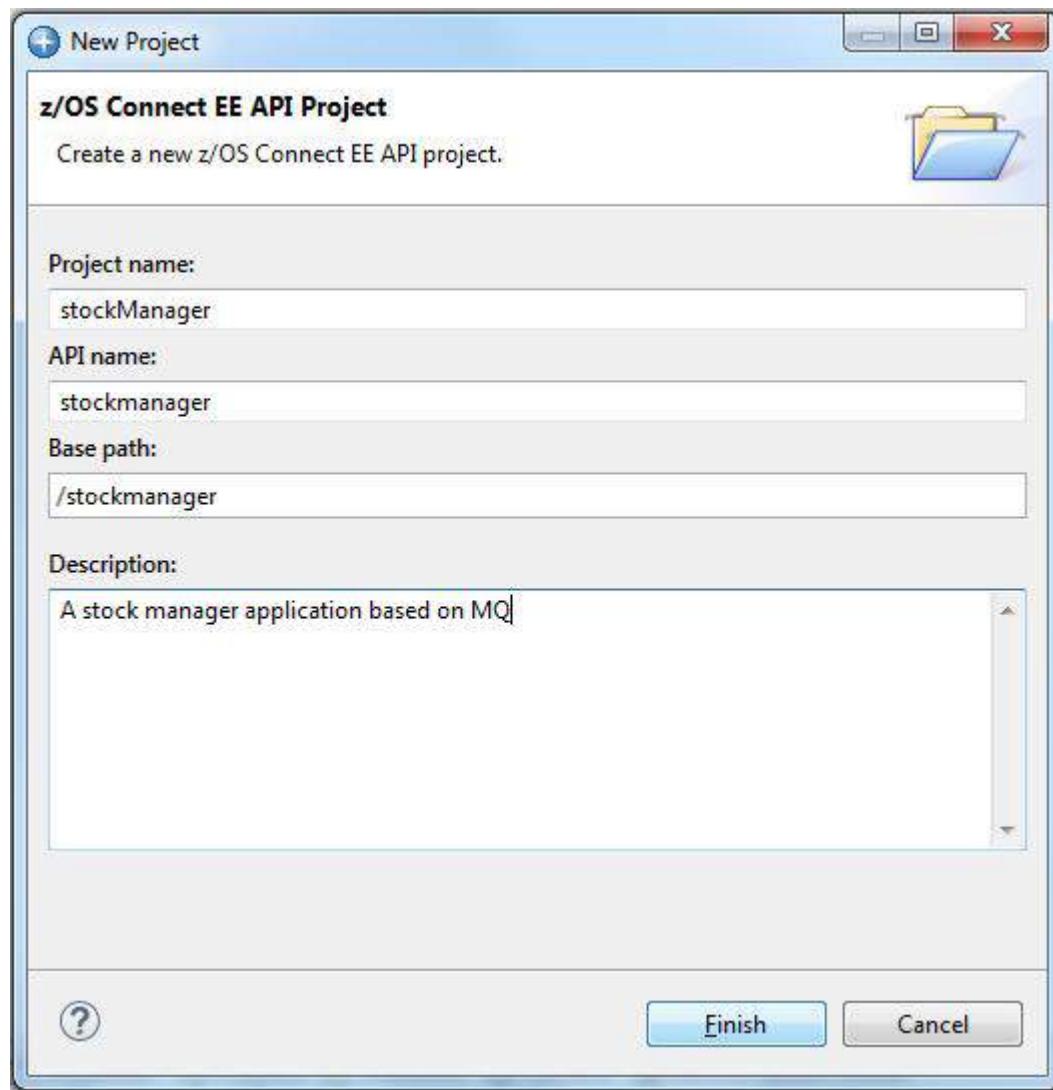
Create a stock manager API that calls the IBM MQ stock query application.

Before you begin

Use the z/OS Connect EE API toolkit to Stop the stockmanager API, and then Remove the stockmanager API.

Procedure

1. Start IBM® Explorer for z/OS and open the z/OS Connect Enterprise Edition perspective.
2. From the main menu, click **File > New > z/OS Connect EE API Project**



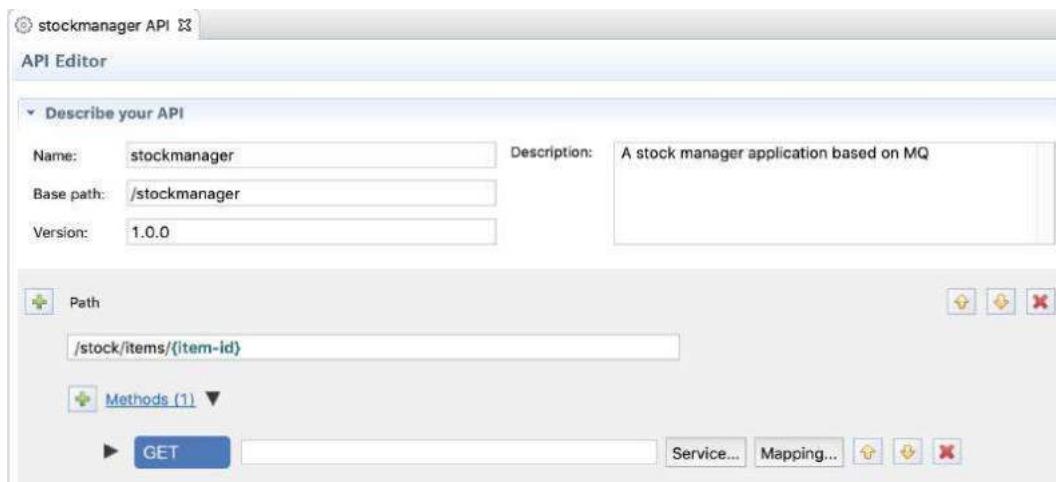
3. Complete the sections for the new API project and click **Finish**.

The z/OS Connect EE API editor opens.

The API editor interface shows the configuration for the 'stockmanager' API. It includes fields for Name ('stockmanager'), Description ('A stock manager application based on MQ'), Base path ('/stockmanager'), and Version ('1.0.0'). Below this, there are sections for 'Path' (containing '/newPath1') and 'Methods' (listing POST, GET, PUT, and DELETE). Each method row has 'Service...', 'Mapping...', and navigation buttons.

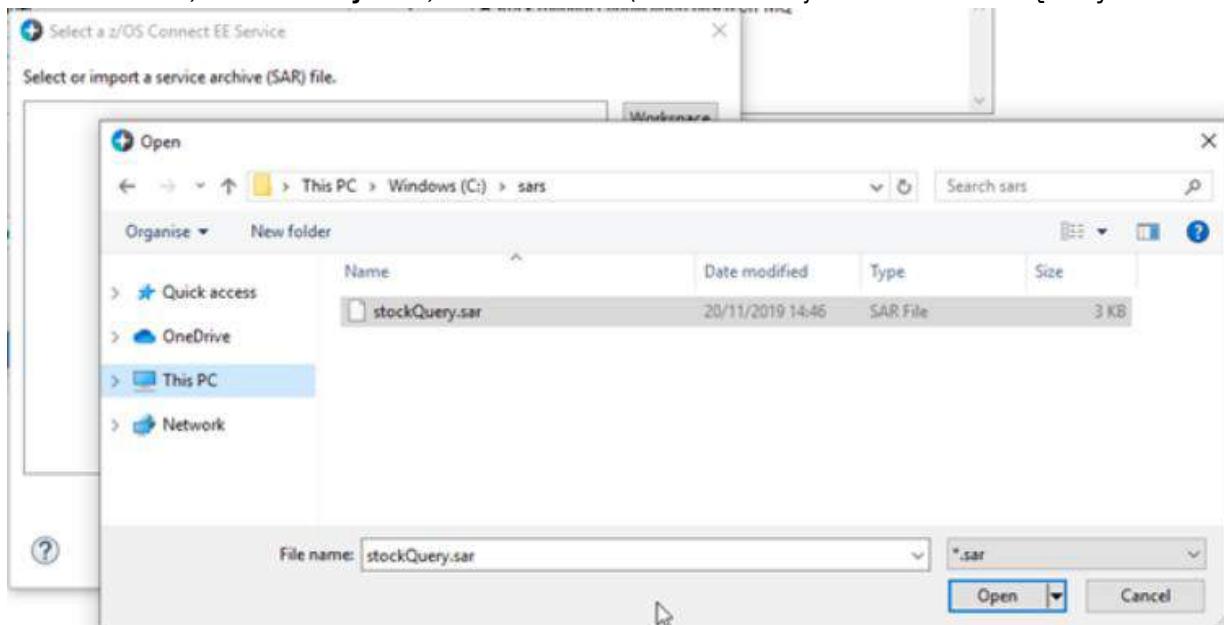
4. Change the value of the path attribute to `/stock/items/{item-id}`, where `{item-id}` indicates a path parameter whose value is substituted at runtime.

For the moment, you use the HTTP GET method of the service only to make stock queries. Delete the other three methods and save the API.



5. Import the service archive file that you created in the task “Create a two-way IBM MQ service” on page 96.

Click **Service...**, select **File System**, and find the C:\sars directory and select stockQuery.sar.

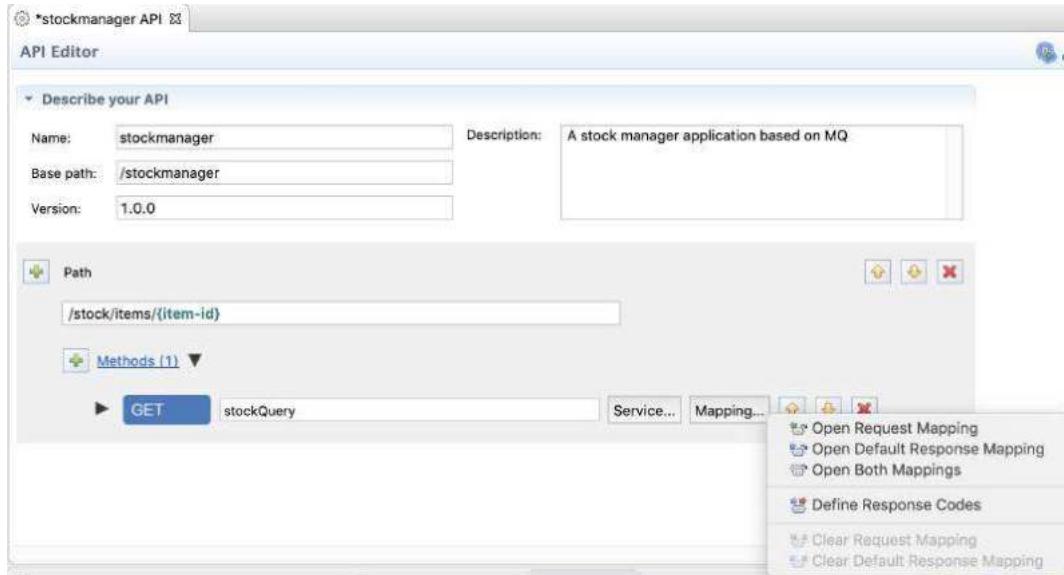


6. Click **Open**, **OK**, **OK**, and **Save**.

The HTTP GET method is now associated with the stockQuery service.

7. Click **Mapping...** and select **Open Both Mappings**.

The **Service Interface Mapping** editor opens.



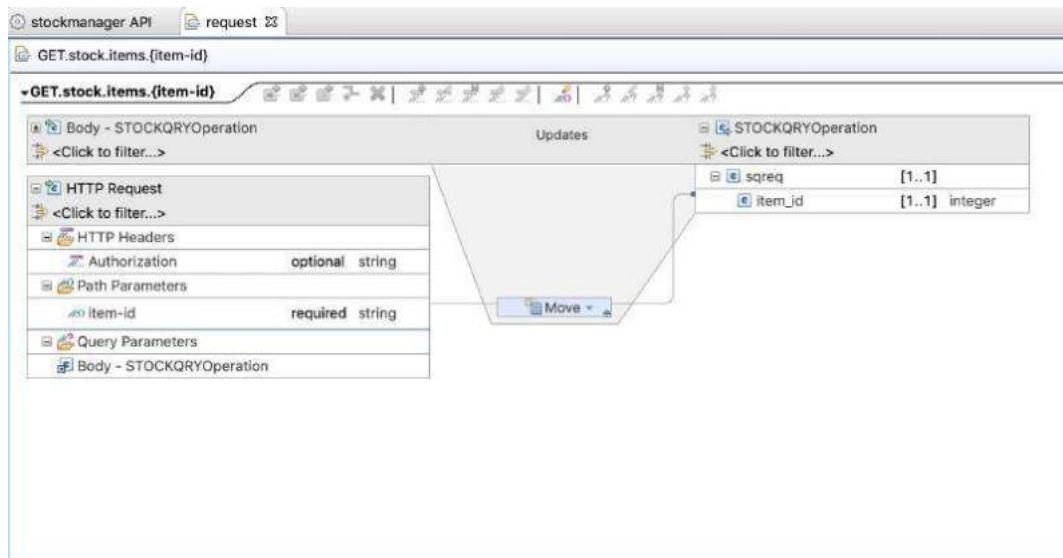
8. Explore the contents of the **Response Mapping** editor.

For this scenario, use the default mapping, which passes the response without modification from the service back to the caller of the API. When you finish exploring, close the **Response Mapping** editor.

9. Use the **Request editor** to map the item-id path parameter that you created in step “4” on page 151 to the item-id in the payload.

This mapping creates a more natural REST API where the caller of the API will not need to pass any payload in the HTTP GET request.

- Left-click and hold on the item-id path parameter, then drag to the item_id integer on the right.



- Save your changes.

Results

You have created a stock manager API.

What to do next

[“Deploy the stock manager API ” on page 153](#)

Deploy the stock manager API

Deploy the stock manager API directly from z/OS Connect EE API toolkit.

Before you begin

A server connection is required to deploy the API.

Procedure

1. In the **Project Explorer** view, right-click your API project and select **z/OS Connect EE > Deploy API to z/OS Connect EE Server**.
2. In the **Deploy API** window, select the server to which to deploy the API.
3. Click **OK**.

Deployment result is reported. The deployed API is automatically started unless specified otherwise in the z/OS Connect EE preferences window.

Results

Your stock manager API is deployed and ready to be tested.

What to do next

[“Test the stock manager API” on page 153](#)

Test the stock manager API

Test your newly created API by invoking it directly from the z/OS Connect EE API toolkit.

Use the **Try it out!** function in the Swagger UI that is embedded in the editor.

Procedure

1. Start your IBM MQ queue manager.
2. Start the stock query application, see [“Prepare the sample IBM MQ stock query application” on page 63](#).
3. Start your z/OS Connect EE server. For more information, see [“Starting and stopping z/OS Connect EE” on page 471](#).
 - a) Check the messages.log file for the following message to confirm that the service is installed.
BAQR7043I: z/OS Connect EE service archive stockQuery installed successfully.
 - b) Check the messages.log file for the following message to confirm that the API is installed.
BAQR7000I: z/OS Connect API archive file stockmanager installed successfully.
4. In the **z/OS Connect EE Servers** view of the API toolkit, under the **APIs** folder, double-click the **stockmanager** API.
The API is opened in the Swagger UI.
5. Click **List Operations** to see available operations in the API.

stockmanager

A stock manager application based on MQ

stockmanager

Show/Hide | List Operations | Expand Operations

GET /stock/items/{item-id}

[BASE URL: /stockmanager , API VERSION: 1.0.0]

Figure 43. Operations in the API

6. Query the state of the stock with item ID 2033.
- Specify a value of 2033 for the item-id and click **GET**.

The screenshot shows a REST API testing interface. At the top, there is a blue button labeled "GET" and a URL field containing "/stock/items/{item-id}". Below the URL, the "Response Class (Status 200)" is listed as "OK". There are two tabs: "Model" and "Example Value", with "Example Value" selected. The example value is a JSON object:

```
{
  "SQRESP": {
    "ITEM_ID": 0,
    "ITEM_DESC": "string",
    "ITEM_COUNT": 0,
    "ITEM_COST": 0
  }
}
```

Below the example value, the "Response Content Type" is set to "application/json". Under "Parameters", there are two entries:

Parameter	Value	Description	Parameter Type	Data Type
Authorization	<input type="text"/>		header	string
item-id	<input type="text" value="2033"/>		path	string

A "Try it out!" button is located at the bottom left of the parameters section.

Figure 44. Testing the GET request

- Click **Try it out!**.

Information about the request URL, request headers, response body, response code, and response headers are provided.

An HTTP response code of 200 is expected with the following response body:

```
{
  "SQRESP": {
    "ITEM_ID": 2033,
    "ITEM_DESC": "A description.",
    "ITEM_COST": 4,
    "ITEM_COUNT": 500
  }
}
```

Results

You verified that your newly created stockmanager API has successfully called the stock query application by using the stock query service running in z/OS Connect EE.

Create a Db2 API to invoke the Db2 employee services

Use the API toolkit to create an API to invoke the Db2 employee services

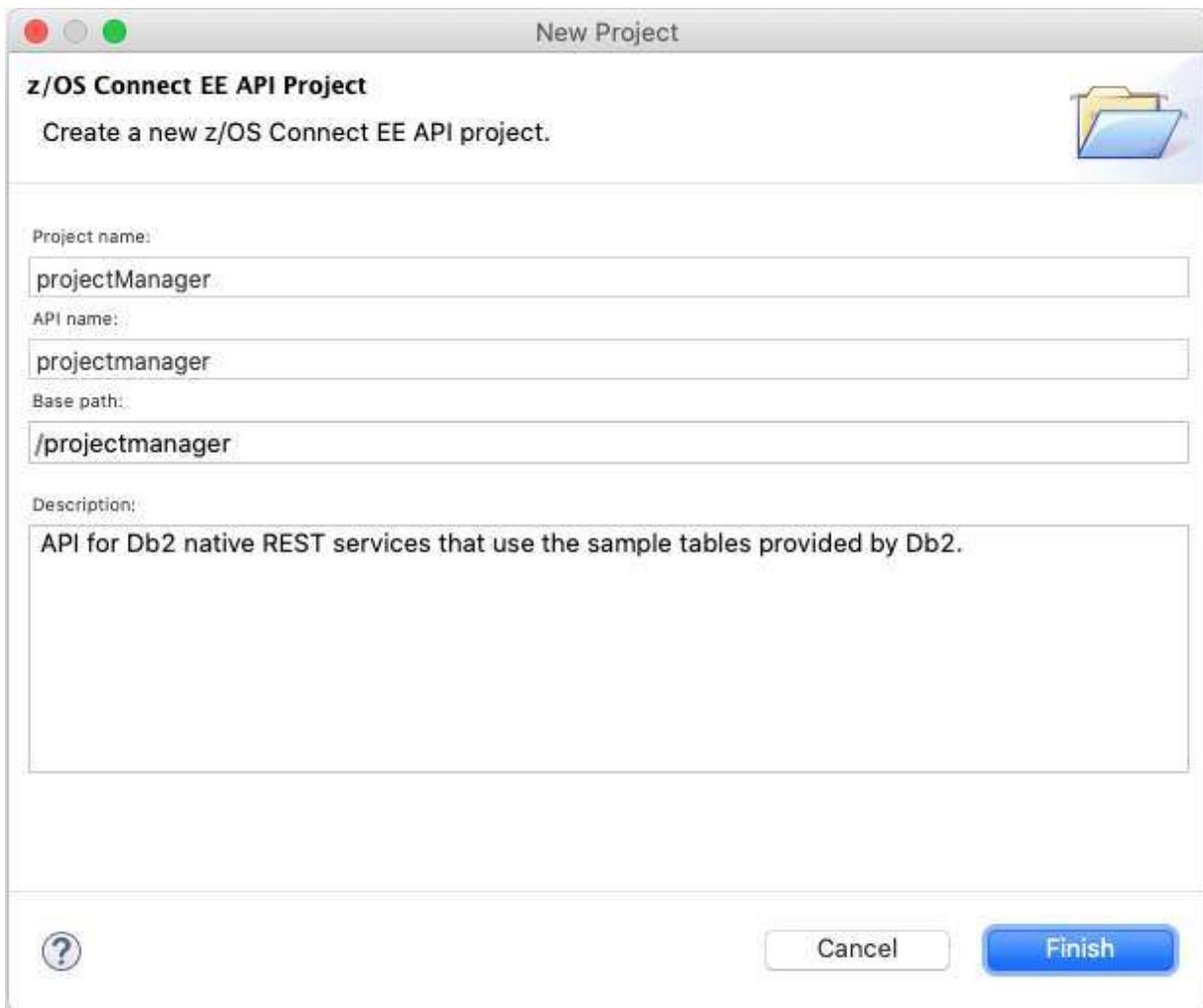
Before you begin

Ensure that the following tasks are complete:

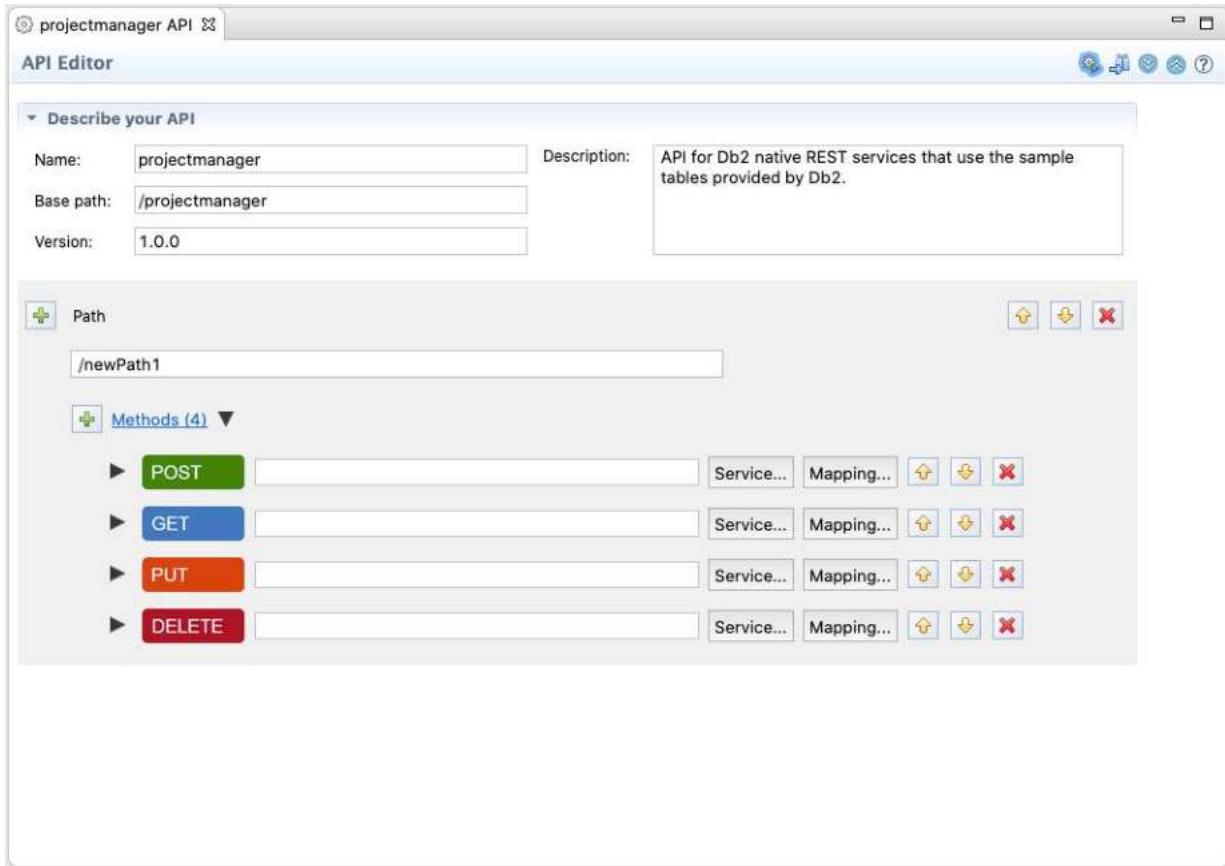
- ["Enable and create Db2 native REST services" on page 66](#)
- ["Create a server to connect to Db2" on page 80](#) This task installs relevant artifacts defined in the supplied configuration template sampleDb2ProjectManager, including the projectManager API.
- Use the z/OS Connect EE API toolkit to stop and then remove the projectManager API.

Procedure

1. Start IBM Explorer for z/OS and open the z/OS Connect EE perspective.
2. From the menu bar, select **File > New > z/OS Connect EE API project**.
3. Enter `projectManager` for the project name and enter API for Db2 native REST services that use the sample tables provided by Db2. for the description. Click **Finish**.



The z/OS Connect EE API Editor opens.



4. Select the **projectManager API** tab and rename the default base path to /projectManager
5. If you did not complete the task “Create a Db2 service” on page 108, the employeeList.sar is located in the <WLP_USER_DIR>/servers/db2ProjectManager/resources/zosconnect/services directory of the server. The employeeDetails.sar and employeeUpdate.sar are located in the same directory. The service archive files must be transferred in binary mode to the workstation where the IBM Explorer for z/OS is installed.

Create an operation to list employees

Create an operation to list employees

Before you begin

If the employeeList.sar file was generated on a different workstation, transfer the file, in binary mode, to the workstation where the API toolkit is installed.

About this task

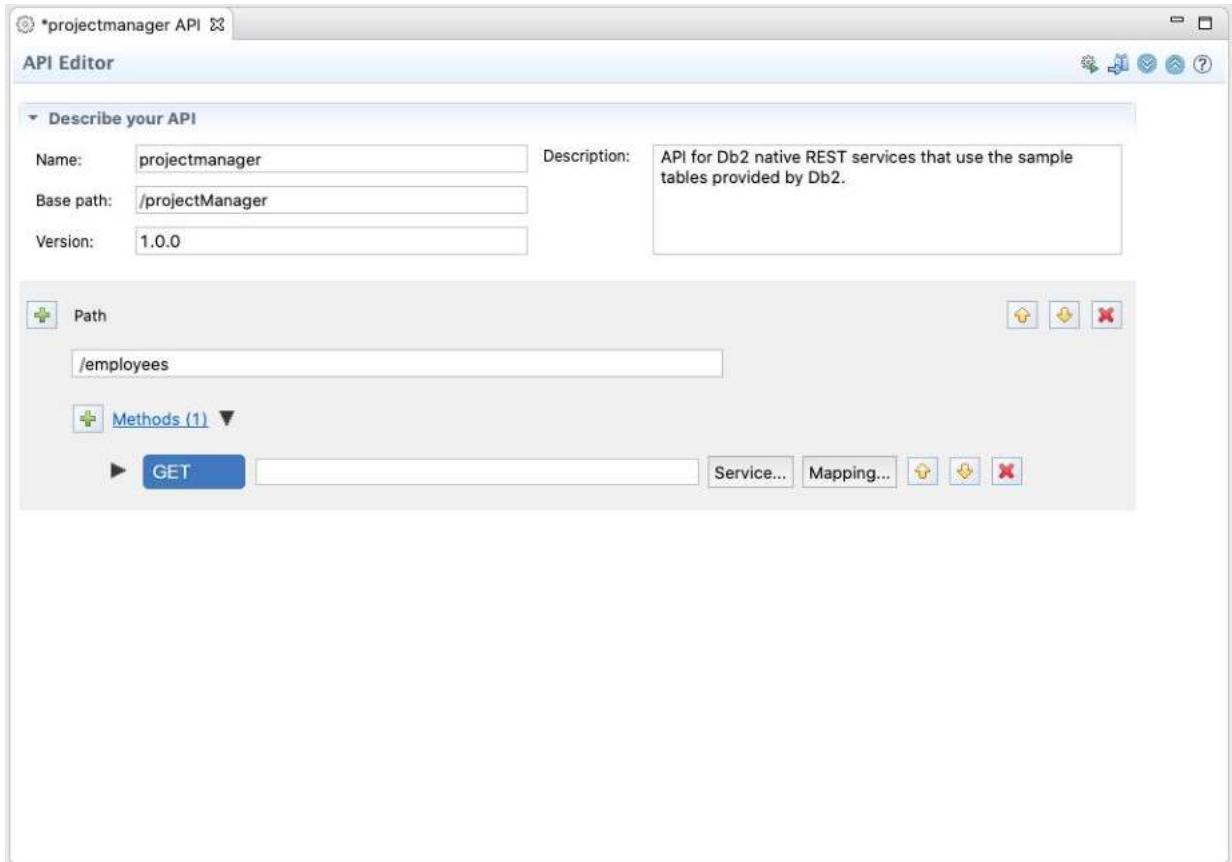
This operation is implemented as an HTTP GET request and shows you how to expose a resource for the employeeList service.

Procedure

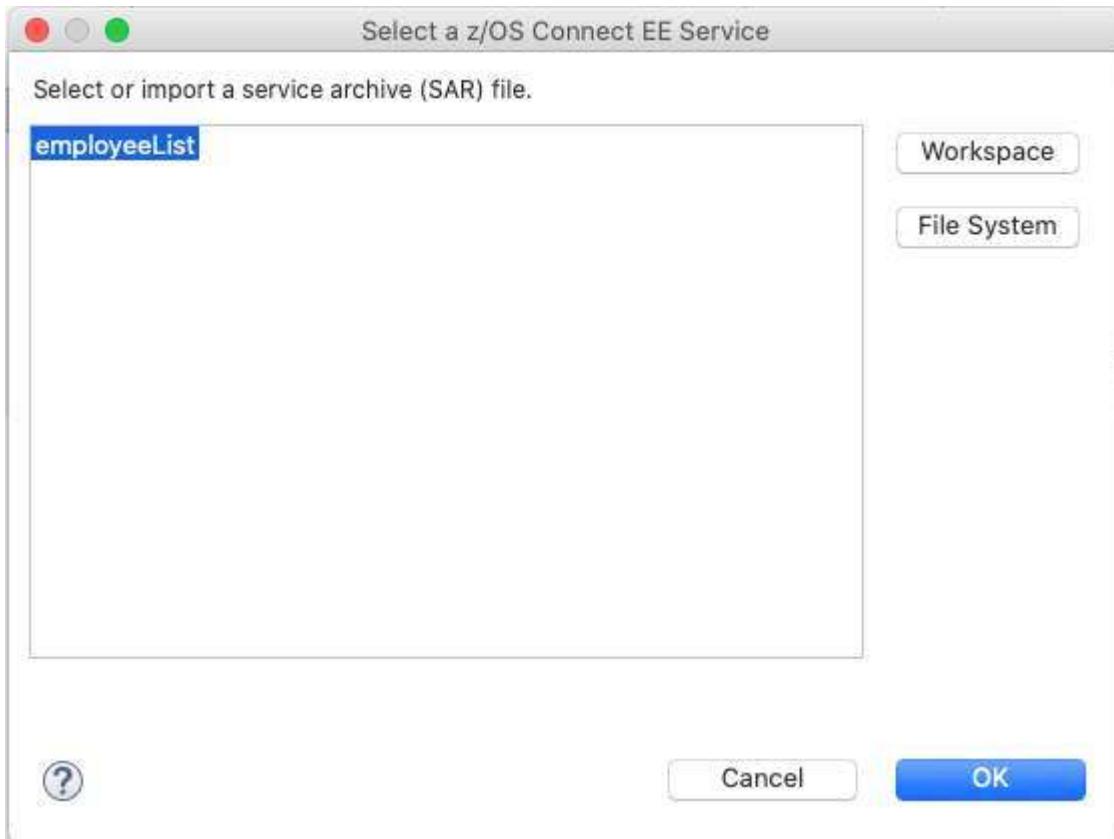
1. Rename the default **Path** value to be /employees by typing over the existing value.



2. Remove the POST, PUT and DELETE methods by clicking on the  at the end of the line. This should leave only the GET method allowed for the path /employees.



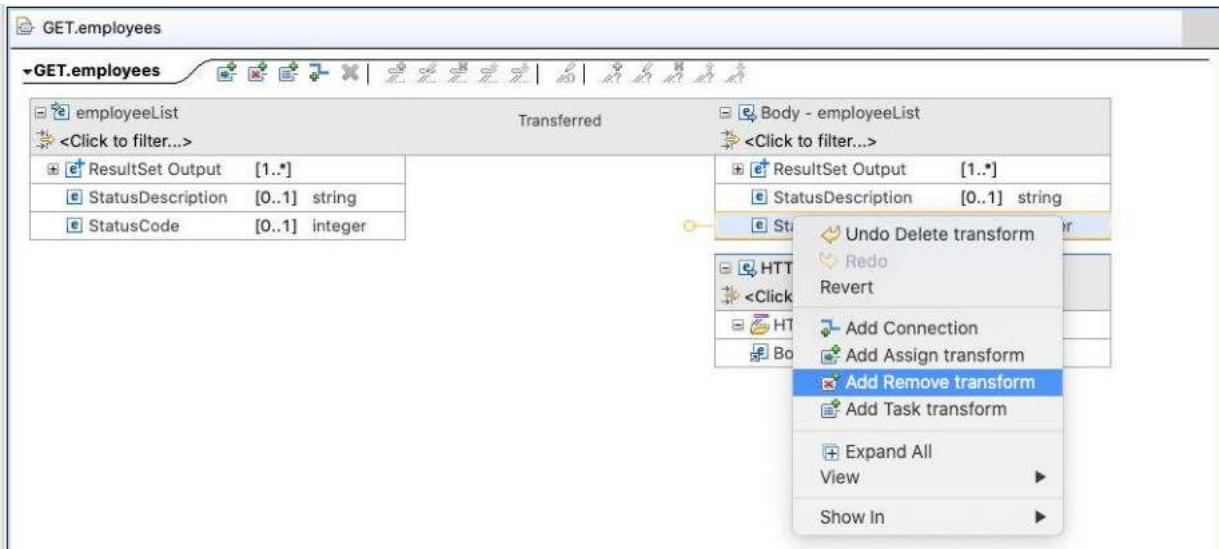
3. Click **Service...** for the GET method to import the `employeeList.sar` file.
The **Select a z/OS Connect EE Service** dialog opens.
4. Click **Workspace** or **File System**, depending on where your `employeeList.sar` file is located, and select it to import. Click **Open**.
The **Import Services** dialog opens. Click **OK**. The `employeeList` service now appears in the Service dialog.



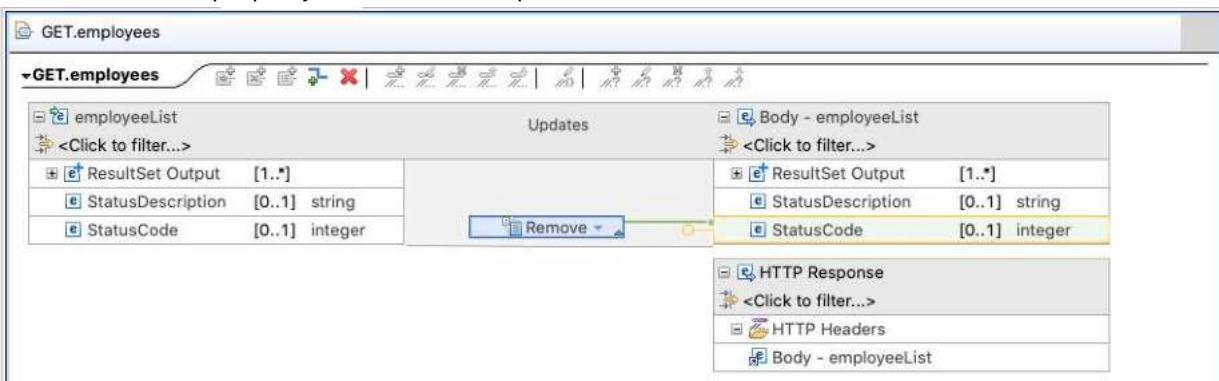
Click **OK**. The operation now maps to the employeeList service.



5. From the menu bar, select **File > Save**.
6. Click **Mapping...** for the GET method that uses the employeeList service. Click **Open Default Response Mapping**. The response mapping editor opens, so that you can define the mappings between the JSON content returned from the z/OS Connect EE service and the response the API returns.
7. In the response mapping, create a Remove transform for the JSON property `StatusCode` by right-clicking on the JSON property and selecting **Add Remove transform**.



This removes the property from the API response.



8. Save and close the response mapping file.

Create an operation to get details of a single employee

Create an operation to get information about a single employee

Before you begin

Transfer the employeeDetails.sar file, in binary mode, to the workstation where the API toolkit is installed.

About this task

This operation is implemented as an HTTP GET request and shows you how to use the employeeDetails service to expose an individual employee's details.

Procedure



1. Select the **projectManager API** tab and click next to **Path** to add a new path.
2. Enter /employees/{employeeNumber} into the path and delete the POST and DELETE methods.
3. Click **Service...** for the GET method to select the service archive file that defines the service on this API path that is called by an HTTP GET request.
The **Select a z/OS Connect EE Service** dialog opens.
4. Click **Workspace** or **File System**, depending on where your employeeDetails.sar file is located, and select it to import it. Click **Open**.
The **Import Services** dialog opens.

5. Click **OK**.

The operation now maps to the employeeDetails service.



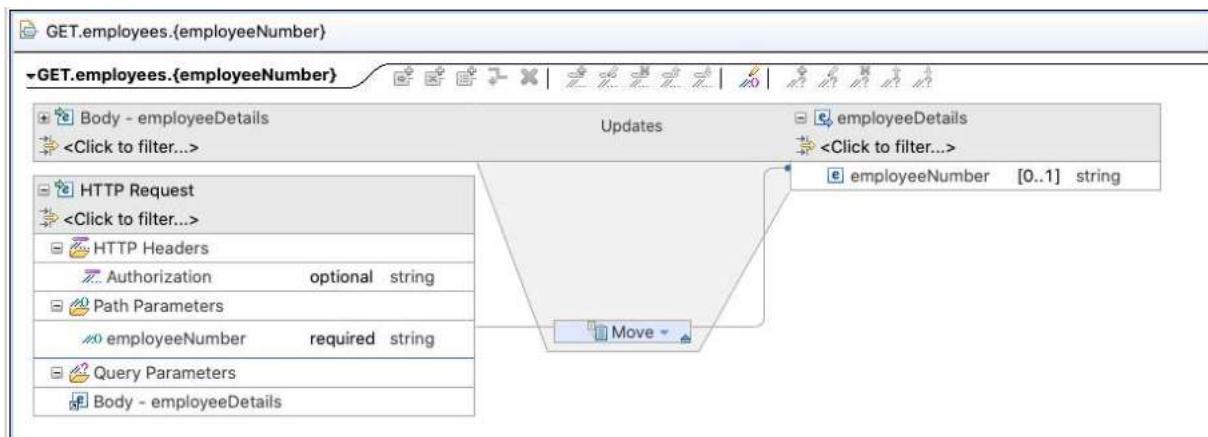
6. From the menu bar, select **File > Save**.

7. Click **Mapping...** for the GET method that uses the employeeDetails service.

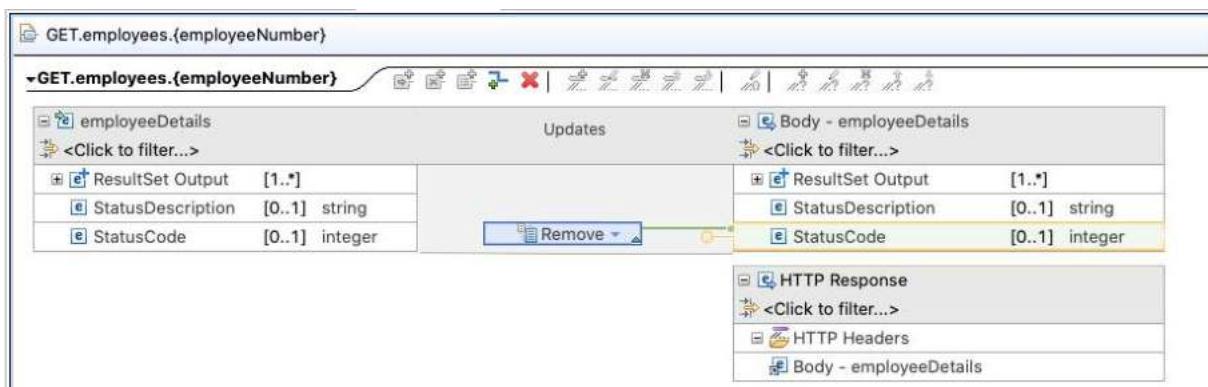
8. Click **Open Both Mappings**.

The request and response mapping editors open so that you can define the mappings between the content of the API's HTTP request and response and the JSON content passed to and from the z/OS Connect EE service.

9. In the request mapping, create a Move transform that copies the value from the path parameter in the HTTP request to the property in the JSON payload passed to the z/OS Connect EE service, by dragging from the path parameter employeeNumber to the JSON property employeeNumber.



10. In the response mapping, create a Remove transform for the JSON property StatusCode by right-clicking on the JSON property and selecting **Add Remove transform**. This removes the property from the API response.



11. Save and close the request and response mapping files.

Create an operation to update an employee's details

Create an operation to update an employee's details

Before you begin

Transfer the `employeeUpdate.sar` file, in binary mode, to the workstation where the API toolkit is installed.

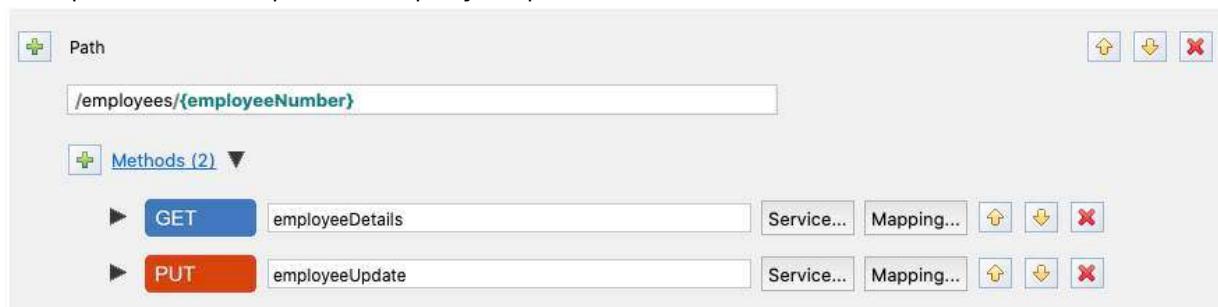
About this task

This operation is implemented as an HTTP GET request and shows you how to use the `employeeUpdate` service to update an individual employee's details.

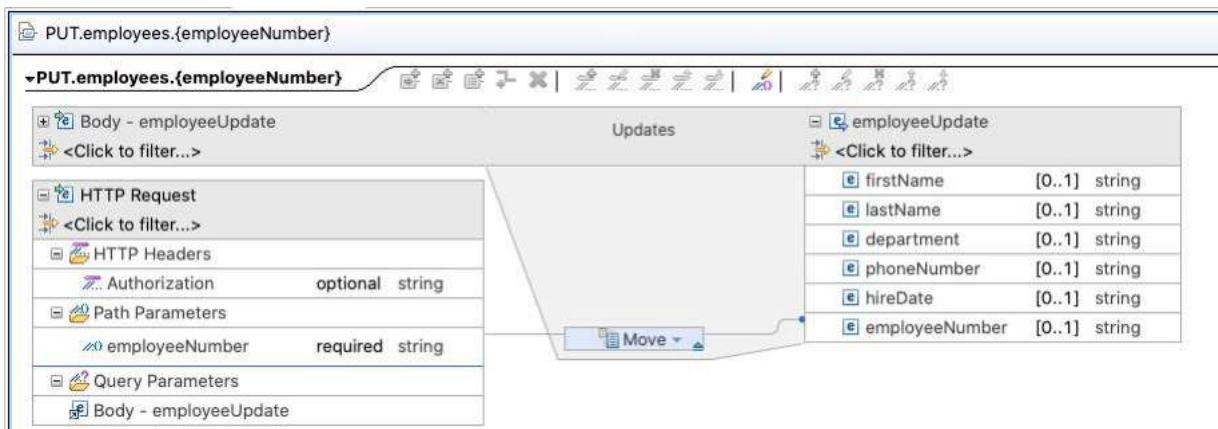
Procedure

1. Select the **projectManager API** tab and click **Service...** for the PUT method to select the service archive file that defines the service for this API. The method is called by an HTTP PUT request.
The **Select a z/OS Connect EE Service** dialog opens.
2. Click **Workspace** or **File System**, depending on where your `employeeUpdate.sar` file is located, and select it to import. Click **Open**.
3. The **Import Services** dialog opens. Click **OK**.

The operation now maps to the `employeeUpdate` service.

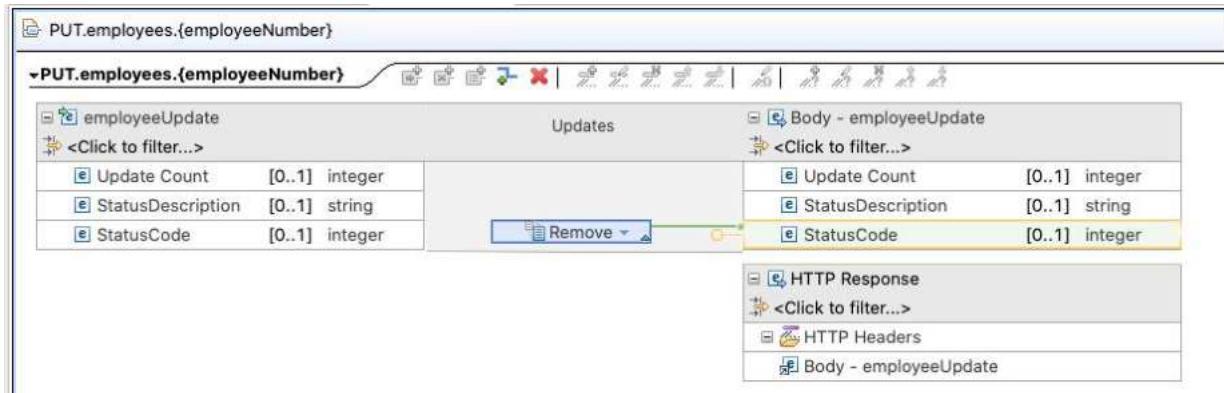


4. From the menu bar, select **File > Save**.
5. Click **Mapping...** for the PUT method that uses the `employeeUpdate` service.
6. Click **Open Both Mappings**.
The request and response mapping editors open so that you can define the mappings between the content of the APIs HTTP request and response, and the JSON content passed to and from the z/OS Connect EE service.
7. In the request mapping, create a Move transform that copies the value from the path parameter in the HTTP request to the property in the JSON payload passed to the z/OS Connect EE service, by dragging from the path parameter `employeeNumber` to the JSON property `employeeNumber`.



8. In the response mapping, create a Remove transform for the JSON property **StatusCode** by right-clicking the JSON property and selecting **Add Remove transform**.

This removes the property from the API response.



9. Save and close the request and response mapping files.

Define a response code mapping for the Db2 projectManager API

Learn how to define a response code mapping for a Db2 projectManager API method. The steps in this task are optional.

Before you begin

Ensure your API project is open in the z/OS Connect EE API editor. You can do this by double-clicking the package.xml file for your Db2 projectManager API project in the **Project Explorer** view.

About this task

In this step, you will define response codes with unique response data mapping for the PUT /employees/{employeeNumber} API operation.

Procedure

1. In the API editor, for the **PUT** method, click **Mapping... > Define Response Codes** to open the response details for the PUT API method.

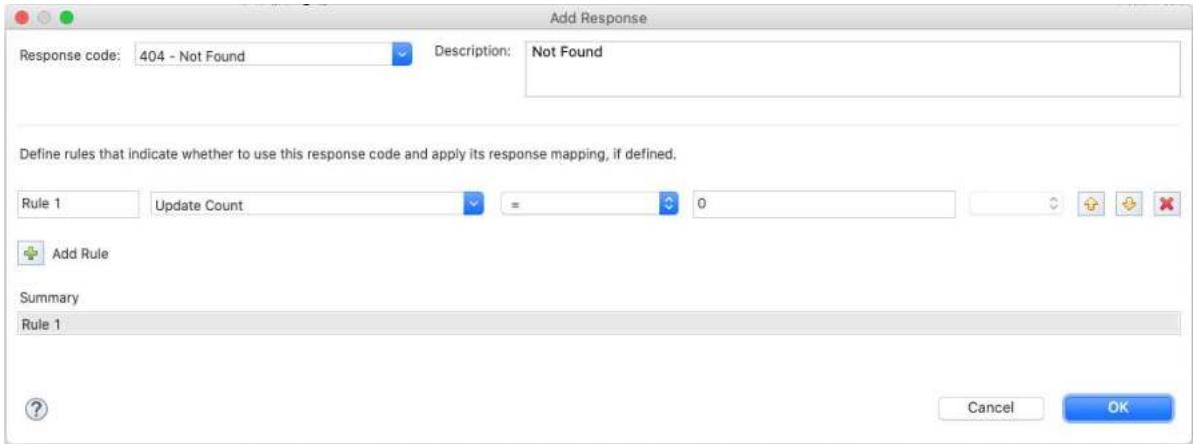
2. Click next to **Add Response**

The **Add Response** window opens.

3. Specify a response code of **404**, and define the following rule:

a) For Rule 1, specify **Update Count** as the service property. Select **=** for the comparison operator, and enter **0** for the comparison value.

b) Click **OK** to save your changes.

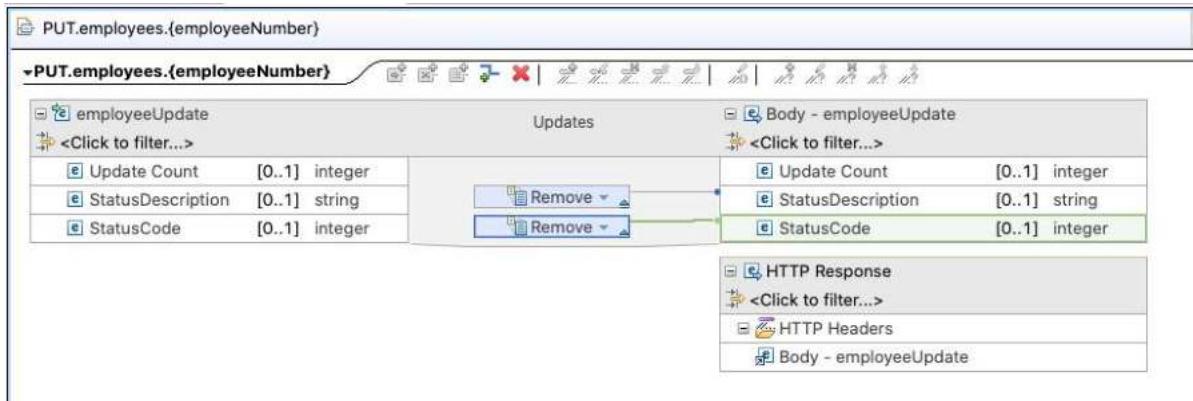


4. In the API editor for PUT 404 response, click **Mapping...** > **Open Response 404 Mapping**.

The **Response Mapping Editor** window opens.

- Add a remove transform for the `StatusDescription` and `StatusCode` JSON properties. For each property, select the property on the right hand side and then right-click and select **Add Remove transform**.

The properties are removed from the API response.



- Save the response mapping.

5. Click **File > Save** to save the changes to your API project.

Results

You have configured multiple response codes for your Db2 projectManager API PUT method. The unique mapping means that if a user calls the API and the employee number in the request is not found, a 404 response is returned.

What to do next

Your API is now ready to deploy. See [“Deploy the projectManager API with the API toolkit” on page 163](#)

Deploy the projectManager API with the API toolkit

Use the API toolkit to deploy the projectManager API.

Procedure

- In the **Project Explorer** view, right click `projectManager`, then click **z/OS Connect EE > Deploy API to z/OS Connect EE Server**.
- In the **Deploy API** dialog, select the z/OS Connect EE connection from the drop-down icon to connect.
- Click **OK**.

The **Deploy API to z/OS Connect EE Server Result** dialog shows the results of the deployment.

4. Click **OK** to close the dialog.

Use the Swagger UI to test the projectManager API

Test your newly created projectManager API by using the **Try it out!** function in the Swagger UI that is embedded in the editor.

Before you begin

The following tasks must be completed:

- “[Create a Db2 API to invoke the Db2 employee services](#)” on page 154
- “[Deploy the projectManager API with the API toolkit](#)” on page 163

Procedure

1. Ensure your Db2 instance which has the native REST services installed is running.
2. Ensure your z/OS Connect EE server is running.

For more information, see “[Starting and stopping z/OS Connect EE](#)” on page 471. Use the **z/OS Connect EE Servers** view to confirm your APIs and services are installed.



- a) View installed services.

Expand the z/OS Connect EE server and the **Services** subfolder and check the employeeList, employeeDetails, and employeeUpdate services are listed

- b) View installed APIs.

Expand the **APIs** subfolder and check the projectManager API is listed

3. In the **z/OS Connect EE Servers** view of the API toolkit, under the APIs folder, double-click on the projectManager API.

The API opens in the Swagger UI.

4. Click **List Operations** to see available operations in the API.

projectmanager

Show/Hide | List Operations | Expand Operations

GET	/employees
GET	/employees/{employeeNumber}
PUT	/employees/{employeeNumber}

[BASE URL: /projectManager , API VERSION: 1.0.0]

5. Click **GET /employees** to test listing employees.

Click **Try it out!**. Information about the request URL, request headers, response body, response code, and response headers are provided.

GET /employees

The response body contains an array of employees.

Response Body

```
{  
    "StatusDescription": "Execution Successful",  
    "ResultSet Output": [  
        {  
            "firstName": "BRUCE",  
            "lastName": "ADAMSON",  
            "department": "D11",  
            "employeeNumber": "000150"  
        },  
        {  
            "firstName": "ROY",  
            "lastName": "ALONZO",  
            "department": "E21",  
            "employeeNumber": "200340"  
        },  
        {  
            "firstName": "DAVID",  
            "lastName": "BROWN",  
            "department": "D11",  
            "employeeNumber": "000200"  
        }  
    ]  
}
```

Response Code

200

6. Click **GET /employees/{employeeNumber}** to get the details of a single employee.

Enter an employeeNumber, as found in the response from the previous request, in the **employeeNumber** path parameter. Click **Try it out!**. Information about the request URL, request headers, response body, response code, and response headers are provided.

GET /employees/{employeeNumber}

The response body contains details for a single employee.

Response Body

```
{  
    "StatusDescription": "Execution Successful",  
    "ResultSet Output": [  
        {  
            "firstName": "BRUCE",  
            "lastName": "ADAMSON",  
            "hireDate": "1972-02-12",  
            "phoneNumber": "4510",  
            "department": "D11",  
            "employeeNumber": "000150"  
        }  
    ]  
}
```

7. Click **PUT /employees/{employeeNumber}** to update the details for a single employee.

Enter the same employeeNumber, firstName, lastName, hireDate, and department as used in the previous request, in the matching parameters. Enter a different 4 digit number in the **employeePhoneNumber** request body. Click **Try it out!**. Information about the request URL, request headers, response body, response code, and response headers are provided.

Parameters

Parameter	Value	Description	Parameter Type	Data Type
Authorization			header	string
employeeNumber	000150		path	string
putEmployeeUpdate_request	<input type="button" value=""/>	Service employeeUpdate invocation HTTP request body	request body	body

Model | Example Value

```
{
  "firstName": "string",
  "lastName": "string",
  "department": "string",
  "phoneNumber": "string",
  "hireDate": "string"
}
```

Parameter content type: application/json

The response body contains details for a single employee.

Response Body

```
{
  "StatusDescription": "Execution Successful",
  "Update Count": 1
}
```

- To confirm the update was successful, click **GET /employees/{employeeNumber}** to get the details of the same employee.

Enter the same employeeNumber, as used in the previous request, in the **employeeNumber** path parameter. Click **Try it out!**. Information about the request URL, request headers, response body, response code, and response headers are provided.

GET /employees/{employeeNumber}

The response body contains details for a single employee.

Response Body

```
{
  "StatusDescription": "Execution Successful",
  "ResultSet Output": [
    {
      "firstName": "BRUCE",
      "lastName": "ADAMSON",
      "hireDate": "1972-02-12",
      "phoneNumber": "4510",
      "department": "D11",
      "employeeNumber": "000150"
    }
  ]
}
```

Results

You have verified that your newly created projectManager API works to Db2.

Build service archives and API archives for DevOps

In a DevOps environment, service projects and API projects that are created in the API toolkit are treated as source files and must be stored in a source control management (SCM) system. When the source files are updated, a script or an automated process is triggered to build the service archives and API archives from the source files in the SCM by using the build toolkit **zconbt** command.

About this task

Deploying services and APIs directly from the API toolkit allows you to develop and test in the development environment. However, you need a way to get the services and APIs promoted to production. You can automate the building, deployment, and promotion of services and APIs to production by using the build toolkit. The build toolkit understands the format of the service projects and API projects, and it can build the source project into the service archive or API archive that is required by the z/OS Connect EE server.

For more information about a general enterprise DevOps workflow and sample DevOps technologies for building the archives and deploying the services and APIs, see “[DevOps with z/OS Connect EE](#)” on page [14](#).

The following getting started tasks show you how to use the build toolkit **zconbt** command to build service archives and API archives so you can implement your DevOps solution.

Build a service archive with the build toolkit

This scenario shows you how to use the build toolkit to generate the service archive from a service project created in the API toolkit.

Before you begin

- Ensure that you complete the steps to create one of the following services in the API toolkit:
 - A CICS catalog manager service as described in “[Create a CICS service](#)” on page [83](#).
 - An IMS phonebook service as described in “[Create an IMS service](#)” on page [89](#).
 - An IBM MQ stock query service as described in “[Create a two-way IBM MQ service](#)” on page [96](#)
- Ensure that the z/OS Connect EE build toolkit is installed. For more information, see “[Installing the z/OS Connect EE build toolkit](#)” on page [199](#).

About this task

Use the build toolkit to generate the service archive from the service project that you created in the API toolkit. In an enterprise DevOps workflow, the build toolkit is likely run by a script in basically the same way as demonstrated in the following steps.

Procedure

In this procedure, substitute the Windows example for the platform of your choice. The build toolkit properties files must be encoded in UTF-8.

1. Create a local directory such as C:/DevOpsScenarios on the Windows system.
2. Download the build toolkit compressed file, zconbt.zip as a binary file, to the same local directory, C:/DevOpsScenarios.
The build toolkit is in the z/OS Connect EE <installation_path> and is called zconbt.zip.
3. Extract the zconbt.zip file into a zconbt sub directory of the C:/DevOpsScenarios local directory.
4. Add the **zconbt** executable file to your path. For example, enter the following command: **set PATH=C:/DevOpsScenarios/zconbt/bin;%PATH%**

5. Obtain the service project folder from your source control management system and place it under C:/DevOpsScenarios

For example, if your Eclipse workspace location is C:/Program Files/IBM/rationalsdp/workspace, you would find the InquireSingle/, phonebook/, or stockQuery/ folder there. Copy this folder into C:/DevOpsScenarios.

6. Run the build toolkit CLI to create the service archive file.

The build toolkit has the following syntax:

```
zconbt --projectDirectory=C:/DevOpsScenarios/<serviceProjectFolder> --file=C:/  
DevOpsScenarios/<serviceName>.sar
```

Use the **-pd** parameter to specify the project directory, and the **-f** parameter to specify the path and the file name of the generated service archive. In a command prompt window, depending on the name of your project, enter one of the following commands:

```
zconbt --projectDirectory=C:/DevOpsScenarios/InquireSingle --file=C:/DevOpsScenarios/  
inquireSingle.sar
```

or

```
zconbt --projectDirectory=C:/DevOpsScenarios/phonebook --file=C:/DevOpsScenarios/  
phonebook.sar
```

or

```
zconbt -projectDirectory=C:/DevOpsScenarios/stockQuery --file=C:/DevOpsScenarios/  
stockQuery.sar
```

Results

The service archive can be copied to the services directory of your server for deployment.

What to do next

For more information about a general enterprise DevOps workflow and sample DevOps technologies for building the archives and deploying the services and APIs, see [“DevOps with z/OS Connect EE” on page 14](#).

Related concepts

[“Automated service archive management” on page 670](#)

Deploy and undeploy your service archives automatically when you add or remove them from the services directory.

Related reference

[“zconbt command syntax” on page 795](#)

The **zconbt** command starts the build toolkit tool. You can use the build toolkit to generate archive files for services, APIs or API requesters.

Build an API archive with the build toolkit

This scenario shows you how to use the build toolkit to generate the API archive from an API project created in the API toolkit.

Before you begin

- Ensure that you complete the steps to create an API as described in one of the following tasks: [“Create an API to invoke the CICS catalog manager services” on page 115](#), [“Create an API to invoke the IMS phone book service” on page 131](#), or [“Create an API to invoke the IBM MQ stock query service” on page 149](#).

- Ensure that the z/OS Connect EE build toolkit is installed. For more information, see “[Installing the z/OS Connect EE build toolkit](#)” on page 199.

About this task

In this task, you use the build toolkit to generate the API archive from the API project that you created in the API toolkit. In an enterprise DevOps workflow, the build toolkit is likely run by a script in basically the same way as demonstrated in the following steps.

Procedure

In this procedure, substitute the Windows example for the platform of your choice. The build toolkit properties files must be encoded in UTF-8.

1. Create a local directory such as C:/DevOpsScenarios on the Windows system.
 2. Download the build toolkit compressed file, zconbt.zip as a binary file, to the same local directory, C:/DevOpsScenarios.
- The build toolkit is in the z/OS Connect EE <installation_path> and is called zconbt.zip.
3. Extract the zconbt.zip file into a zconbt sub directory of the C:/DevOpsScenarios local directory.
 4. Add the **zconbt** executable file to your path. For example, enter the following command: **set PATH=C:/DevOpsScenarios/zconbt/bin;%PATH%**
 5. Obtain the API project folder from your source control management system and place it under C:/DevOpsScenarios

For example, if your Eclipse workspace location is C:/Program Files/IBM/rationalsdp/workspace, you would find a catalog/, contacts/, or stockManager/ folder there. Copy this folder into C:/DevOpsScenarios

6. Run the build toolkit **zconbt** command to create the API archive file.

The build toolkit has the following syntax:

```
zconbt --projectDirectory=C:/DevOpsScenarios/<APIProjectFolder> --file=C:/DevOpsScenarios/<apiName>.aar
```

Use the **-pd** parameter to specify the project directory, and the **-f** parameter to specify the path and the file name of the generated API archive. In a command prompt window, depending on the name of your project, enter one of the following commands:

```
zconbt --projectDirectory=C:/DevOpsScenarios/catalog --file=C:/DevOpsScenarios/catalog.aar
```

or

```
zconbt --projectDirectory=C:/DevOpsScenarios/contacts --file=C:/DevOpsScenarios/contacts.aar
```

or

```
zconbt --projectDirectory=C:/DevOpsScenarios/stockManager --file=C:/DevOpsScenarios/stockManager.aar
```

Results

The API archive can be copied to the designated API directory of your server for deployment.

For more information about a general enterprise DevOps workflow and sample DevOps technologies for building the archives and deploying the services and APIs, see “[DevOps with z/OS Connect EE](#)” on page 14.

Related concepts

[“Automated API management” on page 673](#)

Deploy and undeploy your APIs automatically when you add or remove them from the API directory.

Related reference

[“zconbt command syntax” on page 795](#)

The **zconbt** command starts the build toolkit tool. You can use the build toolkit to generate archive files for services, APIs or API requesters.

Configuring HA

These tasks show you how to configure high availability (HA).

Related concepts

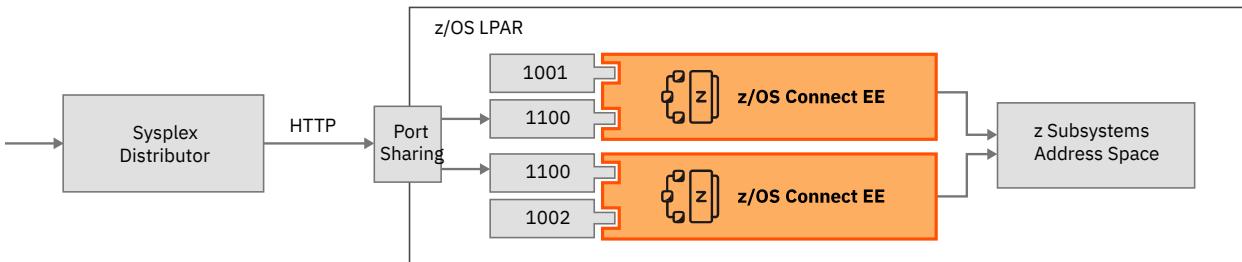
[“High availability” on page 317](#)

With a high availability solution, you can minimize the impact on daily operations of the failure of one or more components within the overall system. The key concepts of a high availability (HA) environment and how they can be implemented with z/OS Connect EE are discussed.

Configure HA for connections to z/OS Connect EE

This scenario shows you how to implement high availability for inbound requests to two z/OS Connect EE servers that use port sharing and shared configuration data.

This scenario is built on the [“Create a server and connect to a System of record \(SoR\)” on page 69](#) scenario and implements high availability for z/OS Connect EE by adding a second z/OS Connect EE server. To simplify the configuration, the scenario describes how to define a configuration that is used by both servers.



Set up TCP/IP port sharing

Set up port sharing so that z/OS Connect EE servers can listen for inbound requests on the same port.

Procedure

1. Choose an unreserved port number to be used for the servers' HTTP listeners. This scenario uses port 1111 as an example, but it can be changed if that port is already in use in your environment.

To display ports that are currently reserved on a single TCP/IP stack, use one of the following methods:

- Enter the following SDSF command, where *procname* is the name of the TCP/IP stack:

```
/D TCPIP,<procname>,N,PORTL
```

- Enter the UNIX System Services command:

```
netstat -o
```

2. Configure the shared port in z/OS to use SHAREPORTWLM.

- a) In the TCP/IP profile for the z/OS LPAR, add or update an existing PORT statement to add an entry for the HTTP port that the server listens on.

In this scenario, port 1111 was chosen, with started task job names that start with "ZCHA" and the SHAREPORTWLM option to allow WLM to manage the incoming workload. The shared port definition in the TCP/IP port profile is as follows.

```
PORT  
1111 TCP ZCHA* SHAREPORTWLM
```

- b) Enable the shared port definition.

Enter the SDSF command:

```
VARY TCPIP,<procname>,OBEYFILE,<datasetname>
```

For further instructions, see the description of the **VARY TCPIP,,OBEYFILE** command in the [z/OS Communications Server: IP System Administrator's Commands](#) documentation.

3. Confirm that the shared port is active by displaying the ports as you did in Step 1.

Display ports that are currently reserved on a single TCPIP stack, use one of the following methods.

For example,

```
PORT# PROT USER FLAGS RANGE SAF NAME  
1111 TCP ZCHA* DASW
```

The S flag indicates that the port is shared and the W flag indicates that the port uses WLM server-specific recommendations.

4. Update the existing `httpEndpoint` element in `server.xml` to use the shared port.

For example,

```
<httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="11111" httpsPort="-1" />
```

Create a shared configuration file

Create a shared configuration file so that servers in the HA environment can share common elements.

About this task

To ensure that both servers in the HA environment can handle the same workload, some elements from the `server.xml` file for each server are shared from a common configuration file. Configuration elements that are unique to a server are defined in a separate `server.xml` file.

Procedure

1. Create directories under `/var/zosconnect` to store the shared configuration file.
For example, run the following command from the `/var/zosconnect` directory.

```
mkdir -p shared/config
```

2. Move the current `server.xml` file that is used by `haserver1` to the shared configuration directory and rename it to `haserver.xml`.
For example, run the following command as a single line:

```
mv /var/zosconnect/servers/haserver1/server.xml  
/var/zosconnect/shared/config/haserver.xml
```

3. Create a file named `server.xml` for `haserver1` in the `/var/zosconnect/servers/haserver1` directory with the following elements:

```
<?xml version="1.0" encoding="UTF-8"?>  
<server description="z/OS Connect EE HA server1">  
    <include location="${shared.config.dir}/haserver.xml"/>  
</server>
```

This configuration file includes the shared `haserver.xml` by using the Liberty property `$ {shared.config.dir}` to reference the directory that contains the shared configuration files. This property points to the directory `WLP_USER_DIR/shared/config`.

Set up shared resources

Move the API and service artifacts to a shared location so they can be accessed by both servers.

About this task

To ensure that workload distribution can route requests to any of the z/OS Connect EE servers that are available in a port shared group, the servers must have the same APIs and services available. For more information, see [“Sharing server configuration” on page 325](#).

Procedure

1. Create directories under /var/zosconnect to store the API archive file, WSBind files, and JSON schemas.

Run the following command from the /var/zosconnect/shared directory:

```
mkdir -p resources/zosconnect/apis resources/zosconnect/services
```

2. Move your API .aar files to /var/zosconnect/shared/resources/zosconnect/apis

Enter the following command on a single line:

```
mv /var/zosconnect/servers/haserver1/resources/zosconnect/apis/*.aar  
/var/zosconnect/shared/resources/zosconnect/apis
```

3. Update the zosconnect_zosConnectAPIs element to configure the shared API location \${shared.resource.dir}/zosconnect/apis.

For example,

```
<zosconnect_zosConnectAPIs  
location="${shared.resource.dir}/zosconnect/apis" updateTrigger="polled" />
```

4. Update the zosconnect_services element to configure the shared services location \${shared.resource.dir}/zosconnect/services.

For example,

```
<zosconnect_services location="${shared.resource.dir}/zosconnect/services"  
updateTrigger="polled" />
```

5. Restart the server for the new API location to take effect.

From SDSF enter the following command to stop the server,

```
/P ZCHA1
```

From SDSF, type / to open the **System Command Extension** pop-up, then enter the command:

```
S BAQSTRT,JOBNAM=ZCHA1,PARMS='haserver1 -clean'
```

6. Test that your API requests still work.

CICS users: rerun “[Test the IPIC connection](#)” on page 70 in the “[Create a server and connect to CICS](#)” on page 69 scenario.

IMS users: rerun “[Test the connection to IMS](#)” on page 73 in the “[Create a server and connect to IMS](#)” on page 72 scenario.

Results

The HA environment is now set up for the first server.

Add a second server

Add a second server to the HA environment.

Procedure

1. Create the second server.

The configuration from the first server is copied across for the second server in a later step so no template is required now. Enter the following command from the <installation_path>/bin directory. For example, /usr/lpp/IBM/zosconnect/v3r0/bin.

zosconnect create haserver2

2. Replace the default Liberty server.xml file with the server.xml file from haserver1.

Run the following command:

```
cp /var/zosconnect/servers/haserver1/server.xml  
/var/zosconnect/servers/haserver2
```

3. Start the z/OS Connect EE server.

To preserve the case of your server name, you must start the server from the System Command Extension window within SDSF.

From SDSF, type / to open the **System Command Extension** window, then enter the command:

```
S BAQSTRT,JOBNAM=ZCHA2,parms='haserver2 --clean'
```

4. Check that the server initialized correctly.

Test HA for connections to z/OS Connect EE

Test the HA configuration and validate that it is distributing work across both servers.

Procedure

1. Start up both z/OS Connect EE servers.

2. Invoke an API to send a request to your System of Record (SoR).

a) For CICS users, invoke the catalog API from the z/OS Connect EE API toolkit by using the "Try it out!" function in the Swagger UI that is embedded in the editor, as described in "[Test the CICS catalog manager API](#)" on page 130.

b) For IMS users, invoke the contacts API from the z/OS Connect EE API toolkit by using the "Try it out!" function in the Swagger UI that is embedded in the editor, as described in "[Test the IMS contacts API](#)" on page 148.

3. Your request to your SoR should have been successful, but which z/OS Connect EE server did it use?

There are various ways we could ascertain which server was used, but the simplest way is:

a) Shut down z/OS Connect EE server 1.

b) Invoke your API again. This should be successful and use z/OS Connect EE server 2.

c) Restart z/OS Connect EE server 1.

d) Shut down z/OS Connect EE server 2.

e) Invoke your API again. This should be successful and use z/OS Connect EE server 1.

f) Restart z/OS Connect EE server 2.

Results

You have verified that your two z/OS Connect EE servers have been configured and successfully work in a high availability environment.

What to do next

For CICS users, follow the steps in “[Configure HA for connections into CICS](#)” on page 174 to configure HA for connections into CICS.

For IMS users, see “[IMS and IMS Connect high availability](#)” on page 321 for information about configuring HA for connections into IMS.

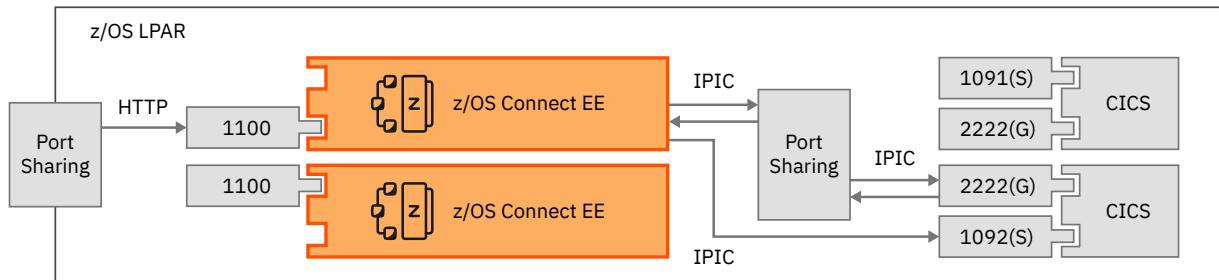
Configure HA for connections into CICS

This scenario shows you how to implement high availability for connections to CICS using the CICS service provider by adding a second CICS region.

This scenario is built on the “[Create a server and connect to CICS](#)” on page 69 scenario. You must complete the steps in that scenario before starting this procedure.

This scenario consists of the following tasks:

- Add a second CICS region in the same way that you created and configured your first CICS region
- Configure your CICS regions for IPIC HA
- Configure z/OS Connect EE for IPIC HA
- Test your HA configuration and validate requests are distributed across both servers to both CICS regions.



Configure your CICS regions for IPIC HA

In this step, you create and configure a z/OS Connect EE server to use the CICS service provider, the z/OS Connect EE API, and services for the Catalog Manager application.

Before you begin

You must create a connection to a second CICS region. Follow the same instructions for creating the first CICS connection in the task “[Create a server and connect to CICS](#)” on page 69, then return here.

The following steps should be repeated for each CICS region.

Note: The name of the specific TCPIPSERVICE must be alphabetically earlier than the generic TCPIPSERVICE.

Procedure

1. Define a specific TCPIPSERVICE to listen on a non-shared port specific to the CICS region.
In this scenario we use port 1091 in CICS-1 and port 1092 in CICS-2
2. Define a generic TCPIPSERVICE in each CICS region to listen on the shared port and set the SPECIFTCPS value to the name of the specific TCPIPSERVICE.
In this scenario the shared port is 2222.

Configure z/OS Connect EE for IPIC HA

Your z/OS Connect EE server must be configured for IPIC HA.

Procedure

In the shared `server.xml` configuration file, locate the `zosconnect_cicsIpicConnection` element, and set `sharedPort=true` and specify the shared port in the `port` attribute.

The definition is:

```
<zosconnect_cicsIpicConnection id="cicsConn" host="localhost" port="2222" sharedPort="true" />
```

Test HA for connections to CICS

Test the IPIC HA configuration and validate that work can be distributed to both CICS regions.

About this task

Verification is easier using just one z/OS Connect EE server.

Procedure

1. Shutdown z/OS Connect EE server 2.
2. Log on to a CICS terminal for each CICS region and check the initial usage count of the Catalog Manager program DFH0XCMN in each region.

Make a note of the values to confirm in later steps that the count was incremented. Enter the following command on each CICS region.

CEMT I PROG(DFH0XCMN)

The use field indicates how many times the program was run.

```
I PROG(DFH0XCMN)
STATUS: RESULTS = OVERTYPE TO MODIFY
Prog(DFH0XCMN) Leng(0000008056) Cob Pro Ena Pri     Ced
             Resc(0000) Use(0000000046) Any Uex Ful Thr Cic           Len
```

3. Invoke the catalog API from the z/OS Connect EE API toolkit by using the "Try it out!" function in the Swagger UI that is embedded in the editor, as described in "[Test the CICS catalog manager API](#)" on [page 130](#).
4. On both CICS regions, check that the usage count was incremented by 1 in one region only.
5. Further API requests will continue to be sent to the same CICS region.
6. To verify that IPIC HA is working correctly, shutdown the CICS region, or close the TCPIPServices for the CICS region that was receiving the HTTP GET requests.
7. Send further API requests.

Verify that these are now sent to the other CICS region by using the following command:

```
CEMT I PROG(DFH0XCMN)
```

8. Restart the shutdown CICS region, or re-open the TCPIPServices.
9. Restart the second z/OS Connect EE server.
10. Send further API requests and observe the usage count of program DFH0XCMN in both CICS regions.

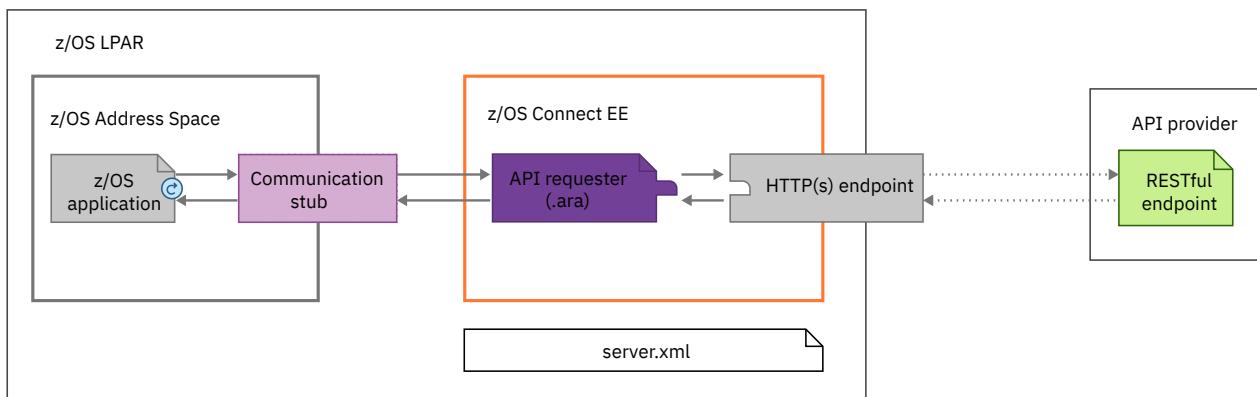
Results

You have successfully configured IPIC high availability for your two CICS regions using port sharing.

Calling an API from a System of Record (SoR)

Follow the steps in these scenarios to learn how to perform tasks to call an API from an SoR.

The following diagram illustrates how the API Requester connects z/OS Connect EE with the z/OS address space so that the z/OS application can call APIs on an SoR.



Call an API from a CICS application

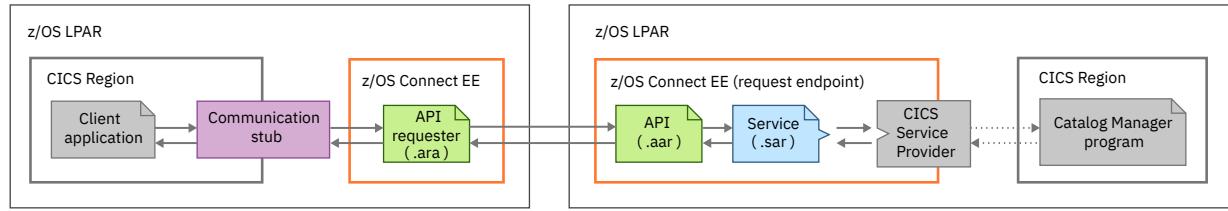
Use this scenario to call an API from a CICS application through z/OS Connect EE.

This scenario involves two CICS applications:

- The CICS catalog manager example application that must be exposed as a REST API for call preparation.
- The CICS application that plays as a client application to call the exposed catalog manager API.

This scenario shows you how to connect the client CICS application to the z/OS Connect EE server, how to connect the z/OS Connect EE server to the request endpoint where the CICS catalog manager API is deployed, how to use the Command Line Interface (CLI) of the build toolkit to generate artifacts for an API requester based on the Swagger file of the API, how to deploy the API requester to the z/OS Connect EE server and how to invoke the API requester from the client CICS application.

The following diagram shows the topology that is used in this scenario:



When a CICS application is developed to call a RESTful service, the z/OS Connect EE server that is used to process the request can be chosen at run time within the CICS application. This option allows for splitting

of workloads between servers, for example, based on business area. This dynamic routing capability is available for CICS applications only. For more information, see [“Overriding the URIMAP in a CICS application” on page 643](#)

Prerequisites

The following facilities are needed to run this scenario.

Software requirements

- z/OS Connect Enterprise Edition V3.0.18 or later installed and the **zconsetup install** command run successfully.
- CICS Transaction Server 5.3 or later installed and a running CICS region.

Prerequisite tasks

You need to complete the following tasks to provide the infrastructure needed by the scenario.

- Make the catalog manager application available as a REST API.

Prepare a REST API for the client CICS application to call by completing the following tasks:

- Prepare the sample CICS application. Follow the instructions in [“Prepare the sample CICS application” on page 59](#).
- Create and configure a server and connect to CICS over IPIC. Follow the instructions in [“Create a server and connect to CICS” on page 69](#)
- Follow the scenario [“Create an API to invoke the CICS catalog manager services ” on page 115](#) as a REST API.

- Use the HTTP method GET to obtain the Swagger file of the catalog manager API.

1. Open a web browser, and enter an HTTP GET request:

```
http(s)://{{host}}:{{port}}/catalogManager/api-docs
```

2. Copy the full content of the response into a local file, name the file `catalogManager.json`, and ensure that the file is encoded in UTF-8.

- Create a z/OS Connect EE server for API requester.

For this scenario, use the specific `apiRequester` template to create a z/OS Connect EE server. Run the **zosconnect** command with the `create` option by using the following syntax:

```
zosconnect create myserver --template=zosconnect:apiRequester
```

For more information, see [“Creating a z/OS Connect Enterprise Edition server” on page 217](#).

- Configure a started task to run the z/OS Connect EE server for API requester. For more information, see [“Starting and stopping z/OS Connect EE” on page 471](#).

Note: In this scenario, two z/OS Connect EE servers are running. One is the request endpoint where the CICS catalog manager API is deployed. The other one is where the API requester is deployed. In this scenario, *request endpoint* refers to the z/OS Connect EE server and the server that is mentioned in the following sub topics is where the API requester is deployed.

Configuring the z/OS Connect EE server to access the request endpoint

To route the API call request issued by the client CICS application from the z/OS Connect EE Server to the request endpoint, you must configure the z/OS Connect EE server to establish a connection with the request endpoint.

Procedure

1. Open the `server.xml` configuration file for editing.
2. Modify the `zosconnect_endpointConnection` element. Set the value of the **host** and **port** attribute to the host and port of the request endpoint.

For more advanced settings about the connection between the z/OS Connect EE server and the request endpoint, see [“Establishing a connection from z/OS Connect EE to the request endpoint” on page 628](#).

Configuring CICS to access the z/OS Connect EE server

To enable the client CICS application to call the catalog manager API through the z/OS Connect EE, enable the communication stub in the CICS region.

About this task

The z/OS Connect EE communication stub is a module that establishes HTTP connections with the z/OS Connect EE server, transfers data between z/OS applications and z/OS Connect EE, and handles status and return codes that are issued by z/OS Connect EE. Before your CICS application can make RESTful API calls, you must configure the communication stub in your CICS region.

To set up the z/OS Connect EE communication stub in CICS, you must have the following resources in your CICS region:

A TDQUEUE resource with the name BAQQ

This resource defines the transient data queue that is used by the z/OS Connect EE communication stub to log communication stub error messages. By default, messages are printed to DD BAQOUT in the CICS job log.

A URIMAP resource with the name BAQURIMP

This resource handles HTTP client requests from the communication stub to the z/OS Connect EE server. Other URIMAP resources can also be configured and used as required by the BAQCSTUB program.

A PROGRAM resource with the name BAQCSTUB

This resource defines attributes for the z/OS Connect EE communication stub program BAQCSTUB.

Sample resources are supplied with z/OS Connect EE and are provided in a CSD group with the name BAQAPIR.

Use the resources as supplied. Depending on the topology you want to use and your CICS and z/OS Connect EE environment, you might need to modify the attributes of the BAQURIMP resource definition.

Procedure

1. Import the BAQAPIR CSD group into your CICS system by using the DFHCSDUP utility program.
The supplied BAQAPIR CSD group is in the `<hlq>.SBAQSAMP` data set.
 - a) Edit the job control statements that you can use to invoke DFHCSDUP. Include a DD statement that references the `<hlq>.SBAQSAMP` data set.
 - b) Run the DFHCSDUP utility program.
2. Modify the BAQURIMP URIMAP resource definition. Set the value of the **HOST** and **PORT** attribute to the host and port of your z/OS Connect EE server.
3. Install the BAQQ, BAQURIMP, and BAQCSTUB resource definitions in your CICS system.

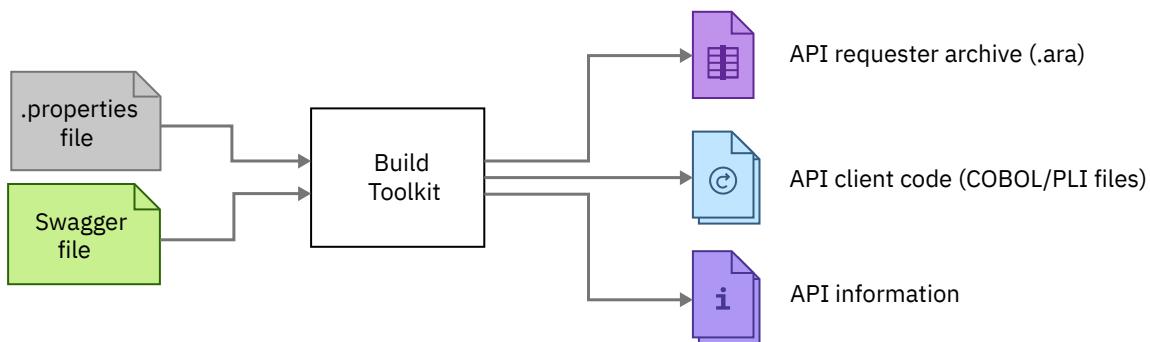
4. Define and install a new CICS LIBRARY to include the <hlq>.SBAQLIB1 data set.

Using the build toolkit to generate artifacts from the Swagger file

Run the build toolkit command line interface (CLI) to generate artifacts for the API requester.

About this task

For this scenario, we use the CLI of the build toolkit to generate all the artifacts to enable the client CICS application to call the catalog manager API. These artifacts include an API requester archive (.ara file) and a summary report for the catalog manager API, and API information files, data structures and logs for each operation in the API. In this task, you will use the build toolkit to create these artifacts from the `catalogManager.json` Swagger file that is obtained in the prerequisite task. For more information, see [“Using the build toolkit to generate artifacts for an API requester” on page 617](#).



Procedure

1. Create a local directory called `buildToolkit` under the `/u/{userid}` directory.
2. Copy the build toolkit compressed file, `zconbt.zip` as a binary file, from the product installation directory and extract the `zconbt.zip` file to the `buildToolkit` directory.
3. Create directories and data sets for the artifacts to be generated.
 - a) Create a directory called `apiReqScenario` under the `/u/{userid}` directory to store the `catalogManager.json` file.
 - b) Create a directory called `archives` under the `apiReqScenario` directory to store the generated API requester archive and summary report.
 - c) Create a directory called `logs` under the `apiReqScenario` directory to store the generated log file for each operation that is defined in the API.
 - d) Create a data set on MVS™ to store the generated API information file and data structures, for example, `TEST.CATALOG.COBOL`.
4. Upload the `catalogManager.json` file, as a binary file, to the `apiReqScenario` directory.
5. Create a local properties file named `catalogManager.properties` and enter the directory or data set names and other attribute values:

```

apiDescriptionFile=../apiReqScenario/catalogManager.json
dataStructuresLocation=TEST.CATALOG.COBOL
apiInfoFileLocation=TEST.CATALOG.COBOL
connectionRef=RemoteAPIConn
logFileDirectory=../apiReqScenario/logs
language=COBOL

```

6. Upload the `catalogManager.properties` file, as a binary file, to the `apiReqScenario` directory.
7. Enter the `buildToolkit` directory and enter the **`zconbt.zos`** command:

```

./bin/zconbt.zos --properties ../apiReqScenario/catalogManager.properties --file

```

```
.../apiReqScenario/archives/catalogManager.ara
```

8. When processing of the command is completed, a message is returned, indicating whether the artifacts were built successfully. If any errors occur, make the necessary corrections and repeat the procedure.

Results

Several artifacts for the API requester are created in the directories as you have configured. For each API, an API requester archive (.ara) file and a summary report are generated. For each API operation, an API information file, request data structures, and response data structures are generated. The following table shows the directory structure (/ data set) and contents created:

<i>Table 12. Directory paths (/ data sets) and contents</i>	
Path (/data set)	Contents
./apiReqScenario/	<ul style="list-style-type: none">• Directory archives• Directory logs• catalogManager.properties• catalogManager.json
./apiReqScenario/archives	<ul style="list-style-type: none">• catalogManager.ara: the API requester file that can be deployed to the z/OS Connect EE server.• catalogManager-summary.txt: the summary report that contains information about all the data structures and API information file.
./apiReqScenario/logs	Logs for each operation defined in the catalogManager.json file: <ul style="list-style-type: none">• getInquireCatalog.log: the log file for the getInquireCatalog operation• getInquireSingle.log: the log file for the getInquireSingle operation• postPlaceOrder.log: the log file for the postPlaceOrder operation

Table 12. Directory paths (/ data sets) and contents (continued)

Path (/data set)	Contents
TEST.CATALOG.COBOL	<ul style="list-style-type: none"> • API00I01.cbl: the API information file of the getInquireCatalog operation • API00P01.cbl: the response data structure of the getInquireCatalog operation • API00Q01.cbl: the request data structure of the getInquireCatalog operation • API01I01.cbl: the API information file of the getInquireSingle operation • API01P01.cbl: the response data structure of the getInquireSingle operation • API01Q01.cbl: the request data structure of the getInquireSingle operation • API02I01.cbl: the API information file of the postPlaceOrder operation • API02P01.cbl: the response data structure of the postPlaceOrder operation • API02Q01.cbl: the request data structure of the postPlaceOrder operation

Deploying the API requester to the z/OS Connect EE server

The API requester archive (.ara file) created in the previous task must be uploaded to the z/OS Connect EE server's API requester directory.

Procedure

1. Copy the catalogManager.ara file, as a binary file, to the \${server.config.dir}/resources/zosconnect/apiRequesters.

Note:

The apiRequesters directory is automatically created under the \${server.config.dir}/resources/zosconnect directory when the z/OS Connect EE server is created with the apiRequester template. You can also specify the location where the API requester archive is stored by setting the **location** attribute of the zosconnect_apiRequesters element in server.xml. For more information, see [“Configuring z/OS Connect EE to support API requesters” on page 304](#).

2. Refresh the z/OS Connect EE server artifacts with the **Modify** command:

```
/modify <jobname>,zcon,refresh
```

You must specify the job name of the target server based on your situation. For more information, see [“The MODIFY command” on page 473](#).

3. Open a web browser and send an HTTP GET request to list all the installed API requesters:

```
http://{host}:{port}/zosConnect/apiRequesters
```

If the catalog manager API requester is installed correctly, you will receive the following response:

```
{
  "apiRequesters": [
    {
      "name": "catalogmanager_1.0.0",
      "version": "1.0.0",
    }
  ]
}
```

```

    "description": "API for the CICS catalog manager sample application.",
    "status": "Started",
    "apiRequesterUrl": "http://{host}:{port}/zosConnect/apiRequesters/catalogmanager_1.0.0",
    "connection": "RemoteAPIconn"
}
]
}

```

Developing the client CICS application to call the catalog manager API

Now you can develop the client CICS application to call the Catalog Manager API. To call the API, the client CICS application must include the generated request and response data structures and the API information file to communicate with the z/OS Connect EE server.

In this scenario, a sample client CICS application is provided. You can get the sample code from the hlq.SBAQSAMP data set and define a transaction to invoke this sample program. The sample program name is BAQCATPO. This sample CICS application tries to call the catalog manager API to invoke the placeOrder service. The names of the generated artifacts for this postPlaceOrder operation are as follows:

- The request copybook: API02Q01
- The response copybook: API02P01
- The API information file: API02I01

Note: You can find the detailed information about the postPlaceOrder operation in the catalogManager-summary.txt, an artifact that was generated in the ./apiReqScenario/archives/ directory by the Build toolkit.

The following excerpts from the sample code shows the key steps to call a REST API from a CICS application:

1. For the postPlaceOrder operation to be called, include the generated request data structure, response data structure and the API info file.

```

01 REQUEST.
COPY API02Q01.
01 RESPONSE.
COPY API02P01.
01 API-INFO-OPER1.
COPY API02I01.

```

2. Declare variables for the request and response. For more information about the variables, see [Variables to declare](#).

01 BAQ-REQUEST-PTR	USAGE POINTER.
01 BAQ-REQUEST-LEN	PIC S9(9) COMP-5 SYNC.
01 BAQ-RESPONSE-PTR	USAGE POINTER.
01 BAQ-RESPONSE-LEN	PIC S9(9) COMP-5 SYNC.
77 COMM-STUB-PGM-NAME	PIC X(8) VALUE 'BAQCSTUB' .

3. Populate the values for the request.

```

MOVE 10 TO ca-item-ref-number IN REQUEST.
MOVE 1 TO ca-quantity-req IN REQUEST.

```

4. Prepare the data for call.

```

SET BAQ-REQUEST-PTR TO ADDRESS OF REQUEST.
MOVE LENGTH OF REQUEST TO BAQ-REQUEST-LEN.
SET BAQ-RESPONSE-PTR TO ADDRESS OF RESPONSE.
MOVE LENGTH OF RESPONSE TO BAQ-RESPONSE-LEN.

```

5. Call the communication stub.

```
CALL COMM-STUB-PGM-NAME USING
```

```
BY REFERENCE API-INFO-OPER1  
BY REFERENCE BAQ-REQUEST-INFO  
BY REFERENCE BAQ-REQUEST-PTR  
BY REFERENCE BAQ-REQUEST-LEN  
BY REFERENCE BAQ-RESPONSE-INFO  
BY REFERENCE BAQ-RESPONSE-PTR  
BY REFERENCE BAQ-RESPONSE-LEN.
```

6. Retrieve the values from the response, when the call is returned.

```
IF BAQ-SUCCESS THEN  
    DISPLAY "Congratulations!"  
    DISPLAY "Return Code: " ca-return-code  
    DISPLAY "Response Message: " ca-response-message  
ELSE  
    ...  
END-IF
```

Errors might come from the API, the z/OS Connect EE server, or the z/OS subsystems. A return code and a status code are included in the response message to provide the error detail. In a COBOL program, the special variable *RETURN-CODE* is set to the value of BAQ-RETURN-CODE. For more information about error handling, see [“Error handling for API requester calls” on page 648](#).

Testing the scenario

Test the client CICS application for the API call.

About this task

In this step, you compile and link-edit the client CICS application to test whether it can call the catalog manager API that is deployed in the request endpoint through the z/OS Connect EE server.

Procedure

1. Add the following data sets to the JCL for compiling your application:
 - TEST.CATALOG.COBOL: containing the generated data structures and the generated API information file.
 - hlq.SBAQCOB: containing the BAQRINFO data structure that contains the parameters to pass to the communication stub for communication with the z/OS Connect EE server.
2. Compile and link-edit your program.

Results

You have called the catalog manager API from the client CICS application and received the following response:

```
Congratulations!  
Return code: 00  
Response Message: ORDER SUCCESSFULLY PLACED
```

Call an API from an IMS or z/OS application

Use this scenario to call an API from a IMS or non-CICS z/OS application through z/OS Connect EE. This scenario is based on the IMS phonebook application.

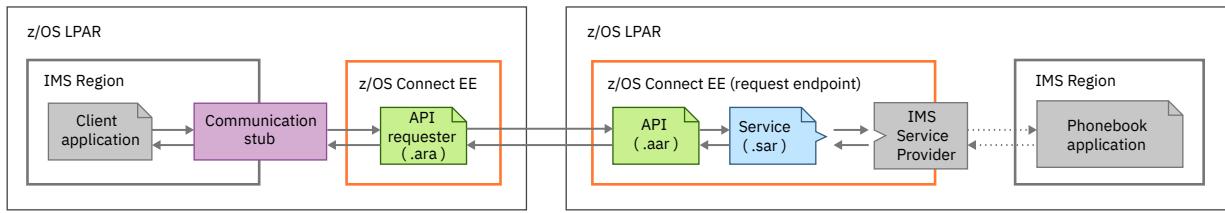
This scenario involves two applications:

- The IMS phonebook application that must be exposed as a REST API for call preparation.
- The IMS application (or non-CICS z/OS application) that acts as a client application to call the exposed API.

This scenario shows you the following key steps:

1. Connect the client application to the z/OS Connect EE server.
2. Connect the z/OS Connect EE server to the request endpoint where the API for accessing the phonebook application is deployed.
3. Use the Command Line Interface (CLI) of the build toolkit to generate artifacts for an API requester based on the Swagger file of the API.
4. Deploy the generated API requester to the z/OS Connect EE server.
5. Invoke the API requester from the client application.
6. Test the scenario.

The following diagram shows the topology that is used in this scenario:



Prerequisites for the API requester scenario

The following facilities are needed to run this scenario.

Before you begin

You must complete the following tasks to provide the infrastructure needed by the scenario.

- z/OS Connect Enterprise Edition V3.0.1 or later is installed.
- The **zconsetup install** command ran successfully.
- In your IMS system, the phonebook application is accessible as a REST API.

To prepare a REST API for the client IMS or z/OS application to call, you must follow the steps in the following scenarios:

1. [“Create an IMS service” on page 89](#). In this scenario, you create a phone book service based on the phonebook application.
2. [“Create an API to invoke the IMS phone book service” on page 131](#). In this scenario, you create a contacts API to act on the phonebook service.

Procedure

1. Use the HTTP method GET to obtain the Swagger file of the contacts API.

a) Open a web browser, and enter an HTTP GET request:

```
http(s)://{{host}}:{{port}}/contacts/api-docs
```

b) Copy the full content of the response into a local file, name the file phonebook.json, and ensure that the file is encoded in UTF-8.

2. Create a z/OS Connect EE server for the API requester scenario.

For this scenario, use the specific `apiRequester` template to create a z/OS Connect EE server. Run the **zosconnect** command with the `create` option by using the following syntax:

```
zosconnect create myserver --template=zosconnect:apiRequester
```

For more information, see [“Creating a z/OS Connect Enterprise Edition server” on page 217](#)

3. Configure a started task to run the z/OS Connect EE server for API requester. For more information, see [“Starting and stopping z/OS Connect EE” on page 471](#).

Note: In this scenario, two z/OS Connect EE servers are running. One is the API request endpoint where the contacts API is deployed. The other one is where the API requester is deployed. To distinguish between these two servers, the former z/OS Connect EE server is referred to as the *request endpoint*. The following sub topics refer to the server where the API requester is deployed.

Configuring the z/OS Connect EE server to access the request endpoint

To route the API call request issued by the client application from the z/OS Connect EE Server to the request endpoint, you must configure the z/OS Connect EE server to establish a connection with the request endpoint.

Procedure

1. Open the `server.xml` configuration file for editing.
2. Modify the `zosconnect_endpointConnection` element. Set the value of the **host** and **port** attribute to the host and port of the request endpoint.

For more advanced settings about the connection between the z/OS Connect EE server and the request endpoint, see [“Establishing a connection from z/OS Connect EE to the request endpoint” on page 628](#).

Configuring IMS to access the z/OS Connect EE server

To enable the client application to call the contacts API through the z/OS Connect EE, enable the communication stub in IMS.

About this task

The z/OS Connect EE communication stub is a module that establishes HTTP connections with the z/OS Connect EE server, transfers data between z/OS applications and z/OS Connect EE, and handles status and return codes that are issued by z/OS Connect EE. Before your application can make RESTful API calls, you must configure the communication stub in IMS.

Procedure

1. Include the load module BAQCSTUB from the hlq.SBAQLIB data set into the IMS STEPLIB concatenation.
2. In your message processing region (MPR) job, specify the URI and port number for the z/OS Connect EE server in the CEEOPTS DD statement.

```
CEEOPTS DD *  
ENVAR("BAQURI=MYSERVER.MY.COM", "BAQPORT=10053")
```

For more information about options you can specify, see [“Configuring IMS to access z/OS Connect EE for API calls” on page 309](#).

3. Compile and link your IMS application with CEEUOPT to set POSIX(ON). The following example demonstrates how to assemble a CEEUOPT CSECT to set POSIX(ON), and link the CEEUOPT CSECT to the load module of the IMS application being run.

```
//APIPRGM JOB ...  
/*  
/* STEP 1: ASSEMBLE CEEUOPT CSECT  
/*  
//CEEUOPT EXEC PGM=ASMA90,COND=(4,LT),REGION=0M,  
// PARM='OBJECT'  
//SYSLIB DD DSN=SYS1.MACLIB,DISP=SHR  
// DD DSN=CEE.SCEEMAC,DISP=SHR  
//SYSIN DD *  
CEEUOPT CSECT  
CEEUOPT AMODE ANY  
CEEUOPT RMODE ANY  
    CEEXOPT POSIX=(ON)  
END  
/*  
//SYSLIN DD DSN=<HLQ>.ASM.OBJECT(CEEUOPT),DISP=SHR  
//SYSPRINT DD DSN=<HLQ>.ASM.LISTING(CEEUOPT),DISP=SHR  
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(1,1))  
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(1,1))  
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(1,1))  
/*  
/* STEP 2: COMPILE AND LINK COBOL PROGRAM  
/*  
//APIPRGM EXEC IGYWCL,LNGPRFX='IGYV5R20',  
// LIBPRFX='SYS1',  
// PARM.COBOL='TEST SOURCE',  
// PARM.LKED='LIST MAP XREF'  
//COBOL.SYSIN DD DSN=<HLQ>.COBOL.SOURCE(APIPRGM),DISP=SHR  
//COBOL.SYSLIB DD DSN=<HLQ>.COBOL.COPYLIB,DISP=SHR  
//COBOL.SYSLIN DD DSN=<HLQ>.COBOL.OBJECT(APIPRGM),DISP=SHR  
//COBOL.SYSDEBUG DD DSN=<HLQ>.COBOL.SYSDEBUG(APIPRGM),DISP=SHR  
//COBOL.SYSPRINT DD DSN=<HLQ>.COBOL.LISTING(APIPRGM),DISP=SHR  
//LKED.OBJECT DD DSN=<HLQ>.ASM.OBJECT,DISP=SHR  
// DD DSN=<HLQ>.COBOL.OBJECT,DISP=SHR  
//LKED.RESLIB DD DSN=IMS.SDFSRESL,DISP=SHR  
//LKED.SYSLIN DD *  
INCLUDE OBJECT(CEEUOPT)
```

```

INCLUDE OBJECT(APIPRGM)
ENTRY APIPRGM
NAME APIPRGM(R)
/*
//LKED.SYSLMOD DD DSN=IMS.PGMLIB(APIPRGM),DISP=SHR

```

Important:

- Ensure that POSIX is set to ON to allow Language Environment to access the UNIX System Services environment for the required HTTP-enabling services. A POSIX(ON) environment also requires the user ID associated with the address space that uses these HTTP-enabling services to have an OMVS segment defined and associated with it.
- Persistent connections are automatically enabled between the z/OS Connect EE server and the communication stub in the IMS region, which requires z/OS Connect EE V3.0.10 or later.
- There are some restrictions when using COBOL programs. For more information, see [“Configuring other z/OS applications to access z/OS Connect EE for API calls” on page 312](#).

Using the build toolkit to generate artifacts from the Swagger file

Run the build toolkit command line interface (CLI) to generate artifacts for the API requester.

About this task

For this scenario, we use the CLI of the build toolkit to generate all the artifacts to enable the client IMS or z/OS application to call the contacts API. These artifacts include an API requester archive (.ara file) and a summary report for the catalog manager API, and API information files, data structures and logs for each operation in the API. In this task, you will use the build toolkit to create these artifacts from the phonebook.json Swagger file that is obtained in the prerequisite task. For more information, see [“Using the build toolkit to generate artifacts for an API requester” on page 617](#).

Procedure

1. Create a local directory called buildToolkit under the /u/{userid} directory.
2. Copy the build toolkit compressed file, zconbt.zip as a binary file, from the product installation directory and extract the zconbt.zip file to the buildToolkit directory.
You can use the java jar command to extract zip files.
3. Create directories and data sets for the artifacts to be generated.
 - a) Create a directory called apiReqScenario under the /u/{userid} directory to store the phonebook.json file.
 - b) Create a directory called archives under the apiReqScenario directory to store the generated API requester archive and summary report.
 - c) Create a directory called logs under the apiReqScenario directory to store the generated log file for each operation defined in the API.
 - d) Create a data set on MVS to store the generated API information file and data structures, for example, TEST.PHBK.COBOL.
4. Upload the phonebook.json file, as a binary file, to the apiReqScenario directory.
5. Create a local properties file named phonebook.properties and enter the directory or data set names and other attribute values:

```

apiDescriptionFile=../apiReqScenario/phonebook.json
dataStructuresLocation=TEST.PHBK.COBOL
apiInfoFileLocation=TEST.PHBK.COBOL
connectionRef=RemoteAPIconn
logFileDirectory=../apiReqScenario/logs
language=COBOL
requesterPrefix=PHB

```

The default prefix is API. For this example, we specify PHB.

6. Upload the phonebook.properties file, as a binary file, to the apiReqScenario directory.
7. Enter the buildToolkit directory and enter the **zconbt.zos** command:

```
./bin/zconbt.zos --properties ../apiReqScenario/phonebook.properties --file
../apiReqScenario/archives/contacts.ara
```

8. When processing of the command is completed, a message is returned, indicating whether the artifacts were built successfully. If any errors occur, make the necessary corrections and repeat the procedure.

Results

Several artifacts for the API requester are created in the directories as you have configured. For each API an API requester archive (.ara) file and a summary report are generated. For each API operation, an API information file, request data structures and response data structures are generated. The following table shows the directory structure (/ data set) and contents created:

<i>Table 13. Directory paths (/ data sets) and contents</i>	
Path (/data set)	Contents
./apiReqScenario/	<ul style="list-style-type: none"> • Directory archives • Directory logs • phonebook.properties • phonebook.json
./apiReqScenario/archives	<ul style="list-style-type: none"> • phonebook.ara: the API requester file that can be deployed to the z/OS Connect EE server. • phonebook-summary.txt: the summary report that contains information about all the data structures and API information file.
./apiReqScenario/logs	Logs for each operation defined in the contacts.json file.

Table 13. Directory paths (/ data sets) and contents (continued)

Path (/data set)	Contents
TEST.PHBK.COBOL	<ul style="list-style-type: none"> • PHB00I01.cbl: the API information file of the postPhonebookService operation • PHB00P01.cbl: the response data structure of the postPhonebookService operation • PHB00Q01.cbl: the request data structure of the postPhonebookService operation • PHB01I01.cbl: the API information file of the getPhonebookService operation • PHB01P01.cbl: the response data structure of the getPhonebookService operation • PHB01Q01.cbl: the request data structure of the getPhonebookService operation • PHB02I01.cbl: the API information file of the putPhonebookService operation • PHB02P01.cbl: the response data structure of the putPhonebookService operation • PHB02Q01.cbl: the request data structure of the putPhonebookService operation • PHB03I01.cbl: the API information file of the deletePhonebookService operation • PHB03P01.cbl: the response data structure of the deletePhonebookService operation • PHB03Q01.cbl: the request data structure of the deletePhonebookService operation

Deploying the API requester to the z/OS Connect EE server

The API requester archive (.ara file) created in the previous task must be uploaded to the z/OS Connect EE server's API requester directory.

Procedure

1. Copy the phonebook.ara file, as a binary file, to the \${server.config.dir}/resources/zosconnect/apiRequesters.

Note:

The apiRequesters directory is automatically created under the \${server.config.dir}/resources/zosconnect directory when the z/OS Connect EE server is created with the apiRequester template. You can also specify the location where the API requester archive is stored by setting the **location** attribute of the zosconnect_apiRequesters element in server.xml. For more information, see [“Configuring z/OS Connect EE to support API requesters” on page 304](#).

2. Refresh the z/OS Connect EE server artifacts with the **Modify** command:

```
/modify <jobname>,zcon,refresh
```

You must specify the job name of the target server based on your situation. For more information, see [“The MODIFY command” on page 473](#).

3. Open a web browser and send an HTTP GET request to list all the installed API requesters:

```
http://{host}:{port}/zosConnect/apiRequesters
```

If the API requester is installed correctly, you will receive the following response:

```
{  
  "apiRequesters": [  
    {  
      "name": "contacts_1.0.0",  
      "version": "1.0.0",  
      "description": "This is an API for managing contacts.",  
      "status": "Started",  
      "apiRequesterUrl": "http://{host}:{port}/zosConnect/apiRequesters/contacts_1.0.0",  
      "connection": "RemoteAPIconn"  
    }  
  ]  
}
```

Developing the client application to call the contacts API

Now you can develop the client IMS or z/OS application to call the contacts API. To call the API, the client application must include the generated request and response data structures and the API information file to communicate with the z/OS Connect EE server.

In this scenario, a sample client application is provided. You can get the sample application from the hlq.SBAQSAMP data set. The sample program name is BAQPHBK1 (for IMS), or BAQPHBKB (for other applications). This sample application tries to call the contacts API to invoke the phonebook service.

A description of the sample programs are as follows:

- BAQPHBK1 - A sample IMS message processing program (MPP). Work with your IMS system programmer or check the *IMS product documentation* for instructions on how to define this sample as an MPP to IMS as well as a transaction code through which to invoke it.
- BAQPHBKB - A sample z/OS batch program.

The names of the generated artifacts for the GET operation are as follows:

- The request copybook: PHB01Q01
- The response copybook: PHB01P01
- The API information file: PHB01I01

Tip: You can find the detailed information about the operations in the generated phonebook-summary.txt file.

The following information walks you through the sample code that demonstrates the steps to call a REST API from an IMS or z/OS application:

1. For the GET operation to be called, include the generated request data structure, response data structure, and the API information file.

```
01 REQUEST.  
COPY PHB01Q01.  
01 RESPONSE.  
COPY PHB01P01.  
01 API-INFO-OPER1.  
COPY PHB01I01.
```

2. Declare variables for the request and response. For more information about the variables, see [Variables to declare](#).

01 BAQ-REQUEST-PTR	USAGE POINTER.
01 BAQ-REQUEST-LEN	PIC S9(9) COMP-5 SYNC.
01 BAQ-RESPONSE-PTR	USAGE POINTER.
01 BAQ-RESPONSE-LEN	PIC S9(9) COMP-5 SYNC.

3. Specify the stub program name.

```
77 COMM-STUB-PGM-NAME          PIC X(8) VALUE 'BAQCSTUB'.
```

4. Prepare the data for call.

```
SET BAQ-REQUEST-PTR TO ADDRESS OF REQUEST
MOVE LENGTH OF REQUEST TO BAQ-REQUEST-LEN
SET BAQ-RESPONSE-PTR TO ADDRESS OF RESPONSE
MOVE LENGTH OF RESPONSE TO BAQ-RESPONSE-LEN
```

5. Populate the values for the request.

```
MOVE IN-LAST-NAME TO lastName of REQUEST
MOVE 10 TO lastName-length OF REQUEST
```

6. Call the communication stub.

```
CALL COMM-STUB-PGM-NAME USING
      BY REFERENCE API-INFO-OPER1
      BY REFERENCE BAQ-REQUEST-INFO
      BY REFERENCE BAQ-REQUEST-PTR
      BY REFERENCE BAQ-REQUEST-LEN
      BY REFERENCE BAQ-RESPONSE-INFO
      BY REFERENCE BAQ-RESPONSE-PTR
      BY REFERENCE BAQ-RESPONSE-LEN.
```

7. Retrieve the values from the response, when the call is returned.

```
IF BAQ-SUCCESS THEN
  DISPLAY "PHBKAPI SUCCESS "
  DISPLAY "Out Message: "
  DISPLAY OUT-MESSAGE2(1:OUT-MESSAGE2-length)
  DISPLAY "First Name: "
  DISPLAY OUT-FIRST-NAME2(1:OUT-FIRST-NAME2-length)
  DISPLAY "Extension: "
  DISPLAY OUT-EXTENSION2(1:OUT-EXTENSION2-length)
  DISPLAY "ZipCode: "
  DISPLAY OUT-ZIP-CODE2(1:OUT-FIRST-NAME2-length)
```

Errors might come from the API, the z/OS Connect EE server, or the z/OS subsystems. A return code and a status code are included in the response message to provide the error detail. In a COBOL program, the special variable *RETURN-CODE* is set to the value of BAQ-RETURN-CODE. For more information about error handling, see [“Error handling for API requester calls” on page 648](#).

Testing the scenario

Test the client IMS application for the API call.

About this task

In this step, you will compile and bind the client application to test whether it can call the contacts API.

Procedure

1. Add the following data sets to the JCL for compiling your application:
 - TEST.PHBK.COBOL: containing the generated data structures and the generated API info file.
 - hlq.SBAQC0B: containing the BAQRINFO data structure that contains the parameters to pass to the communication stub for communication with the z/OS Connect EE server.
2. Compile and bind your program.

Results

You have called the contacts API from the client IMS application and would receive the following response:

```
PHBKAPI SUCCESS
Out Message: ENTRY WAS DISPLAYED
First Name: FIRST1
Extension: 8-111-1111
ZipCode: D01/R0
```


Chapter 5. Installation information

Refer to this section when you are ready to install z/OS Connect Enterprise Edition.

z/OS Connect Enterprise Edition V3.0 (program number 5655-CE3) is available to order using standard IBM ordering systems for z/OS products, such as [IBM Shopz](#). z/OS Connect EE is shipped using CBPDO or other customized offerings, on 3590 tape.

Requirements

Your system must meet these hardware and software requirements to install and run z/OS Connect EE.

Support requirements

Minimum required service levels are listed where appropriate. If a specific level is not listed, support is provided for the General Availability (GA) release of the product. Service levels later than those listed are also supported, if the product provides upward compatibility between service releases.

Hardware requirements

z/OS Connect EE runs on any hardware that supports the chosen operating system.

Software requirements

For the most up-to-date information about the specified operating environments for z/OS Connect EE, go to the IBM Software Product Compatibility reports, at <https://www.ibm.com/software/reports/compatibility/clarity/index.html>.

z/OS Connect EE requires IBM z/OS V2.1 or later, IBM 64-bit SDK for z/OS, and Java Technology Edition V8.0.0 or later.

The runtime z/OS Java requirements also apply to all the product-supplied user scripts such as `securityUtility`.

If you are using z/OS V2.1, you must apply the PTF for APAR OA46622: *z/OS Client Web Enablement Toolkit HTTP support*. If you are using z/OS V2.1 or V2.2, also apply the fix for APAR OA50586: *Miscellaneous z/OS Client Web Enablement Toolkit fixes*.

Requirements for creating APIs to access z/OS subsystems

The z/OS Connect EE API toolkit, the Eclipse-based workstation tool for creating APIs, requires IBM Explorer for z/OS Aqua V3.1 and JRE V8.0 or later.

The z/OS Connect EE build toolkit requires JRE V8.0 or later.

Requirements for calling an API from a z/OS application

To call RESTful APIs through the z/OS Connect EE server on platforms other than CICS, z/OS V2.1 or later is required. If HTTPS is used for communication between IMS and z/OS Connect EE, z/OS V2.1, APAR OA50957 must be applied.

For supported versions of SoRs or z/OS application, see the IBM documentation for that product.

Service requirements

If the HWTxxxx modules are not present in the SYS1.CSSLIB data set, download and apply PTF UA79089 for APAR OA46622 and PTF UA92240 for APAR OA50957 and then rerun the APPLY job for z/OS Connect EE V3.

Installing z/OS Connect EE

After you install z/OS Connect Enterprise Edition or z/OS Connect Enterprise Edition Unlimited, you perform the postinstallation tasks.

About this task

The SMP/E installation process puts the product code onto your system.

Procedure

1. Follow the instructions in the Program Directory to install the product using SMP/E.

Table 14. Program Directories for z/OS Connect products

Product	Publication number
z/OS Connect Enterprise Edition	GI13-4414
z/OS Connect Enterprise Edition Unlimited	GI13-4448

Note:

- By default, the product code is installed in `/usr/lpp/IBM/zosconnect/v3r0/bin`.
- You can set the `WLP_USER_DIR` environment variable to the location where you want your server instances to be stored. If you do not set `WLP_USER_DIR`, the default value `/var/zosconnect` is used.

If you change the `WLP_USER_DIR` environment variable to a different location, new servers are stored in the new location. The servers in the old location still exist, but can be used only when you change `WLP_USER_DIR` back to that location again.

- If you are planning to set up an HA environment, you need to customize `WLP_USER_DIR`, so consider this requirement when you plan your installation.

2. Apply the latest maintenance to update the product.

Results

z/OS Connect EE is installed.

What to do next

Perform the postinstallation steps in [“Creating the z/OS Connect EE shared directory” on page 196](#) and [“Setting up the product extensions directory” on page 197](#) before you attempt to start a server.

Consider preparing your new server to provide good diagnostic information if a problem occurs. For more information, see [“Prepare z/OS Connect EE for FFDC” on page 718](#).

Creating the z/OS Connect EE shared directory

Use this information to create the directories that are required by the z/OS Connect EE servers.

About this task

The shared directory is used as a dedicated file system by all z/OS Connect EE server instances on the same LPAR. z/OS Connect EE uses the directory specified by the `WLP_USER_DIR` environment variable.

The default value of `WLP_USER_DIR` is `/var/zosconnect`.

You must complete the following steps before you create and start your first z/OS Connect EE server instance:

Procedure

1. Create the zosconnect mount point in the z/OS UNIX file system: /var/zosconnect.
2. Set up file system security to enable z/OS Connect EE server instances to access the shared directory.
 - a) Change the owner of the directories to the user ID that is administering z/OS Connect EE.
Use the command

```
chown -R owner /var/zosconnect
```

where *owner* is the name of the required owner of the directory and its contents.
 - b) Change the group ownership of the directories to a group that all the z/OS Connect EE server user IDs belong to.
Use the command

```
chggrp -R group /var/zosconnect
```

where *group* is the required group ID of the directory and its contents.
 - c) Give the owner of the directories read, write, and execute permissions, and give the group read, write, and execute permissions. For example, rwxrwx---.
Use the command

```
chmod -R 770 /var/zosconnect
```

where 770 gives read/write/execute to the user and group associated with the directory and its contents.
3. Optional: Use UNIX System Services access control list (ACL) entries to add group or owner permissions for multiple administrators or groups.
When the ACL is complete, activate the FSSEC resource class and use the **setfac1** command.

What to do next

Proceed to [set up the product extensions directory](#).

Setting up the product extensions directory

The final step in the installation of z/OS Connect EE, is to run the **zconsetup** script to create the product extensions directory.

Before you begin

To use the **zconsetup** script, the file system where you installed z/OS Connect EE (*<installation_path>*) must be mounted read/write; if the file system is not mounted read/write, you will receive error messages when you try to complete these steps. The user ID used to run the **zconsetup** script must have read/write access to the following directories:

- The /var/zosconnect directory to create the /var/zosconnect/v3r0/extensions directory.
- The *<installation_path>/wlp/etc* directory to create the symlink to the /var/zosconnect/v3r0/extensions directory.

About this task

A product extension is a structured directory on disk that stores Liberty features. Each product extension must be configured by a properties file and stored under the product extensions directory so that the extension can be detected by the server. For z/OS Connect EE, the product extensions directory is /var/zosconnect/extensions. For more information, see [Liberty product extension](#) in the *WebSphere Application Server Liberty base* documentation.

The **zconsetup** utility must be used once per LPAR because `/var` is not a shared file system. You cannot configure the specific path for the z/OS Connect EE product extensions. However, you can share your z/OS Connect EE configuration across LPARs by mounting a shared zFS as `/var/zosconnect` on each LPAR.

Note: Do not confuse the product extensions directory `/var/zosconnect/extensions` described here with the user extensions directory `<WLP_USER_DIR>/extension`. The user extensions directory contains the files for user-created features. For more information about creating your own Liberty features, see [Developing a Liberty feature in the WebSphere Application Server Liberty base documentation](#).

Procedure

1. Go to the `<installation_path>/bin` directory.
2. Enter the following command to run the **zconsetup** script:

```
zconsetup install
```

3. Optional: You can now make the file system onto which you installed z/OS Connect EE (`<installation_path>`) read-only. Make the file system read/write again only when applying maintenance.

Results

The script makes the following changes to the system:

- Creates the release-specific product extensions directory in the `/var/zosconnect` directory, for example:

```
/var/zosconnect/v3r0/extensions
```

- Creates a symbolic link of the extensions directory in the product z/OS UNIX installation path. For example:

```
/usr/lpp/IBM/zosconnect/v3r0/wlp/etc/extensions
```

- Install the z/OS Connect EE product extension properties file in the product extensions directory.

The following table shows the directory structure and contents of a typical installation:

Table 15. Directory paths and contents	
Path	Contents
<code>/usr/lpp/IBM/zosconnect/v3r0/bin</code>	Product code
<code>/usr/lpp/IBM/zosconnect/v3r0/runtime/lib/</code>	zosconnect feature files
<code>/usr/lpp/IBM/zosconnect/v3r0/wlp/etc/extensions</code>	Symlink to <code>/var/zosconnect/v3r0/extensions</code>
<code>/var/zosconnect/v3r0/extensions</code>	Product extension properties files that contain the absolute path to the service providers.

What to do next

You can proceed to create your server instance.

- For CICS access that uses the WOLA service provider, start with [“Creating a z/OS Connect Enterprise Edition server” on page 217](#).
- For IMS access that uses the IMS service provider, start with [“Using the IMS service provider” on page 238](#).

Installing the z/OS Connect EE build toolkit

Install the z/OS Connect EE build toolkit to create service archive (.sar) files or artifacts for an API requester.

About this task

The build toolkit is available as a command line tool or a Software Development Kit (SDK) for inclusion in other products. For more information, read the Javadoc included in the `zconbt.zip` file or the [Build Toolkit SPI](#) in the *Reference* section. You can also find examples in [GitHub](#).

Note: This task shows you how to install and use the build toolkit on a local workstation. You can also run the build toolkit on z/OS. To install the build toolkit on z/OS, you need to extract the contents of the `zconbt.zip` file into a UNIX System Services directory and ensure your user ID has execute permission to the `zconbt.zos` command in the bin directory.

Procedure

1. Copy the `zconbt.zip` file, in binary mode, from the product installation directory to the file system of your local workstation.
2. On your local workstation, extract the contents of the `zconbt.zip` file into a local directory.
The `zconbt.zip` file contains the following directories:
 - bin contains utility files that are used by **zconbt**.
 - doc contains the Javadoc files.
 - lib contains the Java classes.
 - plugins includes CICS, IMS, MQ, REST client and WOLA as standard and is extensible to third parties.
3. Ensure that your user ID has execute permission to the `zconbt` command in the bin directory.
4. Ensure that the `plugin.properties` file is in the same directory as the `com.ibm.zosconnect.buildtoolkit.jar` file.
5. From the bin directory, enter the command:

`zconbt --help`

If the installation is successful, the command help prints to the command line.

Related tasks

[“Generating service archives for DevOps” on page 560](#)

Use the build toolkit **zconbt** command to generate the service archive from service projects that are created in the API toolkit or from a properties file that defines the service.

Updating z/OS Connect EE

You should update to the latest level of z/OS Connect EE, by applying the latest maintenance.

Before you apply maintenance, shut down all instances of z/OS Connect EE.

Note: All files in the `<WLP_USER_DIR>/servers` directory are preserved when you apply maintenance.

Converting to z/OS Connect Enterprise Edition Unlimited

You can convert your existing z/OS Connect Enterprise Edition server to z/OS Connect Enterprise Edition Unlimited.

Note: You must have z/OS Connect Enterprise Edition V3.0.26 or later installed to convert.

There are two methods that you can use to convert from z/OS Connect Enterprise Edition to z/OS Connect Enterprise Edition Unlimited. The preferred solution is to create a new installation with new SMP/E zones.

The alternative solution is to modify an existing z/OS Connect Enterprise Edition installation. Creating a new installation has no impact to an existing z/OS Connect Enterprise Edition installation, and logically separates the two products. By using this method, you can perform a staged conversion of your servers and, if required, an easy way to revert to your existing installation.

Preferred method

The preferred method is to install z/OS Connect Enterprise Edition Unlimited into new SMP/E zones by following the instructions in the [Program Directory for z/OS Connect Enterprise Edition Unlimited](#). You can continue to use your existing z/OS Connect Enterprise Edition servers (including configuration and artifacts) that are stored in `/var/zosconnect` as these are unaffected by the installation of z/OS Connect Enterprise Edition Unlimited. After you install z/OS Connect Enterprise Edition Unlimited, update your JCL to use the new data sets and z/OS UNIX System Services paths from the z/OS Connect Enterprise Edition Unlimited installation, such as started tasks for your servers and the Angel to point to the new installation.

Alternative method

If you choose to install z/OS Connect Enterprise Edition Unlimited into existing z/OS Connect Enterprise Edition SMP/E zones, you must customize your BAQURCV or BAQURCVE job to use the z/OS Connect Enterprise Edition Unlimited RELFILES and SMPMCS data set (as described in more detail in the following steps).

In the following procedure, `hlq.zcee` is used as the high-level qualifier for data sets used by the existing z/OS Connect Enterprise Edition installation jobs, while `hlq.unlimited` is used as the high level qualifier for the new z/OS Connect Enterprise Edition Unlimited installation jobs. Use these instructions with the [Program Directory for z/OS Connect Enterprise Edition Unlimited](#).

Procedure

1. Copy the jobs from the `hlq.unlimited.IBM.JZC3003.F2` RELFILE to a work data set for editing and submission.
2. Allocate data sets for z/OS Connect Enterprise Edition Unlimited.
 - a) Customize the sample job BAQUALOC in `hlq.unlimited.IBM.JZC3003.F2` RELFILE. Set `@hlq@` for your existing z/OS Connect Enterprise Edition installation.
 - b) Submit the sample job. A return code of 0 indicates success.
3. Define DDDEFs for z/OS Connect Enterprise Edition Unlimited
 - a) Customize the sample job BAQUDEF in `hlq.unlimited.IBM.JZC3003.F2` RELFILE. Set `@hlqgzone@, @tname@, @hlq@, and @dname@` for your existing z/OS Connect Enterprise Edition installation.
 - b) Submit the sample job. A return code of 0 indicates success.
4. Receive z/OS Connect Enterprise Edition Unlimited into the existing z/OS Connect Enterprise Edition Global Zone.
 - a) Customize the sample job, BAQURCV or BAQURCVE, in `hlq.unlimited.IBM.JZC3003.F2` RELFILE. Set `@hlqgzone@` and `@smpeoptions@` for your existing z/OS Connect Enterprise Edition installation.
 - b) **IMPORTANT:** You must set `@downhlq@` to the `hlq.unlimited` RELFILES and SMPMCS data set on the SMPCNTL and SMPPTFIN statements to receive the contents of z/OS Connect Enterprise Edition Unlimited.
 - c) Submit the sample job BAQURCV or BAQURCVE. A return code of 0 indicates success.
5. Apply z/OS Connect Enterprise Edition Unlimited.

- a) Customize the sample job BAQUAPLY in *hlq.unlimited.IBM.JZC3003.F2 RELFILE*. Set the @hlqgzone@, @tname@, and @smpeoptions@ for your existing z/OS Connect Enterprise Edition installation.
 - b) IMPORTANT: In the SMPCTL command, ensure that the SELECT attribute includes any z/OS Connect Enterprise Edition Unlimited maintenance PTFs in addition to JZC3003.
 - c) Submit the sample job twice, first with CHECK, then without. A return code of 0 indicates success.
6. Optional: Accept z/OS Connect Enterprise Edition Unlimited.
- a) Customize the sample job BAQUACPT in *hlq.unlimited.IBM.JZC3003.F2 RELFILE*. Set the @hlqgzone@, @dname@, and @smpeoptions@ for your existing z/OS Connect Enterprise Edition installation.
 - b) Submit the sample job twice, first with CHECK, then without. A return code of 0 indicates success.

Installing z/OS Explorer and the z/OS Connect EE API toolkit

To design and create APIs you need to use the z/OS Connect EE API toolkit that is provided as a plug-in for IBM Explorer for z/OS Aqua.

Before you begin

If you already have IBM Explorer for z/OS Aqua V3.1 installed, you should still follow this procedure, but in step 4, select only the z/OS Connect EE API toolkit plug-in.

About this task

In this task, you download and install IBM Explorer for z/OS Aqua V3.1. You can then install the z/OS Connect EE API toolkit plug-in to define and deploy an API.

The z/OS Connect EE API toolkit can be downloaded from the Download Eclipse tools by following the installation path for Aqua 3.1 (Eclipse 4.6 Neon).

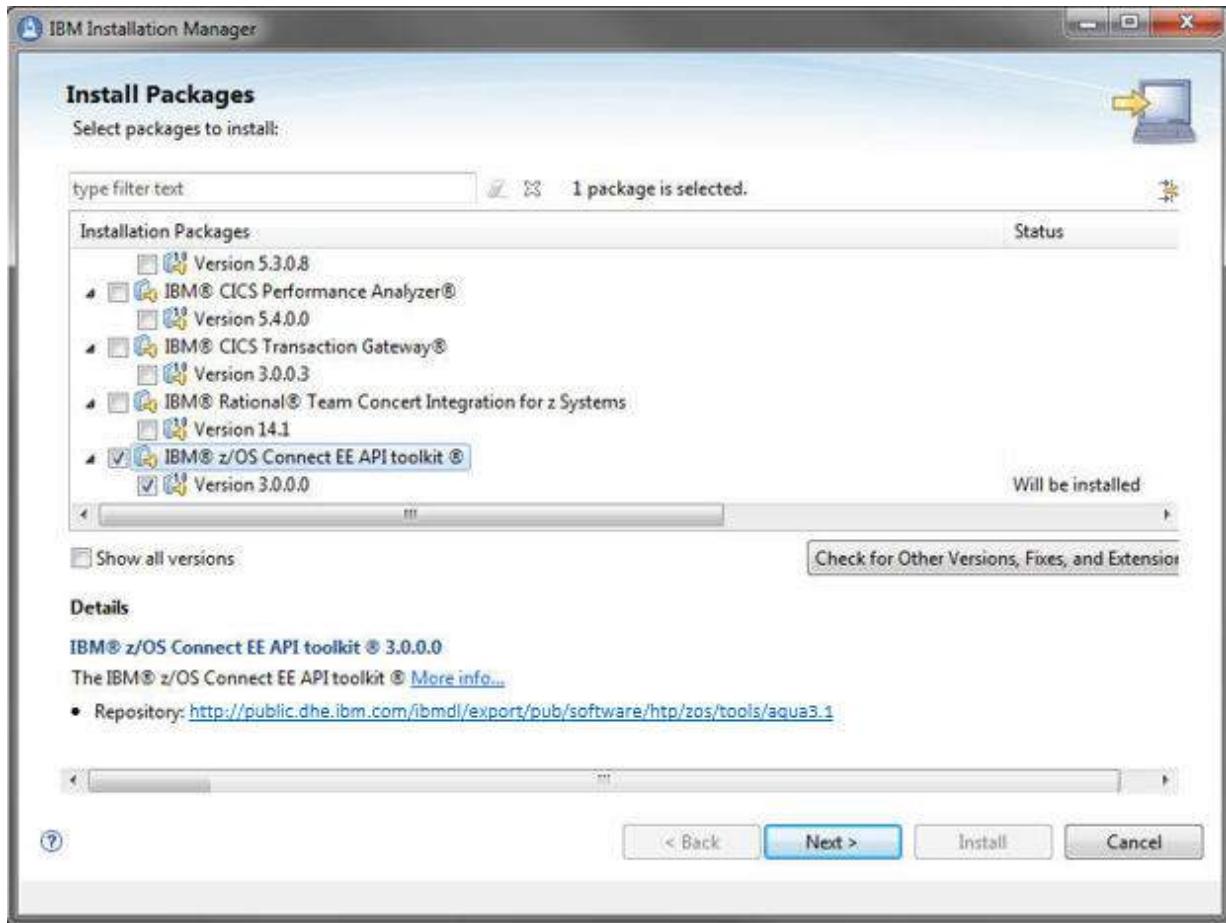
Procedure

1. Go to the IBM Z development tools downloads page.

You can choose either the IBM Installation Manager (IBM IM) or Eclipse p2 installation path. In this example, we will use IM.

2. Follow the IBM IM path and the instructions on the page to choose your installation option and add the repository URL to your Installation Manager.

Important: Follow the instructions for Aqua 3.1 (Eclipse 4.6 Neon) on the download site.



3. On the main IM screen, click **Install**.
4. On the **Install Packages** dialog, select IBM Explorer for z/OS, if not already installed, and IBM z/OS Connect EE API toolkit. Click **Next**.
5. Follow the on-screen instructions to download and install the selected packages.
6. When the installation has completed, close Installation Manager.

Updating z/OS Explorer and the z/OS Connect EE API toolkit

Be aware of these considerations when planning to update z/OS Explorer and the z/OS Connect EE API toolkit.

Updating z/OS Explorer

If you want to update z/OS Explorer, you should do so before starting the z/OS Connect EE update.

Installing the new API toolkit into Eclipse

You can install the new API toolkit into Eclipse while projects are still in flight. However, you should export your projects before beginning the update.

Migrating from z/OS Connect V1

Follow these steps to migrate your API deployments to z/OS Connect Enterprise Edition V3.0.

To migrate an existing configuration based on z/OS Connect V1 to z/OS Connect Enterprise Edition V3.0, you must make the following updates to your `server.xml` file:

1. In the Feature Manager section, replace `<feature>zosconnect-1.0</feature>` or `<feature>zosconnect-1.2</feature>` with:

```
<feature>zosconnect:zosconnect-2.0</feature>
```

- Prefix all z/OS Connect configuration elements for services, interceptors, DataXForm and the z/OS Connect manager with `zosconnect_`. For example:

```
<zosConnectService id='example' name='example' serviceRef='exampleService'/>
```

becomes

```
<zosconnect_zosConnectService id='example' name='example' serviceRef='exampleService'/>
```

Upgrading from z/OS Connect EE V2

Before you upgrade from z/OS Connect EE V2, ensure that you are aware of the compatibility considerations, file location changes, and behavior differences between z/OS Connect EE V3 and z/OS Connect EE V2.

IMS, CICS, and IBM MQ services are created in the z/OS Connect EE API toolkit.

The z/OS Connect EE API toolkit now provides a consistent and integrated service creation experience for the IMS, CICS, and IBM MQ service providers. You must use the API toolkit to create IMS services. IMS Explorer for Development is not compatible with z/OS Connect EE V3. You cannot connect to a V3 server from IMS Explorer for Development.

IMS services that are created for V3 servers are not compatible with V2 servers.

As a result of the consolidation of service creation, deployment, and management processes in z/OS Connect EE V3, IMS services that are created for V3 servers are not compatible with V2 servers.

V2 services can run on V3 servers in toleration mode by configuring the IMS mobile service registry on a V3 server to point to the IMS mobile service registry location in V2. No additional configuration is required. You can continue to create APIs to access these services, but these services must run as-is and cannot be changed. If service changes are required, contact IBM Software Support for migration assistance. For more information about configuring the IMS mobile service registry, see [“IMS mobile services and service registry” on page 246](#).

JCL sample files are now in a new location.

Table 16. Sample JCL locations

Location in V2	Location in V3
<code><installation_path>/jcl/baqstrt.jcl</code>	<code><hlq>.SBAQSAMP(BAQSTRT)</code>
<code><installation_path>/jcl/baqs123.jcl</code>	<code><hlq>.SBAQSAMP(BAQS123)</code>
<code><installation_path>/jcl/baqzcbt.jcl</code>	<code><hlq>.SBAQSAMP(BAQZCBT)</code>
<code><installation_path>/jcl/baqls2js.jcl</code>	<code><hlq>.SBAQSAMP(BAQLS2JS)</code>
<code><installation_path>/jcl/baqjs2ls.jcl</code>	<code><hlq>.SBAQSAMP(BAQJS2LS)</code>

Error responses are returned in JSON format by default.

The default value of the `useJsonErrorResponses` attribute on the `zosconnect_zosConnectManager` element in the `server.xml` configuration file is changed to true.

The WOLA service provider returns more HTTP status codes.

The WOLA service provider no longer uses only the HTTP status code 500 Internal Server Error for all error cases. Different HTTP status codes are now used to help distinguish server errors from errors caused by bad input data. The `useGenericError` attribute of the `zosconnect_localAdaptersConnectService` element can be set to `true` to maintain compatibility with previous versions of z/OS Connect EE.

Chapter 6. Performance considerations

As you configure your z/OS Connect EE server and create APIs and services, consider the information in these topics to help you get the best performance.

For information about improving performance in Liberty, see [Tuning Liberty in the IBM WebSphere Application Server for z/OS Liberty](#) documentation.

Performance best practices for a z/OS Connect EE production system

These practices are aimed at system administrators monitoring production systems.

Minimum levels of z/OS Connect EE

Use the following minimum levels of z/OS Connect EE to avoid performance issues:

- If you use clientauth TLS certificate validation, use z/OS Connect EE V3.0.6.
- If you use the authorization interceptor, use z/OS Connect EE V3.0.7.
- If you use client certificate authentication, use z/OS Connect EE V3.0.34

Your `server.xml` file

Polling for updates to the `server.xml` file, and running with debug enabled impacts performance. Check the following elements in your `server.xml` configuration file:

- Disable polling. Depending on your polling rate, polling can use high levels of CPU. In a production system, polling should be disabled. If you need to update your `server.xml` file in a production environment, use the **modify refresh** command. For more information, see [“Configuration updates on demand” on page 300](#).

By default, Liberty makes a poll request every 500 ms. To disable continuous polling, include the following lines in the `server.xml` configuration file:

```
<!-- applicationMonitor is not applicable for z/OS Connect EE servers -->
<applicationMonitor updateTrigger="disabled" dropinsEnabled="false"/>
```

- Ensure that trace is disabled. Check that all logging `traceSpecification` options are disabled. Any form of tracing has a considerable impact on performance. For more information, see [“Enabling trace in z/OS Connect EE server” on page 715](#).

Asynchronous TCP/IP sockets I/O for Liberty (AsyncIO)

By default, the z/OS Connect EE server uses a Java implementation to handle incoming TCPIP requests. To obtain improvements in performance and scalability in many environments, you can enable the Asynchronous TCP/IP sockets I/O for Liberty (AsyncIO) on z/OS service to take advantage of native z/OS services.

You must have the Angel process enabled, and issue a RACF (or equivalent) command to permit your z/OS Connect EE server to use the authorized AsyncIO service (ZOSAIO). For more information, see [“Configuring the Liberty Angel process and z/OS authorized services” on page 465](#).

When enabled, the following messages are written to the z/OS Connect EE messages log:

CWWKB0103I: Authorized service group ZOSAIO is available.
CWWK00229I: Native Asynchronous I/O support for z/OS has been activated.

Authenticating with the z/OS Connect EE server

By default, z/OS Connect EE uses client certificate authentication and attempts to map the provided client certificate to a user ID. If basic authentication is required and client certificate authentication is not required by any requests to this server, the default authentication method can be overridden by configuring `<webAppSecurity overrideHttpAuthMethod="BASIC" />` in the `server.xml` file.

Third-party authentication requests are not affected by this configuration setting as these take precedence over the basic authentication and client certificate authentication.

For more information, see [“API provider authentication and identification” on page 349](#) and [“API requester basic authentication to z/OS Connect EE” on page 407](#)

Connections

Establishing connections between components, for example, a client and a z/OS Connect EE server, can incur a cost. However, with persistent sessions, subsequent requests on the same connection are much cheaper because persistent sessions between a client and server reduce the need for regular handshakes.

- Most clients these days support HTTP/HTTPS 1.1 where persistent sessions (also known as *keepAlive*) are enabled by default.
- SSL handshakes can be costly but subsequent requests (assuming persistent sessions are being used) should be a lot faster. The strengths of the ciphers chosen have a marginal effect on performance.
- Establishing a connection to a System of Record (SoR), such as an IPIC connection to CICS, or a TCP/IP connection to IMS, can be costly the first time, but subsequent requests are faster on an established connection.
- The IBM MQ service provider uses a pool of JMS connections to connect to queue managers. If the pool isn't correctly sized for the number of received requests, this can affect throughput. For more information, see [Configuring connection pooling for JMS connections](#) in the *WebSphere Application Server Liberty* documentation.

Service Classes and Report Classes

Everything that is managed by z/OS Workload Manager has a service class and optionally, a report class. Service classes relate to the process of managing the work, and report classes relate to the process of collecting data about the work.

A service class is a group of work with similar performance goals, resource requirements, and business importance. z/OS Workload Manager manages each group of work according to the performance goal assigned to the service class, and the business importance assigned to that performance goal.

A report class is a group or item of work for which z/OS Workload Manager reports data. By default, data is reported as a total for each service class, so report classes are needed if you want to distinguish between different items within a service class.

Check the z/OS Workload Manager service classes of your applications that run on your LPAR including the service class that is used by z/OS Connect EE. For the best results, place work of the same type, with the same goals and importance, into the same service class wherever possible. For example, consider whether using the same service class for z/OS Connect EE as your CICS system or IMS region is appropriate for your environment.

Hardware processors

When monitoring performance, if your LPAR is sharing CPs with other LPARs, you might find your CPU usage varies. If you are looking for less variation in CPU usage, consider using dedicated CPs.

Even with dedicated CPs, the L3 and L4 memory buffer caches are typically shared with other CPs that are used by other LPARs. This can lead to CPU variation because those caches can have their data invalidated by CPs that are used by the other LPARs.

A large proportion of z/OS Connect EE is written in Java, so it benefits from running on zIIPs. Depending on the balance of GCPs and zIIPs and other products running in parallel to z/OS Connect EE, you might find improved performance by using zIIPs to prevent possible contention on GCPs with products that are not zIIP-eligible. Using hardware cryptography to encrypt and decrypt payloads reduces the amount of GCP usage. The savings depend on the strength of the ciphers chosen. Stronger ciphers use marginally more processing.

Monitoring

To monitor the performance of a running system, IBM provides a number of different facilities, including:

- IBM Health Center - a diagnostic tool for monitoring the status of a running Java (or Node.js) application such as z/OS Connect EE. The Health Center uses a small amount of processor time and memory (approximately 2% CPU), and can open some log and trace files for analysis in addition to providing recommendations to improve the performance.
- IBM OMEGAMON for JVM on z/OS provides resource level monitoring for all Java virtual machines (JVMs) on z/OS. It can quickly raise alerts to Java-related performance issues that allow corrective action to be taken on the root cause before problems affect business processes.
- System management facility (SMF) collects and records system and job-related information that an installation can use for many purposes, including monitoring the system resources that z/OS Connect EE uses.
- RMF (Resource Measurement Facility) gathers SMF data and provides reports about z/OS resource usage.

Understand your workload

Workloads are not always constant. Some have peaks and troughs, perhaps related to the time of day, or the period within a month. In general, the goal of performance tuning is to increase throughput, reduce response times, and increase the capacity for concurrent requests, all balanced against costs.

Monitor your workloads throughout the day by gathering performance measurements at different times to learn where your bottlenecks are. If required, repeat for different times of the month.

Lightweight Third-Party Authentication (LTPA) tokens

LTPA is a proprietary authentication technology that is used by Liberty z/OS and other IBM products. When multiple Liberty z/OS servers are configured to use LTPA, it is possible to enable SSO so that clients can reuse their login to access all the servers.

When LTPA is used, a token is created that contains the user information, an expiration time, and the signature of an LTPA key. The LTPA token passes between the client application and the Liberty server as a cookie when SSO is enabled. z/OS Connect EE supports the use of LTPA for authentication and SSO. However, it is more common to use an open authentication token such as a JWT.



Warning: The default expiration time of an LTPA token is 2 hours. If the user is then re-authenticated, a new LTPA token is created. If you use the same default expiration time for multiple servers that were all started at the same time, the LTPA tokens will all expire at the same time, possibly causing a degradation in performance as the user is re-authenticated and a new token created. Consider setting expiration times so that when a token expires, this re-authentication process occurs at a suitable time for your environment. Also, it is not advisable to set a long expiry time as it could cause a security exposure. For more information, see [Configuring LTPA in Liberty](#)

Java and JIT optimization

Most of z/OS Connect EE is written in Java running in its own JVM. A running JVM must perform several critical functions. The most significant of these functions are JIT compilation and garbage collection (GC). JIT and GC can use non-trivial amounts of CPU in the JVM. Therefore, before taking any performance measurements it is important to warm up the JIT so it is fully optimized. When the JIT is fully optimized

ensure GCs are not taken too frequently else CPU is spent purely on taking GCs rather than running your workloads.

Before taking performance measurements, it is worth understanding how the JIT compiler works so that you take your measurements at the optimum time.

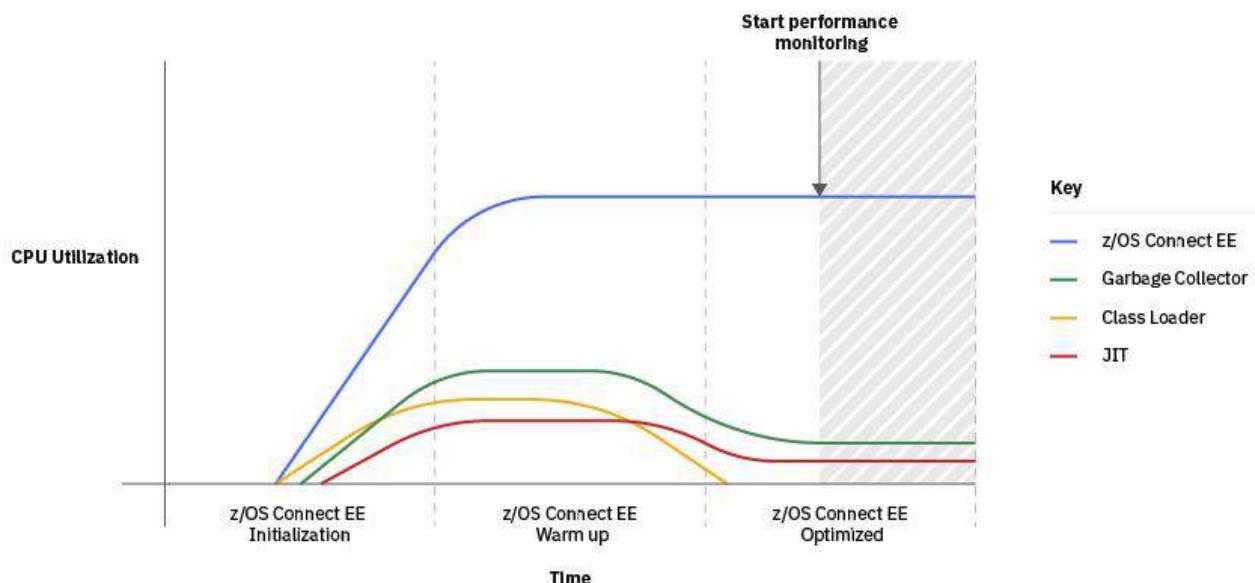
Java programs, such as z/OS Connect EE, consist of classes, which contain Java bytecode that is platform-neutral, meaning that it is not specific to any hardware or operating system platform. At run time, the Java virtual machine (JVM) compiles Java bytecode into IBM z/Architecture® instructions, using the just-in-time compiler (JIT) component.

Producing highly optimized z/Architecture instructions from Java bytecode requires processor time and memory. If all Java methods were compiled to the most aggressive level of optimization on first execution, this process results in long application initialization times, along with wasting significant quantities of CPU time optimizing methods that are used only during startup.

To provide a balance between application startup times and long-term performance, the JIT compiler uses an iterative process to optimize the bytecode. The JIT compiler maintains a count of the number of times each Java method is called. When the call count of a method exceeds a JIT recompilation threshold, the JIT recompiles the method to a more aggressive level of optimization and resets the method invocation count. This process is repeated until the maximum optimization level is reached. Therefore, often-used methods are compiled soon after the JVM starts, and less-used methods are compiled much later or not at all. The JIT compilation threshold helps the JVM to start quickly and still have good long-term performance.

This process of progressively optimizing Java methods leads to a change over time in the amount of CPU consumed by otherwise identical transactions. The first time a transaction is executed in Java, the z/Architecture instructions that are produced by the JIT compiler are at a low optimization level, which results in a relatively high CPU cost to execute the Java methods. As more transactions are executed, the Java method invocation counts are increased. Therefore, the JIT recompiles a Java method to a more aggressive level of optimization. This greater level of optimization results in a Java method that requires less CPU to execute than before the recompilation took place. As a result, the CPU that is required to execute the transaction reduces. This process is repeated several times during the lifetime of the JVM.

When executing benchmarks that use JVMs, ensure that the JIT compiler has fully optimized the most important Java methods in the z/OS Connect EE server before starting to take CPU measurements. To minimize variability introduced by the JIT compiler, run the workload at a constant transaction rate for a period of time, known as the warm-up period. The warm-up period completes at the point at which the CPU cost per transaction ceases to show any improvements. After the workload is running in a steady-state for the warm-up period, it is assumed that the JIT compiler will not optimize the z/OS Connect EE server further, and CPU measurements can be taken.



Shutting down a z/OS Connect EE server (and thus the JVM it runs in) discards the JIT-compiled native code; therefore, the iterative process of optimization begins again when the JVM is restarted.

Performance problems?

If you suspect you are having performance problems, such as out of memory errors, high processor usage, system crash, hang, or poor throughput, obtain guidance on what information to collect from the MustGather document [Runtimes for Java Technology](#).

How payload size and transformation affect performance for the API provider

When data passes through a z/OS Connect EE server, the server needs to transform each byte in the payload between JSON and the language structure. Therefore, the larger the JSON payload, the more CPU processing time is needed. Although this data transformation is zIIP eligible, unnecessary CPU usage might slow down your overall performance.

When you create a service archive (.sar) file, you can use the API toolkit to reduce the size of your JSON payload by excluding unused fields or renaming fields, to improve performance. These features are not available with the build toolkit.

If your payload contains arrays, you can optimize the JSON payload by using OCCURS DEPENDING ON (ODO) in COBOL, and REFER in PL/I, or use array counters, or both. This method can reduce the number of array elements in the JSON payload.

Using ODO (COBOL) and REFER (PL/I) can also reduce the size of the language structure, which might further improve z/OS subsystem performance. For example, by reducing IMS message sizes.

The following sections describe these techniques in more detail:

- [“Exclude unused fields” on page 209](#)
- [“Rename fields” on page 212](#)
- [“Array sizes” on page 213](#)

Exclude unused fields

When creating z/OS Connect EE services using the API toolkit, fields that are not needed for your service should be excluded from the JSON payload by clearing the checkbox for those fields in the **Include** column in the **Service Interface Editor**. This method reduces the size of the JSON payload and makes the schemas more accurate as they do not describe properties that are not relevant.

Example 1

In this example, the request and response use the same copybook, but different fields. When the same copybook is used for both the request and the response, define separate request and response service interfaces if you plan to exclude or rename different fields in the request versus the response.

By default, all fields are included for both the request and the response. So for an HTTP PUT or POST, all fields in the request and response are subject to data transformation. For an HTTP GET, all fields in the response are subject to data transformation.

[Figure 45 on page 210](#) shows an HTTP PUT request that includes a 100 element array. The copybook of 30,376 bytes generates a JSON payload of 64,496 bytes.

Service Interface Editor

Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

Fields	Include	Interface Rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA	<input checked="" type="checkbox"/>	CARRTCP		STRUCT	56	1
CARRTCP	<input checked="" type="checkbox"/>	REQ_TXN_REQUEST_BLOCK_HEADER		CHAR	8	1
REQ_CAR_MODEL	<input checked="" type="checkbox"/>	REQ_CAR_MODEL		CHAR	8	9
REQ_CAR_MODEL_TYPE	<input checked="" type="checkbox"/>	REQ_CAR_MODEL_TYPE		CHAR	36	17
REQ_EXTRAS	<input checked="" type="checkbox"/>	REQ_MAX_NBR_OF_REC_REQUESTED		UINT	4	53
REQ_MAX_NBR_OF_REC_REQUESTED	<input checked="" type="checkbox"/>	RESP_TXN_RESPONSE_BLOCK_HEADER		STRUCT	5	57
RESP_CODE	<input checked="" type="checkbox"/>	RESP_CODE		CHAR	1	57
[!] RESP_NBR_OF_REC_RETURNED	<input checked="" type="checkbox"/>	RESP_NBR_OF_REC_RETURNED		UINT	4	58
CARMODEL_CAR_FINDER	<input checked="" type="checkbox"/>	CARMODEL_CAR_FINDER		STRUCT	30272	62
CARMODEL_INFO_TABLE	<input checked="" type="checkbox"/>	CARMODEL_INFO_TABLE		STRUCT	30272	62
[!] CARMODEL_INFO_DATA [1..100]	<input checked="" type="checkbox"/>	CARMODEL_INFO_DATA		ARRAY	30200	62
CARMODEL DEALER_LOCATION	<input checked="" type="checkbox"/>	CARMODEL DEALER_LOCATION		STRUCT	62	
CARMODEL LOC_COUNTRY	<input checked="" type="checkbox"/>	CARMODEL LOC_COUNTRY		CHAR	2	
CARMODEL LOC_ADDRESS	<input checked="" type="checkbox"/>	CARMODEL LOC_ADDRESS		CHAR	60	
CARMODEL_1_REF	<input checked="" type="checkbox"/>	CARMODEL_1_REF		CHAR	12	
CARMODEL_2_MODEL	<input checked="" type="checkbox"/>	CARMODEL_2_MODEL		CHAR	12	
CARMODEL_3_COST	<input checked="" type="checkbox"/>	CARMODEL_3_COST		CHAR	12	
CARMODEL_4 LEATHERSEATS	<input checked="" type="checkbox"/>	CARMODEL_4 LEATHERSEATS		CHAR	12	
CARMODEL_5 SUNROOF	<input checked="" type="checkbox"/>	CARMODEL_5 SUNROOF		CHAR	12	
CARMODEL_6 REARCAMERA	<input checked="" type="checkbox"/>	CARMODEL_6 REARCAMERA		CHAR	12	
CARMODEL_7 SATNAV	<input checked="" type="checkbox"/>	CARMODEL_7 SATNAV		CHAR	12	
CARMODEL_8 HEATEDSEATS	<input checked="" type="checkbox"/>	CARMODEL_8 HEATEDSEATS		CHAR	12	
CARMODEL_9 BLUETOOTH	<input checked="" type="checkbox"/>	CARMODEL_9 BLUETOOTH		CHAR	12	
CARMODEL_10 PARKINGSENSORS	<input checked="" type="checkbox"/>	CARMODEL_10 PARKINGSENSORS		CHAR	12	
CARMODEL_11 ALLOYS	<input checked="" type="checkbox"/>	CARMODEL_11 ALLOYS		CHAR	12	
CARMODEL_12 REMOTESTART	<input checked="" type="checkbox"/>	CARMODEL_12 REMOTESTART		CHAR	12	
CARMODEL_13 BLINDSPOT	<input checked="" type="checkbox"/>	CARMODEL_13 BLINDSPOT		CHAR	12	
CARMODEL_14 THIRDROWSEATING	<input checked="" type="checkbox"/>	CARMODEL_14 THIRDROWSEATING		CHAR	12	
CARMODEL_15 HEATEDMIRRORS	<input checked="" type="checkbox"/>	CARMODEL_15 HEATEDMIRRORS		CHAR	12	
CARMODEL_16 FRONTCAMERA	<input checked="" type="checkbox"/>	CARMODEL_16 FRONTCAMERA		CHAR	12	
CARMODEL_17 HEATEDWINDSCREEN	<input checked="" type="checkbox"/>	CARMODEL_17 HEATEDWINDSCREEN		CHAR	12	
CARMODEL_18 ALARM	<input checked="" type="checkbox"/>	CARMODEL_18 ALARM		CHAR	12	
CARMODEL_19 TRACKER	<input checked="" type="checkbox"/>	CARMODEL_19 TRACKER		CHAR	12	
CARMODEL_20 POWERSEATS	<input checked="" type="checkbox"/>	CARMODEL_20 POWERSEATS		CHAR	12	
FILL_0	<input checked="" type="checkbox"/>	FILL_0		CHAR	72	

Figure 45. A request that includes all fields.

Transforming unnecessary fields in this JSON payload is inefficient and degrades performance. Therefore, clear the **Include** checkbox of fields that are not needed for the request, and similarly for the response, as shown in Figure 46 on page 211. In this example, the request does not require all the response fields to be checked, as the COBOL application does not require these fields as input to the program. This reduces the number of bytes from 30,376 to 112, and reduces the JSON payload size from approximately 64,496 to 122.

Service Interface Editor

Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

Fields	Include	Interface Rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA	<input checked="" type="checkbox"/>	CARRCP				
CARRCP	<input checked="" type="checkbox"/>	REQ_TXN_REQUEST_BLOCK_HEADER		STRUCT	56	1
REQ_CAR_MODEL	<input checked="" type="checkbox"/>	REQ_CAR_MODEL		CHAR	8	1
REQ_CAR_MODEL_TYPE	<input checked="" type="checkbox"/>	REQ_CAR_MODEL_TYPE		CHAR	8	9
REQ_EXTRAS	<input checked="" type="checkbox"/>	REQ_EXTRAS		CHAR	36	17
REQ_MAX_NBR_OF_REC_REQUESTED	<input checked="" type="checkbox"/>	REQ_MAX_NBR_OF_REC_REQUESTED		UINT	4	53
RESP_TXN_RESPONSE_BLOCK_HEADER	<input checked="" type="checkbox"/>	RESP_TXN_RESPONSE_BLOCK_HEADER		STRUCT	5	57
RESP_CODE	<input type="checkbox"/>	RESP_CODE		CHAR	1	57
[!] RESP_NBR_OF_REC_RETURNED	<input checked="" type="checkbox"/>	RESP_NBR_OF_REC_RETURNED		UINT	4	58
CARMODEL_CAR_FINDER	<input checked="" type="checkbox"/>	CARMODEL_CAR_FINDER		STRUCT	30272	62
CARMODEL_INFO_TABLE	<input type="checkbox"/>	CARMODEL_INFO_TABLE		STRUCT	30272	62
[!] CARMODEL_INFO_DATA [1..100]	<input type="checkbox"/>	CARMODEL_INFO_DATA		ARRAY	30200	62
CARMODEL DEALER LOCATION	<input type="checkbox"/>	CARMODEL DEALER LOCATION		STRUCT	62	
CARMODEL LOC COUNTRY	<input type="checkbox"/>	CARMODEL LOC COUNTRY		CHAR	2	
CARMODEL LOC ADDRESS	<input type="checkbox"/>	CARMODEL LOC ADDRESS		CHAR	60	
CARMODEL_1_REF	<input type="checkbox"/>	CARMODEL_1_REF		CHAR	12	
CARMODEL_2_MODEL	<input type="checkbox"/>	CARMODEL_2_MODEL		CHAR	12	
CARMODEL_3_COST	<input type="checkbox"/>	CARMODEL_3_COST		CHAR	12	
CARMODEL_4 LEATHERSEATS	<input type="checkbox"/>	CARMODEL_4 LEATHERSEATS		CHAR	12	
CARMODEL_5 SUNROOF	<input type="checkbox"/>	CARMODEL_5 SUNROOF		CHAR	12	
CARMODEL_6 REARCAMERA	<input type="checkbox"/>	CARMODEL_6 REARCAMERA		CHAR	12	
CARMODEL_7 SATNAV	<input type="checkbox"/>	CARMODEL_7 SATNAV		CHAR	12	
CARMODEL_8 HEATEDSEATS	<input type="checkbox"/>	CARMODEL_8 HEATEDSEATS		CHAR	12	
CARMODEL_9 BLUETOOTH	<input type="checkbox"/>	CARMODEL_9 BLUETOOTH		CHAR	12	
CARMODEL_10 PARKINGSENSORS	<input type="checkbox"/>	CARMODEL_10 PARKINGSENSORS		CHAR	12	
CARMODEL_11 ALLOYS	<input type="checkbox"/>	CARMODEL_11 ALLOYS		CHAR	12	
CARMODEL_12 REMOTESTART	<input type="checkbox"/>	CARMODEL_12 REMOTESTART		CHAR	12	
CARMODEL_13 BLINDSPOT	<input type="checkbox"/>	CARMODEL_13 BLINDSPOT		CHAR	12	
CARMODEL_14 THIRDROWSEATING	<input type="checkbox"/>	CARMODEL_14 THIRDROWSEATING		CHAR	12	
CARMODEL_15 HEATEDMIRRORS	<input type="checkbox"/>	CARMODEL_15 HEATEDMIRRORS		CHAR	12	
CARMODEL_16 FRONTCAMERA	<input type="checkbox"/>	CARMODEL_16 FRONTCAMERA		CHAR	12	
CARMODEL_17 HEATEDWINDSCREEN	<input type="checkbox"/>	CARMODEL_17 HEATEDWINDSCREEN		CHAR	12	
CARMODEL_18 ALARM	<input type="checkbox"/>	CARMODEL_18 ALARM		CHAR	12	
CARMODEL_19 TRACKER	<input type="checkbox"/>	CARMODEL_19 TRACKER		CHAR	12	
CARMODEL_20 POWERSEATS	<input type="checkbox"/>	CARMODEL_20 POWERSEATS		CHAR	12	
FILL_0	<input type="checkbox"/>	FILL_0		CHAR	72	

Figure 46. A request that excludes unnecessary fields.

Example 2

The request in Figure 47 on page 212 is asking for details about a user's car insurance but is not interested in the user's holiday insurance or building insurance. Therefore, by clearing the checkboxes for the holiday insurance and building insurance, the size of the JSON payload is reduced considerably.

Service Interface Editor

Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

Fields	Include	Interface Rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA	<input type="checkbox"/>					
INSCP	<input checked="" type="checkbox"/>					
REQ_TXN_REQUEST_BLOCK_HEADER	<input type="checkbox"/>	REQ_TXN_REQUEST_BLOCK_HEADER		STRUCT	86	1
REQ_CUSTOMER_NAME	<input type="checkbox"/>	REQ_CUSTOMER_NAME		CHAR	30	1
REQ_CUSTOMER_ACCOUNT_NUMBE	<input type="checkbox"/>	REQ_CUSTOMER_ACCOUNT_NUMBE		CHAR	16	31
REQ_POLICIES	<input type="checkbox"/>	REQ_POLICIES		CHAR	36	47
REQ_MAX_NBR_OF_REC_REQUESTED	<input type="checkbox"/>	REQ_MAX_NBR_OF_REC_REQUESTED		UINT	4	83
RESP_TXN_RESPONSE_BLOCK_HEADER	<input checked="" type="checkbox"/>	RESP_TXN_RESPONSE_BLOCK_HEAD		STRUCT	5	87
RESP_CODE	<input checked="" type="checkbox"/>	RESP_CODE		CHAR	1	87
[#] RESP_NBR_OF_REC_RETURNED	<input checked="" type="checkbox"/>	RESP_NBR_OF_REC_RETURNED		UINT	4	88
HOLIDAY_INSURANCE	<input type="checkbox"/>	HOLIDAY_INSURANCE		STRUCT	6732	92
HOLIDAY_INSURANCE_DETAILS	<input type="checkbox"/>	HOLIDAY_INSURANCE_DETAILS		STRUCT	6732	92
[#] HOLIDAY_CLAIMS_DATA [1..30]	<input type="checkbox"/>	HOLIDAY_CLAIMS_DATA		ARRAY	6660	92
HOLIDAY_CLAIM_DATE	<input type="checkbox"/>	HOLIDAY_CLAIM_DATE		CHAR	18	
HOLIDAY_CLAIM_DETAILS	<input type="checkbox"/>	HOLIDAY_CLAIM_DETAILS		CHAR	200	
HOLIDAY_CLAIM_PAID_OUT	<input type="checkbox"/>	HOLIDAY_CLAIM_PAID_OUT		UINT	4	
FILL_0	<input type="checkbox"/>	FILL_0		CHAR	72	
CAR_INSURANCE	<input checked="" type="checkbox"/>	CAR_INSURANCE		STRUCT	6732	
CAR_INSURANCE_DETAILS	<input checked="" type="checkbox"/>	CAR_INSURANCE_DETAILS		STRUCT	6732	
[#] CAR CLAIMS DATA [1..30]	<input checked="" type="checkbox"/>	CAR CLAIMS DATA		ARRAY	6660	
CAR CLAIM_DATE	<input checked="" type="checkbox"/>	CAR CLAIM_DATE		CHAR	18	
CAR CLAIM DETAILS	<input checked="" type="checkbox"/>	CAR CLAIM DETAILS		CHAR	200	
CAR CLAIM_PAID_OUT	<input checked="" type="checkbox"/>	CAR CLAIM_PAID_OUT		UINT	4	
FILL_1	<input type="checkbox"/>	FILL_1		CHAR	72	
BUILDING_INSURANCE	<input type="checkbox"/>	BUILDING_INSURANCE		STRUCT	6732	
BUILDING_INSURANCE_DETAILS	<input type="checkbox"/>	BUILDING_INSURANCE_DETAILS		STRUCT	6732	
[#] BUILDING CLAIMS DATA [1..30]	<input type="checkbox"/>	BUILDING CLAIMS DATA		ARRAY	6660	
BUILDING CLAIM_DATE	<input type="checkbox"/>	BUILDING CLAIM_DATE		CHAR	18	
BUILDING CLAIM DETAILS	<input type="checkbox"/>	BUILDING CLAIM DETAILS		CHAR	200	
BUILDING CLAIM_PAID_OUT	<input type="checkbox"/>	BUILDING CLAIM_PAID_OUT		UINT	4	
FILL_2	<input type="checkbox"/>	FILL_2		CHAR	72	

Figure 47. A request that excludes holiday and building insurance fields.

Rename fields

When a copybook is imported in the API toolkit, the field names that are shown in the **Interface Rename** column in the **Service Interface Editor** default to the same name as the original field name. For example, the following JSON snippet in Figure 48 on page 212 is generated from part of the COBOL copybook imported in Figure 48 on page 212.

```
{
  "CAR CLAIMS DATA": {
    "CAR CLAIM_DATE": "...",
    "CAR CLAIM DETAILS": "...",
    "CAR CLAIM_PAID_OUT": "..."
  }
}
```

Figure 48. A JSON schema with default field names

These field names are not good practice and an API developer fluent in JSON would expect to see something similar to Figure 49 on page 212.

```
{
  "carClaims": {
    "date": "...",
    "details": "...",
    "paidOut": "..."
  }
}
```

Figure 49. A JSON schema with renamed fields

To rename fields, use the *Interface Rename* feature of the API toolkit. Renaming these fields to names shorter than the default saves CPU processing cost by reducing the amount of JSON parsing or generation that is needed, and also provides field names that are more familiar to the JSON programmer.

Figure 50 on page 213 is an extract from a typical data structure imported from a COBOL copybook and shows how the fields are renamed in the **Interface Rename** column to shorter names that are more recognizable to an API developer. These renamed fields are used in the JSON payload, so the shorter the names, the less transformation is needed, resulting in improved performance.

Service Interface Editor							
Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.							
Fields		Include	Interface Rename	Default Field Value	Data Type	Field Length	Start Byte
CARMODEL DEALERS		<input checked="" type="checkbox"/>	Dealers		STRUCT	3976	121
CARMODEL CAR FINDER		<input checked="" type="checkbox"/>	Cars		STRUCT	3976	121
CARMODEL DEALER LOCATION		<input checked="" type="checkbox"/>	Dealer		STRUCT	118	121
CARMODEL LOCATION		<input checked="" type="checkbox"/>	Location		CHAR	2	121
CARMODEL LOC ADDRESS		<input checked="" type="checkbox"/>	Address		CHAR	90	123
CARMODEL COUNTRY		<input checked="" type="checkbox"/>	Country		CHAR	26	213
CARMODEL INFO TABLE		<input checked="" type="checkbox"/>	Info		STRUCT	3858	239
[%] CARMODEL_INFO_DATA [16..16]		<input checked="" type="checkbox"/>	Details		ARRAY	3840	239
	CARMODEL_1_REF	<input checked="" type="checkbox"/>	Ref		CHAR	12	
	CARMODEL_2_MODEL	<input checked="" type="checkbox"/>	Model		CHAR	12	
	CARMODEL_3_COST	<input checked="" type="checkbox"/>	Cost		CHAR	12	
	CARMODEL_4 LEATHERSEATS	<input checked="" type="checkbox"/>	LeatherSeats		CHAR	12	
	CARMODEL_5 SUNROOF	<input checked="" type="checkbox"/>	Sunroof		CHAR	12	
	CARMODEL_6 REARCAMERA	<input checked="" type="checkbox"/>	RearCamera		CHAR	12	
	CARMODEL_7 SATNAV	<input checked="" type="checkbox"/>	SetNav		CHAR	12	
	CARMODEL_8 HEATEDSEATS	<input checked="" type="checkbox"/>	HeatedSeats		CHAR	12	
	CARMODEL_9 BLUETOOTH	<input checked="" type="checkbox"/>	Bluetooth		CHAR	12	
	CARMODEL_10 PARKINGSENSORS	<input checked="" type="checkbox"/>	ParkingSensors		CHAR	12	
	CARMODEL_11 ALLOYS	<input checked="" type="checkbox"/>	Alloys		CHAR	12	
	CARMODEL_12 REMOTESTART	<input checked="" type="checkbox"/>	RemoteStart		CHAR	12	
	CARMODEL_13 BLINDSPOT	<input checked="" type="checkbox"/>	Blindspot		CHAR	12	
	CARMODEL_14 THIRDROWSEATING	<input checked="" type="checkbox"/>	ThirdRowSeating		CHAR	12	
	CARMODEL_15 HEATEDMIRRORS	<input checked="" type="checkbox"/>	HeatedMirrors		CHAR	12	
	CARMODEL_16 FRONTCAMERA	<input checked="" type="checkbox"/>	FrontCamera		CHAR	12	
	CARMODEL_17 HEATEDWINDSCREEN	<input checked="" type="checkbox"/>	HeatedWindscreen		CHAR	12	
	CARMODEL_18 ALARM	<input checked="" type="checkbox"/>	Alarm		CHAR	12	
	CARMODEL_19 TRACKER	<input checked="" type="checkbox"/>	Tracker		CHAR	12	
	CARMODEL_20 POWERSEATS	<input checked="" type="checkbox"/>	PowerSeats		CHAR	12	

Figure 50. A request with renamed fields.

This saving is especially noticeable when large arrays are used in the data structure, as shown in the Figure 50 on page 213.

Array sizes

The size of the JSON payload can also be reduced by using array sizes determined by using the following techniques:

- OCCURS DEPENDING ON (ODO) for COBOL copybooks, or REFER for PL/I includes.
- Array counters.

ODO or REFER

In Figure 51 on page 214, CARMODEL_INFO_DATA has a maximum of 100 elements in the array. However, a user might want to request only the first 10 cars listed in a database, rather than have the maximum set of 100 cars returned. Therefore, by specifying 10 in the request, only 10 elements of the array will be included in the JSON response payload, instead of 100 elements, considerably reducing the size of the payload to be transformed.

Service Interface Editor

Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

Search: [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] []

Fields	Include	Interface Rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA		CARRRCP				
CARRRCP						
REQ_TXN_REQUEST_BLOCK_HEADER	<input type="checkbox"/>	REQ_TXN_REQUEST_BLOCK_HEADER		STRUCT	56	1
REQ_CAR_MODEL	<input type="checkbox"/>	REQ_CAR_MODEL		CHAR	8	1
REQ_CAR_MODEL_TYPE	<input type="checkbox"/>	REQ_CAR_MODEL_TYPE		CHAR	8	9
REQ_EXTRAS	<input type="checkbox"/>	REQ_EXTRAS		CHAR	36	17
REQ_MAX_NBR_OF_REC_REQUESTED	<input type="checkbox"/>	REQ_MAX_NBR_OF_REC_REQUESTED		UINT	4	53
RESP_TXN_RESPONSE_BLOCK_HEADER	<input checked="" type="checkbox"/>	RESP_TXN_RESPONSE_BLOCK_HEAD		STRUCT	5	57
RESP_CODE	<input checked="" type="checkbox"/>	RESP_CODE		CHAR	1	57
[#] RESP_NBR_OF_REC_RETURNED	<input checked="" type="checkbox"/>	RESP_NBR_OF_REC_RETURNED		UINT	4	58
CARMODEL_CAR_FINDER	<input checked="" type="checkbox"/>	CARMODEL_CAR_FINDER		STRUCT	30272	62
CARMODEL_INFO_TABLE	<input checked="" type="checkbox"/>	CARMODEL_INFO_TABLE		STRUCT	30272	62
[8] CARMODEL_INFO_DATA [1..100]	<input checked="" type="checkbox"/>	CARMODEL_INFO_DATA		ARRAY	30200	62
CARMODEL DEALER LOCATION	<input checked="" type="checkbox"/>	CARMODEL DEALER LOCATION				
CARMODEL LOC COUNTRY	<input checked="" type="checkbox"/>	CARMODEL LOC COUNTRY				
CARMODEL LOC ADDRESS	<input checked="" type="checkbox"/>	CARMODEL LOC ADDRESS				
CARMODEL 1 REF	<input checked="" type="checkbox"/>	CARMODEL 1 REF				
CARMODEL 2 MODEL	<input checked="" type="checkbox"/>	CARMODEL 2 MODEL		CHAR	12	
CARMODEL 3 COST	<input checked="" type="checkbox"/>	CARMODEL 3 COST		CHAR	12	
CARMODEL 4 LEATHERSEATS	<input checked="" type="checkbox"/>	CARMODEL 4 LEATHERSEATS		CHAR	12	
CARMODEL 5 SUNROOF	<input checked="" type="checkbox"/>	CARMODEL 5 SUNROOF		CHAR	12	
CARMODEL 6 REARCAMERA	<input checked="" type="checkbox"/>	CARMODEL 6 REARCAMERA		CHAR	12	
CARMODEL 7 SATNAV	<input checked="" type="checkbox"/>	CARMODEL 7 SATNAV		CHAR	12	
CARMODEL 8 HEATEDSEATS	<input checked="" type="checkbox"/>	CARMODEL 8 HEATEDSEATS		CHAR	12	
CARMODEL 9 BLUETOOTH	<input checked="" type="checkbox"/>	CARMODEL 9 BLUETOOTH		CHAR	12	
CARMODEL 10 PARKINGSENSORS	<input checked="" type="checkbox"/>	CARMODEL 10 PARKINGSENSORS		CHAR	12	
CARMODEL 11 ALLOYS	<input checked="" type="checkbox"/>	CARMODEL 11 ALLOYS		CHAR	12	
CARMODEL 12 REMOTESTART	<input checked="" type="checkbox"/>	CARMODEL 12 REMOTESTART		CHAR	12	
CARMODEL 13 BLINDSPOT	<input checked="" type="checkbox"/>	CARMODEL 13 BLINDSPOT		CHAR	12	
CARMODEL 14 THIRDROWSEATING	<input checked="" type="checkbox"/>	CARMODEL 14 THIRDROWSEATING		CHAR	12	
CARMODEL 15 HEATEDMIRRORS	<input checked="" type="checkbox"/>	CARMODEL 15 HEATEDMIRRORS		CHAR	12	
CARMODEL 16 FRONTCAMERA	<input checked="" type="checkbox"/>	CARMODEL 16 FRONTCAMERA		CHAR	12	
CARMODEL 17 HEATEDWINDSCREEN	<input checked="" type="checkbox"/>	CARMODEL 17 HEATEDWINDSCREEN		CHAR	12	
CARMODEL 18 ALARM	<input checked="" type="checkbox"/>	CARMODEL 18 ALARM		CHAR	12	
CARMODEL 19 TRACKER	<input checked="" type="checkbox"/>	CARMODEL 19 TRACKER		CHAR	12	
CARMODEL 20 POWERSEATS	<input checked="" type="checkbox"/>	CARMODEL 20 POWERSEATS		CHAR	12	
FILL_0	<input type="checkbox"/>	FILL_0				

CARRRCP.CARMODEL_CARMODEL_INFO_TABLE.CARMODEL_INFO_DATA ARRAY
OCCURS 1 TO 100 TIMES
DEPENDS ON CARRRCP.RESP_TXN_RESPONSE_BLOCK_HEADER.RESP_NBR_OF_REC_RETURNED

Figure 51. A copybook with an array and ODO.

Array counters

You can use array counters in two ways,

- To optimize the JSON payload for fixed-length arrays.
- To optimize the JSON payload for nested variable-length arrays (ODO or REFER).

Array counters for a fixed-length or variable-length array.

Consider the following snippet from a fixed array copybook definition.

```

15 CARMODEL-INFO-TABLE.
 20 CARMODEL-INFO-DATA OCCURS 100 TIMES.
   25 CARMODEL-1-REF          PIC X(12).
   25 CARMODEL-2-MODEL        PIC X(12).
   25 CARMODEL-3-COST         PIC X(12).
   25 CARMODEL-4-LEATHERSEATS PIC X(12).
   ::                      ::          ::
```

After you import this copybook into the API toolkit, depending on the COBOL (or PL/I) application, it might be possible to specify an array counter to limit the number of array elements.

Service Interface Editor

Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

Fields	Include	Interface Rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA						
CAR100CP		CAR100CP				
REQ_TXN_REQUEST_BLOCK_HEADER	<input checked="" type="checkbox"/>	REQ_TXN_REQUEST_BLOCK_HEADER		STRUCT	120	1
REQ_CAR_MODEL	<input checked="" type="checkbox"/>	REQ_CAR_MODEL		CHAR	8	1
REQ_EXTRAS	<input checked="" type="checkbox"/>	REQ_EXTRAS		CHAR	36	9
REQ_MAX_NBR_OF_REC_REQUESTED	<input checked="" type="checkbox"/>	REQ_MAX_NBR_OF_REC_REQUESTED		UINT	4	45
REQ_CODE	<input checked="" type="checkbox"/>	REQ_CODE		CHAR	1	49
REQ_NBR_OF_REC_RETURNED	<input checked="" type="checkbox"/>	REQ_NBR_OF_REC_RETURNED		UINT	4	50
REQ_TXN_INPUT_DATA_LEN	<input checked="" type="checkbox"/>	REQ_TXN_INPUT_DATA_LEN		UINT	4	54
REQ_TXN_OUTPUT_DATA_LEN	<input checked="" type="checkbox"/>	REQ_TXN_OUTPUT_DATA_LEN		UINT	4	58
REQ_TXN_DATA_LEN	<input checked="" type="checkbox"/>	REQ_TXN_DATA_LEN		UINT	4	62
FILL_0	<input checked="" type="checkbox"/>	FILL_0		CHAR	8	66
REQ_MAJ_RET_CODE	<input checked="" type="checkbox"/>	REQ_MAJ_RET_CODE		SHORT	2	74
REQ_MNR_RET_CODE	<input checked="" type="checkbox"/>	REQ_MNR_RET_CODE		SHORT	2	76
REQ_RET_MSG	<input checked="" type="checkbox"/>	REQ_RET_MSG		STRUCT	43	78
CARMODEL DEALERS	<input checked="" type="checkbox"/>	CARMODEL DEALERS		STRUCT	30000	121
CARMODEL CAR FINDER	<input checked="" type="checkbox"/>	CARMODEL CAR FINDER		STRUCT	30000	121
CARMODEL DEALER LOCATION	<input checked="" type="checkbox"/>	CARMODEL DEALER LOCATION		STRUCT	118	121
CARMODEL LOCATION	<input checked="" type="checkbox"/>	CARMODEL LOCATION		CHAR	2	121
CARMODEL LOC ADDRESS	<input checked="" type="checkbox"/>	CARMODEL LOC ADDRESS		CHAR	90	123
CARMODEL COUNTRY	<input checked="" type="checkbox"/>	CARMODEL COUNTRY		CHAR	26	213
CARMODEL INFO TABLE	<input checked="" type="checkbox"/>	CARMODEL INFO TABLE		STRUCT	29882	239
CARMODEL INFO DATA [0..100]	<input checked="" type="checkbox"/>	CARMODEL INFO DATA		ARRAY	24000	239
CARMODEL 1 REF	<input checked="" type="checkbox"/>	CARMODEL 1 REF				
CARMODEL 2 MODEL	<input checked="" type="checkbox"/>	CARMODEL 2 MODEL				
CARMODEL 3 COST	<input checked="" type="checkbox"/>	CARMODEL 3 COST				
CARMODEL 4 LEATHERSEATS	<input checked="" type="checkbox"/>	CARMODEL 4 LEATHERSEATS				
CARMODEL 5 SUNROOF	<input checked="" type="checkbox"/>	CARMODEL 5 SUNROOF		CHAR	12	
CARMODEL 6 REARCAMERA	<input checked="" type="checkbox"/>	CARMODEL 6 REARCAMERA		CHAR	12	
CARMODEL 7 SATNAV	<input checked="" type="checkbox"/>	CARMODEL 7 SATNAV		CHAR	12	
CARMODEL 8 HEATEDSEATS	<input checked="" type="checkbox"/>	CARMODEL 8 HEATEDSEATS		CHAR	12	
		COUNTED BY CAR100CP.REQ_TXN_REQUEST_BLOCK_HEADER.REQ_MAX_NBR_OF_REC_REQUESTED				
		OCCURS 0 TO 100 TIMES				
		COUNTED BY CAR100CP.REQ_TXN_REQUEST_BLOCK_HEADER.REQ_MAX_NBR_OF_REC_REQUESTED				

Figure 52. Specifying an array counter to limit the number of array elements.

In this example, the number of entries in the array field CARMODEL_INFO_DATA, which is an ODO/REFER array subject, is controlled by array counter REQ_MAX_NBR_OF_REC_REQUESTED. Specifying an array counter allows you to set a minimum number of entries so that the array appears variable-length to an API client.

Array counters with nested ODO or REFER.

Nested ODO or REFER statements can generate very large JSON payloads. The use of array counters can reduce the size of these JSON payloads considerably, and improve overall performance.

The following example demonstrates the value of an array counter in the context of a nested ODO array.

The field CARMODEL-NBR-OF-COMMENTS is used to maintain the count of the ODO array CARMODEL-COMMENTS by using the arrays counter support. This allows each CARMODEL-INFO-DATA to have a different number of comments in the request or response JSON within the range of 1 to (REQ-MAX-NBR-OF-COMMENTS <= 20) times.

```

05 REQ-NBR-OF-REC-RETURNED PIC 9(09) COMP.
05 REQ-MAX-NBR-OF-COMMENTS PIC 9(09) COMP.

      ::          ::          ::

15 CARMODEL-INFO-TABLE.
  20 CARMODEL-INFO-DATA OCCURS 1 TO 100 TIMES
    DEPENDING ON REQ-NBR-OF-REC-RETURNED.
  25 CARMODEL-NBR-OF-COMMENTS PIC 9(09) COMP.
  25 CARMODEL-COMMENTS OCCURS 1 TO 20 TIMES
    DEPENDING ON REQ-MAX-NBR-OF-COMMENTS.
  30 CARMODEL-COMMENT-A.
  30 CARMODEL-COMMENT-B.
  30 CARMODEL-COMMENT-C.

```

As the arrays are nested, the size of the JSON payload can become very large. Therefore, the use of array counters can reduce the size of the JSON payload, and improve performance as less transformation is needed.

Figure 53 on page 216 shows the two arrays, with the second ODO highlighted, and shows the array counter that is assigned to it.

Service Interface Editor						
Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.						
Fields	Include	Interface Rename	Default Field Value	Data Type	Field Length	Start Byte
✓ CARODO2C		CARODO2C		STRUCT	124	1
REQ_TXN_REQUEST_BLOCK_HEADER		REQ_TXN_REQUEST_BLOCK_HEADER		CHAR	8	1
REQ_CAR_MODEL		REQ_CAR_MODEL		CHAR	36	9
REQ_EXTRAS		REQ_EXTRAS		UINT	4	45
[#] REQ_NBR_OF_REC_RETURNED		REQ_NBR_OF_REC_RETURNED		UINT	4	49
[#] REQ_MAX_NBR_OF_COMMENTS		REQ_MAX_NBR_OF_COMMENTS		UINT	4	53
REQ_CODE		REQ_CODE		CHAR	1	54
REQ_NBR_OF_COMMENTS		REQ_NBR_OF_COMMENTS		UINT	4	58
REQ_TXN_INPUT_DATA_LEN		REQ_TXN_INPUT_DATA_LEN		UINT	4	62
REQ_TXN_OUTPUT_DATA_LEN		REQ_TXN_OUTPUT_DATA_LEN		UINT	4	66
REQ_TXN_DATA_LEN		REQ_TXN_DATA_LEN		UINT	4	70
FILL_0		FILL_0		CHAR	8	78
REQ_MAJ_RET_CODE		REQ_MAJ_RET_CODE		SHORT	2	80
REQ_MNR_RET_CODE		REQ_MNR_RET_CODE		SHORT	2	82
> REQ_RET_MSG		REQ_RET_MSG		STRUCT	43	125
CARMODEL DEALERS		CARMODEL DEALERS		STRUCT	630518	125
CARMODEL CAR FINDER		CARMODEL CAR FINDER		STRUCT	630518	125
CARMODEL DEALER LOCATION		CARMODEL DEALER LOCATION		STRUCT	118	125
CARMODEL LOCATION		CARMODEL LOCATION		CHAR	2	125
CARMODEL LOC ADDRESS		CARMODEL LOC ADDRESS		CHAR	90	127
CARMODEL COUNTRY		CARMODEL COUNTRY		CHAR	26	217
CARMODEL INFO TABLE		CARMODEL INFO TABLE		STRUCT	630400	243
[#] CARMODEL INFO DATA [1..100]		CARMODEL INFO DATA		ARRAY	624400	243
CARMODEL NBR OF COMMENTS		CARMODEL NBR OF COMMENTS		UINT	4	6000
CARMODEL COMMENTS [1..20]		CARMODEL COMMENTS		ARRAY	6000	
CARMODEL COMMENT A		CARMODEL COMMENT A		STRUCT	16	
CARMODEL COMMENT B		CARMODEL COMMENT B		CHAR	12	
CARMODEL COMMENT C		CARMODEL COMMENT C		CHAR	12	
CARMODEL 1 REF		CARMODEL 1 REF		STRUCT	16	
CARMODEL 2 MODEL		CARMODEL 2 MODEL		CHAR	12	
CARMODEL 3 COST		CARMODEL 3 COST		CHAR	12	
CARODO2C.CARMODEL DEALERS.CARMODEL CAR FINDER.CARMODEL INFO TABLE.CARMODEL INFO DATA.CARMODEL COMMENTS ARRAY OCCURS 1 TO 20 TIMES DEPENDS ON CARODO2C.REQ_TXN_REQUEST_BLOCK_HEADER.REQ_MAX_NBR_OF_COMMENTS						

Figure 53. An array counter used in a nested ODO array.

For request service interfaces, ODO/REFER array objects are automatically set by z/OS Connect EE so that the application receives only as many array entries in the ODO/REFER array subject as are present in the JSON. For nested ODO/REFER array subjects, respective ODO array objects are automatically set to the maximum number of entries across all occurrences of the nested array subject in the request JSON.

For response service interfaces, ODO/REFER array objects must be set by the application so that z/OS Connect EE processes only the requested number of array entries in the respective ODO/REFER array subjects. For nested ODO/REFER array subjects, respective ODO array objects must be set by the application to the maximum number of entries across all occurrences of the array subject in the response data.

For more information about array counters, see “Defining array counters” on page 499.

Conclusion

You can use these techniques separately or concurrently to ensure that z/OS Connect EE processes only the payload data that is relevant to your application and eliminate unnecessary transformation.

Chapter 7. Configuring

Use this section to configure the different elements of z/OS Connect EE and the back-end z/OS subsystem.

Note: All configuration settings for the server are contained in the `server.xml` file. To change the configuration, you need authority to edit this file.

The `server.xml` configuration file is tagged in the z/OS file system as a text file with an ISO8859-1 encoding. If the z/OS UNIX System Services (USS) automatic code page conversion feature is active, you can use standard editors and text utilities to view and manage the configuration file.

Creating a z/OS Connect Enterprise Edition server

Use the `zosconnect` command to create a z/OS Connect EE server.

Before you begin

To use the `zosconnect` command, you must be a z/OS Connect EE administrator with access to the OMVS shell.

Before you create a z/OS Connect EE server, ensure that the shared directory is available. For more information, see [“Creating the z/OS Connect EE shared directory” on page 196](#).

For step-by-step examples of creating a server and connecting it to a System of Record (SoR), see:

- [“Create a server and connect to CICS” on page 69](#)
- [“Create a server and connect to IMS” on page 72](#)
- [“Create a server to connect to IBM MQ” on page 77](#)
- [“Create a server and connect to an IMS database” on page 74](#)

About this task

This task shows you how to use the default template to create a z/OS Connect EE server. Templates are a Liberty feature that you can use when you create a server. Other service providers can supply their own templates to enable a quicker start with their functions. For more information, see [Creating Liberty servers from custom configurations](#) in the *WebSphere Application Server Liberty* documentation.

Note: The file permissions on the server are controlled by the umask of the user that created the server.

Procedure

1. Optional: Set the `WLP_USER_DIR` environment variable to the location where you want your server instances and user features to be stored. For example, `/var/myserver`. If you do not set this value, the default, `/var/zosconnect` will be used.

You can specify the `WLP_USER_DIR` environment variable in the shell environment. This variable must be an absolute path.

2. Run the `zosconnect` command with the `create` option.

Use the following syntax:

```
zosconnect create <server name> --template=<template name>
```

For example, to create a server named `server1` with the default z/OS Connect EE template, type the command:

```
zosconnect create server1 --template=zosconnect:default
```

To create a server for the z/OS application to call REST APIs, you can use the specific template for API requester as in the following command:

```
zosconnect create myserver --template=zosconnect:apiRequester
```

Note:

- You must add the same *WLP_USER_DIR* environment variable to the started task procedure:

```
//STDENV *
```

```
WLP_USER_DIR=/var/my servers
```

For more information about the started task procedure, see [“Starting and stopping z/OS Connect EE” on page 471](#).

- To specify optional environment variables you can use either of these methods:
 - Specify the STDENV variables in line, as shown in the sample started task procedure provided in *<hlq>.SBAQSAMP(BAQSTRT)*.
 - Use a z/OS UNIX file. For example, place the file *<name>.env* in the *<WLP_USER_DIR>/servers* directory, enter each of your environment variables on separate lines, and add the following to the started task procedure:

```
//STDENV DD PATH='<WLP_USER_DIR>/servers/<name>.env'
```

where *<name>.env* is the name of the file containing the environment variables in the *<WLP_USER_DIR>/servers* directory.

- To specify JVM options, you must use the *JVM_OPTIONS STDENV* variable. If you need to specify multiple JVM options which would exceed the line length of your started task procedure, you can either store all the STDENV variables in a z/OS UNIX file as described above, or store only your JVM options in a z/OS UNIX file. For example, create a z/OS UNIX EBCDIC file called *<name>.options* in the *<WLP_USER_DIR>/servers* directory, enter each of your JVM options on separate lines and add the following to the STDENV statement of the started task procedure:

```
JVM_OPTIONS=-Xoptionsfile=<WLP_USER_DIR>/servers/<name>.options
```

where *<name>.options* is the name of the file that contains the JVM options in the *<WLP_USER_DIR>/servers* directory. For more information, see https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.zos.80.doc/diag/appendices/cmdline/Xoptionsfile.html?cp=SS7K4U_liberty in the *WebSphere Application Server for z/OS Liberty* documentation.

Results

A z/OS Connect EE server is created in the *<WLP_USER_DIR>/servers* directory with a *server.xml* configuration file that defines the z/OS Connect EE feature and the default location of the API archive files, API requester archives, service archives and policy rules. The shared resources and the server definitions are also located in this directory when servers are running.

The following table shows the directory structure and contents when a server is created:

Table 17. Directory paths and contents	
Path	Contents
<i><WLP_USER_DIR>/servers</i>	Server instances. One subdirectory per server instance, each created by using the create option of the zosconnect command.
<i><WLP_USER_DIR>/extension</i>	The location where user features are installed.

Related tasks

[“Deleting or renaming a z/OS Connect Enterprise Edition server” on page 219](#)

To delete a z/OS Connect EE server, delete the server directory under <WLP_USER_DIR>/servers. To rename a server, change the server directory name.

Deleting or renaming a z/OS Connect Enterprise Edition server

To delete a z/OS Connect EE server, delete the server directory under <WLP_USER_DIR>/servers. To rename a server, change the server directory name.

About this task

If a z/OS Connect EE server instance is no longer needed, or you want to rename a server, delete or rename the directory that is named after the server under the <WLP_USER_DIR>/servers directory.

Procedure

1. Ensure the server is stopped.
2. Go to the <WLP_USER_DIR>/servers directory.
3. Delete or rename the directory for the server instance.

Related information

[“Starting and stopping z/OS Connect EE” on page 471](#)

Configuring services

z/OS Connect EE service definitions provide the mechanism for reaching z/OS application assets using REST calls. Services require the zosconnect:zosconnect-2.0 feature to be configured in server.xml.

Before you begin

Recommendations:

z/OS Connect EE provides service implementation by using CICS, WOLA, IMS, and MQ service providers, in addition to the REST client service, all of which are supplied with z/OS Connect EE. Services implemented in this way are defined using service archive (.sar) files.

If services cannot be defined by using these methods, or if you have specific layout requirements for the Liberty server.xml configuration file, manually configure service definitions in the server.xml file.

For more information about service definition using the WebSphere Optimized Local Adapters (WOLA) service provider, see [“Using the WOLA service provider” on page 231](#).

About this task

z/OS Connect EE services are defined by the zosConnectService configuration element. This element contains attribute definitions that might have globally defined counterparts. Attributes can be defined globally through the zosConnectManager element. These attributes, if defined, apply to all services. If both global and service level attributes are configured, the values defined at the zosConnectService level are used.

Note: If your service is called as part of an API call, the interceptors and security configuration included with the API will override the configuration included in the service.

Procedure

1. Choose one of the following options.

- For services that are defined by service archive files (.sar files), ensure you have specified a `zosconnect_services` element in the `server.xml` configuration file.

If you wish to override any of the default values assigned to a service, add a nested `service` subelement, and if applicable, a nested `properties` element to override property values defined in the .sar file.

The following excerpt from a `server.xml` file shows a sample definition for a CICS service called `HealthService`, which does not require SSL and which overrides the `transid` value specified in the .sar file with the value `PDMI`:

```
<zosconnect_services>
  <service name="HealthService" requireSecure="false">
    <property name="transid" value="PDMI"/>
  </service>
</zosconnect_services>
```

- For old style services that are not defined by service archive files, add a `zosconnect_zosConnectService` configuration element for each service in your `server.xml` configuration.

You must configure the `serviceRef` and `serviceName` attributes, other attributes are optional. The `serviceRef` element points to the service provider configuration element. The `serviceName` attribute identifies the service to z/OS Connect EE and is also used as part of the URL for z/OS Connect EE requests that are targeted to a specific service. The service name must be unique.

The following excerpt from a `server.xml` file shows a sample definition for a z/OS Connect EE service called `recordOpsCreate`:

```
<zosconnect_zosConnectService
  serviceName="recordOpsCreate"
  serviceRef="<varname>{id of service provider}</varname>" />
```

For more information about each attribute, see [“Configuration elements” on page 753](#) in the *Reference* section.

2. Optional: Add the `zosConnectManager` configuration element to the `server.xml` file.

This element is a singleton in the `server.xml` configuration and it contains global values that apply to all z/OS Connect EE services that are defined for the server. The element can contain the names of z/OS Connect EE interceptors or data transformation provider configuration elements that apply to all services in the server.

The following example describes a `zosConnectManager` configuration element that defines an operations group name of `Operator1` under the item `globalOperationsGroup`. This is the name of the security (SAF or LDAP) group that the requesting client user ID needs to be in before z/OS Connect EE operations requests are permitted, such as `action=start|stop|status`. This group applies to all services in the z/OS Connect EE configurations. The item `globalDataformRef` defines the element name where z/OS Connect EE will find a data transformation provider. Supplying the `globalDataformRef` in the `zosConnectManager` element means this is the transformation provider implementation for all services in the configuration that do not already have their own data transformation element reference. The item `globalInterceptorsRef` is the name of the element in the configuration that describes the set of z/OS Connect EE interceptors that apply to all services in the configuration.

```
<zosconnect_zosConnectManager
  globalOperationsGroup="Operator1"
  globalDataformRef="XformJSON2Byte"
  globalInterceptorsRef="GlobalInterceptors"/>
```

For more information on each attribute, see [“Configuration elements” on page 753](#) in the *Reference* section.

Using the CICS service provider

This CICS service provider that is supplied with z/OS Connect EE connects to CICS using an IPIC connection. Both COMMAREA and channels with multiple containers are supported.

Services are defined using service archive (.sar) files.

You can set the transaction ID on the connection for all services. Furthermore, individual services can override the connection's transaction ID with a transaction ID in the service archive file, or in the server.xml configuration file using the service element under the zosconnect_services element.

For a step-by-step example of creating a z/OS Connect EE server and connecting it to CICS, see [“Create a server and connect to CICS” on page 69](#).

Configuring an IPIC connection to CICS

Configure one or more IPIC connections to your CICS systems.

Study the following topics to create IPIC connections between z/OS Connect EE and CICS

IP interconnectivity (IPIC) overview

IPIC provides access to CICS applications over the TCP/IP protocol, supporting both COMMAREA and CICS channel applications. COMMAREAs are limited to 32KB of application data, while CICS channels allow you to send and receive more than 32KB of application data in a single request.

Connections to CICS

IPIC communications between z/OS Connect EE and CICS Transaction Server V4.1 (or later), use up to two sockets for each IPIC connection. If the IPIC connection is defined to use a maximum of one session, a single socket is used.

If one or more of the socket connections used for an IPIC connection ends unexpectedly, for example, because of a network error, all the sockets are closed and the IPIC connection is released.

Connection sessions

Each IPIC connection can be defined with up to 999 sessions. A single session handles a single request at a time and uses one CICS task, so that the number of sessions determines the maximum number of simultaneous requests that can be outstanding over an IPIC connection. The number of sessions is defined on the CICS server by the Receivecount parameter of the IPCONN definition and in z/OS Connect EE by the sendSessions attribute of the zosconnect_cicsIpicConnection element in the server.xml configuration file. If the number of sessions differs on either end of the connection the actual number of sessions established is negotiated when the connection is established.

Setting session limits

The actual number of send sessions established depends on the values of the sendSessions and Receivecount attributes.

Table 18. How the number of simultaneous transactions over an IPIC connection is determined		
zosconnect_cicsIpicConnection sendSessions value	IPCONN Receivecount value	Number of simultaneous transactions allowed
Set	Set (on IPCONN resource definition or customized autoinstall)	The lesser of the two values is used.
Set	Not set (default autoinstall)	The value of the z/OS Connect EE sendSessions setting is used.

Table 18. How the number of simultaneous transactions over an IPIC connection is determined (continued)

zosconnect_cicsIpicConnection sendSessions value	IPCONN Receivecount value	Number of simultaneous transactions allowed
Not set	Set (on IPCONN resource definition or customized autoinstall)	The value of the CICS IPCONN Receivecount setting is used.
Not set	Not set (default autoinstall)	A value of 100 is used.

For more information about parameters for the `zosconnect_cicsIpicConnection` element, see [Configuration elements](#) in the *Reference* section.

Configuring an IPIC connection in CICS

Follow these steps to configure an IPIC connection in CICS Transaction Server.

Before you begin

For information about security options available with IPIC connections, see “[Configuring security for an IPIC connection](#)” on page 225.

About this task

Your CICS system administrator must make the following configuration changes.

Procedure

1. Set the System Initialization (SIT) parameter `TCPIP=YES`.
2. Define the TCP/IP address and host name for the z/OS Connect EE server. By default, they are defined in the `PROFILE.TCPIP` and `TCPIP.DATA` data sets.
3. Add a TCP/IP listener to CICS.

Use the following CEDA command to define a `TCPIPSERVICE` in a group:

`CEDA DEF TCPIPSERVICE(service-name) GROUP(group-name)`

Ensure that the group in which you define the service is in the startup `GRPLIST` so that the listener starts when CICS is started. The following list shows the key attributes:

POrtnumber

The port on which the TCP/IP service listens.

PRotocol

The protocol of the service is IPIC.

TTransaction

The transaction that CICS runs to handle incoming IPIC requests. Set it to CISS (the default).

Backlog

The number of TCP/IP requests that are queued before TCP/IP starts to reject incoming requests.

Ipaddress

The IP address (in dotted decimal form) on which the `TCPIPSERVICE` listens. For configurations with more than one IP stack, specify ANY to make the `TCPIPSERVICE` listen on all addresses.

SOcketclose

Specifies whether CICS waits to close the socket after it issues a `receive` for incoming data on that socket. Use NO for IPIC connections to ensure that the connection from z/OS Connect EE always remains open.

SSL

Specifies whether the TCP/IP service is to use the secure sockets layer (SSL) for encryption and authentication.

- Set this value to No when SSL is not to be used on the connection with the z/OS Connect EE server.
- Set this value to Yes when SSL is to be used on the connection with the z/OS Connect EE server.
- Set this value to Clientauth when SSL is to be used on the connection with the z/OS Connect EE server. In this case, CICS also expects to receive a client certificate from the z/OS Connect EE server during the SSL handshake, when the connection is being acquired. You must specify this value if you send an authenticated identity to CICS and the CICS region is in a different sysplex from the z/OS Connect EE server.

CErtificate

This attribute is only applicable when **SSL(Yes)** or **SSL(Clientauth)** is also specified. Specifies the label of an X.509 certificate that is used as the server certificate during the SSL handshake when the connection with the z/OS Connect EE server is acquired. If this attribute is omitted, the default certificate that is defined in the key ring for the CICS region user ID is used. The certificate must be stored in a key ring in the database of the external security manager.

4. Use the following command to install the TCPIPSERVICE definition:

CEDA INS TCPIPSERVICE (*service-name*) **GROUP** (*group-name*)

5. Choose whether to predefine or to autoinstall IPIC connections in CICS Transaction Server.

You can predefined IPCONN connection definitions in CICS, or connections can be configured to be auto-installed by either the default or a customized autoinstall program. When z/OS Connect EE connects to CICS, it flows the *zosConnectApplid* and *zosConnectNetworkid* values that are defined on the *zosconnect_cicsIpicConnection* element in *server.xml*. If these values match the **Applid** and **Networkid** attributes of an IPCONN definition, then that definition is used to install the connection. If no matching IPCONN definition exists, the connection is auto-installed.

Note: If the *zosConnectNetworkid* is not specified in *server.xml* and the Networkid is left blank in the IPCONN definition, a match will not occur because CICS defaults the blank Networkid to the local network ID.

You can customize auto-installed IPIC connections. To do this, you must create an IPCONN definition with the customized attributes to act as a template and this definition must be referenced as the template in a customized IPCONN autoinstall user program. The name of the autoinstall user program must be specified on the URM option of the installed TCPIPSERVICE definition.

6. Create a predefined or auto-installed IPCONN definition.

The following list shows the key attributes:

Applid

For a predefined IPCONN, set this value to the value of the **zosConnectApplid** attribute that is specified on the *zosconnect_cicsIpicConnection* element in *server.xml*. For an auto-installed IPCONN, optionally set this value to identify the instance of the connection.

Networkid

For a predefined IPCONN, set this value to the value of the **zosConnectNetworkid** attribute that is specified on the *zosconnect_cicsIpicConnection* element in *server.xml*. For an auto-installed IPCONN, optionally set this value to identify the instance of the connection.

TCPIPSERVICE

Set this value to match the name of the TCPIPService defined earlier.

Receivecount

Set this value to specify the number of requests that can be processed in parallel. This should match the number set in the **sendSessions** attribute that is specified on the *zosconnect_cicsIpicConnection* element in z/OS Connect EE. For information on how the actual number of sessions is determined on connection establishment, see “[IP interconnectivity \(IPIC\) overview](#)” on page 221.

SENdcount

Set this value to zero because IPIC connections are always inbound to CICS from z/OS Connect EE, unlike CICS to CICS connections for which the value must not be zero.

Inservice

Set this value to Yes.

Linkauth

Set this value to Secuser to use the user ID that is specified in the **SECurityname** attribute.

Set this value to Certuser to use an SSL client certificate that is mapped to a SAF user ID. The referenced TCPIPSERVICE definition must be configured for SSL and client authentication.

SECurityname

Set this value to a SAF user ID that is authorized to establish IPIC connections.

Userauth

Set this value to Local or Defaultuser when no user credentials are required to be passed from z/OS Connect EE. The CICS program then runs under the link user ID or default user ID.

Set this value to Identify when user identity propagation or identity assertion is required and z/OS Connect EE is configured to use distributed identities or flow an asserted SAF user ID. In this instance, if the z/OS Connect EE server is not in the same sysplex as the CICS region, you must use an IPIC SSL connection that is configured with client authentication.

Set this value to Verify when connection level user security is required and a `zosconnect_authData` element is referenced on the `zosconnect_cicsIpicConnection` element in `server.xml`.

SSL

This attribute is not applicable for a connection to z/OS Connect EE. This attribute is used when a CICS region acts as the client end of an IPIC connection to another CICS region.

CErtificate

This attribute is not applicable for a connection to z/OS Connect EE. This attribute is used when a CICS region acts as the client end of an IPIC connection to another CICS region.

What to do next

Follow the instructions in [“Configuring an IPIC connection in z/OS Connect EE” on page 224](#)

Configuring an IPIC connection in z/OS Connect EE

Follow these steps to configure an IPIC connection to CICS in your z/OS Connect EE server.

Before you begin

Before you begin this task, ensure that a server instance has been set up, see [“Creating a z/OS Connect Enterprise Edition server” on page 217](#).

About this task

Edit the `server.xml` file to add the CICS service provider feature, configure a CICS IPIC connection element and optionally define additional security elements.

Procedure

1. Update the `server.xml` file to use the `cicsService-1.0` feature by adding the following configuration information in the `<featureManager>` section:

```
<featureManager>
  .
  .
  <feature>zosconnect:cicsService-1.0</feature>
</featureManager>
```

2. Configure a `zosconnect_cicsIpicConnection` element in `server.xml`. The **host** and **port** attributes must be specified. For example:

```
<zosconnect_cicsIpicConnection id="cicsABC" host="9.1.2.345" port="1110"/>
```

For more information about the available configuration attributes and default values, see [“zosconnect_cicsIpicConnection” on page 763](#).

3. If the CICS system has security initialized (SEC=YES is defined in the CICS SIT parameters), see [“Configuring security for an IPIC connection” on page 225](#).

Transaction override precedence

A transaction code must be specified during CICS service project creation. This transaction code can be overridden in the service interface definitions or at run time.

Precedence for transaction overrides is in the following order from lowest to highest priority:

1. A transaction code must be specified in the `zosconnect_cicsIpicConnection` element.
2. This transaction code can be overridden by the setting in the service archive (`.sar`) file.
3. This transaction code can be overridden by setting `zosconnect_services`.
4. If an active policy contains a rule that modifies the transaction code, the modified value overrides all other transaction code specifications.

Note: If you use your own transid, the transaction must be defined as an exact clone of CSMI. In particular, PROGRAM(DFHMIRS) and PROFILE(DFHCICSA).

Related reference

[“Configuration elements” on page 753](#)

Configuring security for an IPIC connection

IPIC connections can enforce bind security to prevent an unauthorized client system from connecting to CICS, link security to restrict the resources that can be accessed over a connection to a CICS system, and user security to restrict the CICS resources that can be accessed by a user. If the CICS system supports password phrases, a password phrase can be used for user security.

Bind security

Bind security can be applied to check that the remote system is authorized to connect to CICS. Bind security is implemented by configuring the connection to use SSL client authentication.

Link security

Link security restricts the resources that users can access. The practical effect of link security is to prevent a remote user from attaching a transaction or accessing a resource for which the link user ID has no authority.

You can specify the link user ID for IPIC connections to be either a specific SAF user ID or, if the TCPIPSERVICE is configured to require SSL client authentication, the SAF user ID associated with that client certificate. Specify the link user with the SECURITYNAME attribute, or an SSL certificate, in the IPConn definition in CICS. You can use an SSL certificate if you have a client authenticated SSL connection. The client's certificate is mapped by RACF to a specific user ID, which is defined as the link user. With this method, you can specify different link users depending on which certificate you are using. For more information see [“Configuring an IPIC connection in CICS” on page 222](#).

User security

In addition to the security restrictions set by link security, you can further restrict each remote user's access to the transactions and resources in your system. In z/OS Connect EE, user credentials can either

be predefined on specific IPIC connection definitions, or authenticated user identities associated with individual requests are automatically passed to CICS.

To predefine user security on an IPIC connection, the CICS IPCONN definition must be configured with USERAUTH=VERIFY, requiring that a user ID and password is sent to CICS on each request. The security credentials are configured on the `zosconnect_cicsIpicConnection` element in the `server.xml` file. For more information, see [“Configuring IPIC connection level user security” on page 227](#).

To enable authenticated user identities that are associated with individual requests to be sent to CICS, the IPCONN definition in CICS must be configured with USERAUTH=IDENTIFY. If the z/OS Connect EE server is not in the same sysplex as the CICS region, you must use an IPIC SSL connection that is configured with client authentication. The user identity can be any of the following:

- An authenticated distributed identity that is defined in an LDAP registry and mapped to a SAF user ID in the SAF registry that is used by CICS. For more information, see [“Configuring distributed identity propagation” on page 228](#).
- An authenticated SAF user ID that has originated from any security mechanism supported by Liberty. For example:
 - An X.509 client certificate that is mapped to a SAF user ID. For more information, see [“API provider authorization” on page 373](#).
 - Another security credential such as a JWT token mapped to a SAF user ID.

For more information on IPIC security, see the [CICS Transaction Server documentation](#).

Configuring SSL on an IPIC connection

Follow these steps to configure SSL on an IPIC connection to CICS. Client authentication is required if the z/OS Connect EE server and CICS region are in different sysplexes.

Before you begin

1. Configure the TCPIPSERVICE on the CICS region to use SSL. For more information, see [“Configuring an IPIC connection in CICS” on page 222](#).
2. Configure a `zosconnect_cicsIpicConnection` element in `server.xml`. For more information, see [“Configuring an IPIC connection in z/OS Connect EE” on page 224](#).

About this task

The `server.xml` file is updated to configure SSL on an existing IPIC connection definition, with both client and server authentication.

Procedure

1. Create a personal SSL certificate and key ring for the CICS region.

In this example, the CICS region uses the following artifacts:

- A self-signed certificate. In a production system, you should consider using a CA signed certificate.
- A RACF key ring called CICSTS-KEYRING.

If this certificate is not the default in the server keyring, it must be specified in the `Certificate` attribute of the TCPIPSERVICE definition. For more information, see [“Configuring an IPIC connection in CICS” on page 222](#).

2. Create a personal SSL certificate for the z/OS Connect EE server.

In this example, the z/OS Connect EE server uses a self-signed certificate in a JKS keystore.

3. Export the public part of the CICS region's personal certificate and import it into the z/OS Connect EE server's truststore as a trusted certificate.

This allows the z/OS Connect EE server to trust the CICS region during the SSL handshake.

4. If client authentication is required, export the public part of the z/OS Connect EE server's personal certificate and import into the CICS region's RACF key ring.
This allows the CICS region to trust the z/OS Connect EE server during the client authentication step of the SSL handshake.
5. Edit the z/OS Connect EE server configuration file to define an SSL element or SSLDefault element.
6. Add an **sslCertsRef** attribute to the `zosconnect_cicsIpicConnection` element to reference the SSL element.
For example,

```

<featureManager>
  <feature>zosconnect:cicsService-1.0</feature>
</featureManager>

  <!-- Define the SSL configuration. -->
  <!-- Set clientAuthentication to true if the z/OS Connect EE server and CICS region are
in different sysplexes. -->
  <ssl id="defaultSSLConfig" keyStoreRef="defaultKeyStore"
trustStoreRef="defaultTrustStore" clientAuthentication="false" />

  <!-- Define a keystore. Contains the z/OS Connect EE server's personal certificate to be
sent on SSL handshake. -->
  <keyStore id="defaultKeyStore" password="zosconnect" location="${server.config.dir}/
resources/security/serverKey.jks" />

  <!-- Define a truststore. Contains the CICS region's public certificate expected to be
sent on the SSL handshake. -->
  <keyStore id="defaultTrustStore" password="zosconnect" location="${server.config.dir}/
resources/security/serverTrust.jks"/>

  <!-- Define the IPIC connection to CICS -->
  <zosconnect_cicsIpicConnection id="cicsConn" sslCertsRef="defaultSSLConfig"
host="9.1.2.345" port="1110" />

```

Configuring IPIC connection level user security

Follow these steps to configure predefined user security credentials for an IPIC connection to CICS.

Before you begin

1. Configure an IPCONN definition in CICS with USERAUTH=VERIFY. For more information, see [“Configuring an IPIC connection in CICS” on page 222](#).
2. Configure a `zosconnect_cicsIpicConnection` element in `server.xml`. For more information, see [“Configuring an IPIC connection in z/OS Connect EE” on page 224](#).
3. Locate the WebSphere Liberty profile server **securityUtility** command tool, which can be found in the `<installation_path>/wlp/bin` directory.

Procedure

1. Use the **securityUtility encode** command to encode the password for the user ID that is to be defined for the connection. For more information, see the [securityUtility command](#).
2. Define a `zosconnect_authData` element in `server.xml`, the **user** and **password** attributes must both be specified. For the password, specify the entire encoded string output by running the **securityUtility** command.
3. Add an **authDataRef** attribute to the `zosconnect_cicsIpicConnection` element to reference the `zosconnect_authData` element. For example:

```

<zosconnect_authData id="authABC" user="sysidABC" password="{$xor}0jIvb18oMzs="/>
<zosconnect_cicsIpicConnection id="cicsABC" host="9.1.2.345" port="1110" authDataRef="authABC" />

```

Configuring distributed identity propagation

Follow these steps to configure the propagation of distributed identities over IPIC connections to CICS for user authorization.

Before you begin

1. Configure an IPCONN definition in CICS with USERAUTH=IDENTIFY. For more information, see [“Configuring an IPIC connection in CICS” on page 222](#).
2. Configure a zosconnect_cicsIpicConnection element in server.xml. For more information, see [“Configuring an IPIC connection in z/OS Connect EE” on page 224](#).

About this task

The server.xml file is updated to define an LDAP user registry. As CICS retrieves the mapped SAF user IDs from distributed identities for user authorization, there is no need to map the distributed identity to a SAF user ID in z/OS Connect EE. No change to the zosconnect_cicsIpicConnection element is required.

Procedure

1. Configure an LDAP user registry.

For example, add the following elements to server.xml:

```
<featureManager>
  ...
  ...
  <feature>ldapRegistry-3.0</feature>
</featureManager>
<ldapRegistry id="LDAP"
  realm="SampleLdapIDSRealm" host="myserver" port="123"
  ignoreCase="true"
  baseDN="o=myco,c=us"
  userFilter="(&(uid=%v)(objectclass=ePerson))"
  groupFilter="(&(cn=%v)(|(objectclass=groupOfNames)
  (objectclass=groupOfUniqueNames)(objectclass=groupOfURLs)))"
  userIdMap="*:uid"
  groupIdMap="*:cn"
  groupMemberIdMap="myco-allGroups:member"
  ldapType="IBM"
  Tivoli Directory Server"
  searchTimeout="8m">
</ldapRegistry>
```

2. Define users in the LDAP registry and give them the required access to the z/OS Connect EE server in the authorization-roles element in the server.xml file.
3. Set **requireAuth="true"** on the zosconnect_zosConnectManager element or on the zosconnect_services element for the individual CICS services.

This setting ensures that API and service requests fail with a 401 HTTP response if invalid credentials are supplied, and no attempt is made to process the request without valid credentials. If **requireAuth="false"** is defined, the request is rejected by the CICS server and failsCICS with a 500 HTTP response.

4. Map each distributed identity to a SAF user ID in the security manager that is used by CICS. More than one distributed identity can be mapped to the same SAF user ID.

Note: You only need to set `<safCredentials mapDistributedIdentities="true" />` if you are using SAF authorization in z/OS Connect EE. The distributed identity is always mapped to a SAF identity in CICS.

Results

You can view the distinguished name and realm for a distributed identity in the association data of the CICS task, by using the command **CEMT INQUIRE ASSOCIATION(taskid)**, or in the Task Associations view in CICS Explorer.

The distinguished name and realm information can also be recorded to SMF in type 110, subtype 01 records. For more information, see [Identity class data](#). The identity monitoring data values of interest are **MNI_ID_USERID**, which contains the mapped SAF user ID, and the **MNI_ENTRY_FIELD**, which contains the distinguished name and realm. These fields are described in [Identity class data: Listing of data fields](#) and [MNI - Transaction identity monitoring data](#). CICS Transaction Server provides a sample program to print monitoring data, called DFH\$MOLS. For more information, see [Sample monitoring data print program \(DFH\\$MOLS\)](#).

Handling request timeouts

In asynchronous mode, requests time out if they do not complete within the time that is specified by the **asyncRequestTimeout** attribute on the `zosconnect_zosConnectManager` element in `server.xml`.

If the processing of a request is timed out, the client receives a 503 HTTP response code. The CICS service provider terminates processing of the request and the associated IPIC session is freed. The time to process the first request that is sent to any CICS server includes the time taken to establish the IPIC connection. This time must be considered when the value of the **asyncRequestTimeout** attribute is set.

Configuring IPIC High Availability

CICS service provider IPIC connections can use port sharing or be configured to use sysplex distributor to connect to a cluster of CICS regions. The CICS regions must be configured with the same program, transaction, and security settings. The IPIC connections can be configured to use SSL.

Before you begin

A shared TCP/IP port must be configured, which each of the CICS regions that connect from z/OS Connect EE can listen on. For example, by using z/OS TCP/IP port sharing or sysplex distributor.

Procedure

1. Configure z/OS Connect EE

Perform the following steps for each z/OS Connect EE server that connects to the CICS cluster.

- a) On the `zosconnect_cicsIpicConnection` element, set **sharedPort=true** and specify the shared port in the **port** attribute.

The shared port is the port number configured in TCP/IP port sharing or sysplex distributor, which the CICS regions will listen on

- b) Optionally, specify the **zosConnectApplid** and **zosConnectNetworkid** attributes to identify an auto-installed IPCONN or match a predefined IPCONN. Do not set the **cicsApplid** and **cicsNetworkid** attributes; if specified, these parameters will be ignored by CICS when **sharedPort=true**.

2. Configure CICS.

Perform the following steps for each CICS region in the cluster. The specific TCPIPSERVICE definition must be installed before the associated generic TCPIPSERVICE definition. If you create both definitions in the same CSD group, ensure that the name of the specific TCPIPSERVICE is alphabetically earlier than the generic TCPIPSERVICE.

- a) Define a specific TCPIPSERVICE to listen on a non-shared port specific to that CICS region.
- b) Define a generic TCPIPSERVICE in CICS to listen on the shared port and set the **SPECIFTCPS** value to the name of the specific TCPIPSERVICE.

3. Complete the configuration by following the steps in [Defining IPIC high availability connections in the CICS Transaction Server for z/OS documentation](#),

In these tasks, z/OS Connect EE acts as the CICS client region.

Context containers

In some situations, CICS programs need information about the context in which they were called. The CICS service provider sends this information to CICS in context containers.

The use of context containers is optional and available only with CHANNEL, not COMMAREA, programs. You can enable the function in SAR files for the CICS service provider. To enable this function, use either the **Use context containers** and **Context containers HTTP headers** configuration options in the API toolkit, or use the `useContextContainers` and `httpHeaders` parameters in the build toolkit. For more information, see [“Configuring service properties” on page 510](#) for the API toolkit, or the [“Creating a CICS service for C or top-down” on page 798](#) for the build toolkit.

The CICS service provider can send the following context containers, which are all of type CHAR.

DFHWS-URI

The full URL for the request. For example, `https://localhost:9080/zosConnect/services/cicsService?action=invoke`.

DFHWS-URI-QUERY

The query string for the request. For example, `action=invoke`.

DFHHTTPMETHOD

The HTTP method for the request. The container is padded with spaces to be 8 characters long.

BAQHTTPHEADERS

Specific headers on the request in a JSON document in the following format.

```
{ "httpHeaders":  
  { "<headerName>": "<headerValue>", ... }  
}
```

For example:

```
{ "httpHeaders":  
  { "MyHeader": "Somevalue", "SecondHeader": "OtherValue" }  
}
```

For more information about specifying the headers to include, see **Context containers HTTP headers** configuration options in the API toolkit [“Configuring service properties” on page 510](#) or the `httpHeaders` parameter for the build toolkit [“Creating a CICS service for C or top-down” on page 798](#).

Note:

1. The DFHWS-URI-QUERY container is created only if a query string exists in the URL.
2. The BAQHTTPHEADERS container is created only if the **Context containers HTTP headers** configuration option is configured for the API toolkit or the `httpHeaders` parameter is specified for the build toolkit.
3. Headers are considered optional by the CICS service provider, so if a header is specified in **Context containers HTTP headers** or `httpHeaders` but not included in the HTTP request, the service provider does not include the header in the container. It does not treat this situation as an error. For example, if no headers are on the request the container would be as follows:

```
{ "httpHeaders":  
  { }  
}
```

4. The headers with DFH names are CICS context containers. DFHHTTPMETHOD is documented by CICS TS as a control container, but z/OS Connect EE treats it only as a context container as it does not affect how the response is processed. For more information, see [Containers used in the pipeline](#).

5. Header names are case-insensitive, so the BAQHTTPHEADERS container uses the same case as the header name specified in **Context containers HTTP headers** and **httpHeaders**.
6. Headers that are specified by **Context containers HTTP headers** and **httpHeaders** are not included in any swagger documents for APIs that expose services that use context containers. You must manually add the headers in the request-mapping editor to include them in the API documentation.
7. If the same header appears multiple times on a request, the BAQHTTPHEADERS container includes only one JSON property for the header, with the values concatenated into a single value. For more information, see [HTTP Protocol Specifications RFC 2616](#).

Migration from the WOLA service provider

If you upgrade from the WOLA service provider to the CICS service provider and use context containers, consider these differences.

- Context container names are different for each service provider.

Table 19. Context container names for WOLA and CICS service providers

CICS service provider name	WOLA service provider name
DFHWS-URI	ZCONReqURL
DFHWS-URI-QUERY	ZCONReqQParams
DFHHTTPMETHOD	ZCONReqMethod
BAQHTTPHEADERS	ZCONHTTPHEADERS

- The CICS service provider context containers are all of type CHAR. If the CICS program previously used BIT context containers it must be updated to handle CHAR context containers.
- The DFHHTTPMETHOD context container includes the HTTP method name padded with spaces to be 8 characters long, whereas the ZCONReqMethod context container does not include padding.

Using the WOLA service provider

Follow these steps to use the WOLA service provider supplied with z/OS Connect EE.

Before you begin

The WebSphere Optimized Local Adapter (WOLA) service provider requires that both the `zosconnect:zosconnect-2.0` and the `zosLocalAdapters-1.0` features are configured. The WOLA service provider is included with z/OS Connect EE and uses the WOLA function provided with the Liberty profile. The WOLA service provider can be used by both WOLA-enabled applications and CICS (to support z/OS Connect EE V2 configurations).

About this task

z/OS Connect EE provides a service implementation that allows z/OS Connect EE requests to interact with z/OS assets through WOLA. The service is automatically enabled when both `zosconnect:zosconnect-2.0` and the `zosLocalAdapters-1.0` features are configured. Instances of this service can be defined through the `zosconnect_localAdaptersConnectService` configuration element.

Procedure

For a service to use the WOLA service provider, the service definition requires a `zosconnect_localAdaptersConnectService` configuration element in the `server.xml` file.

The minimum configuration consists of the target WOLA Connection Factory name, Register name (RGN), and Service name. Configuration of z/OS Connect EE follows the Liberty profile conventions by only requiring the minimum parameters to be specified.

For more information about all the supported attributes, see [“zosconnect_localAdaptersConnectService” on page 767](#) in the Reference section.

The only supported authentication mechanism on a WOLA connection factory is container managed authentication. The authentication credentials are defined using an authData element that is referenced on the containerAuthDataRef attribute of the connectionFactory element.

WOLA uses a three part name to uniquely identify the WOLA server. This name is derived from the wolaGroup, wolaName2 and wolaName3 attribute values on the local adapter's configuration element zosLocalAdapters, and must be unique per server in the same z/OS LPAR.

The following code snippet is an example of configuring the WOLA service provider:

```
<!-- z/OS Connect service definitions -->
<zosconnect_zosConnectService id="zcs1"
  serviceName="recordOpsCreate"
  serviceRef="wolaOpsCreate"/>

<!-- Local adapters connection factory definition -->
<authData id="cauth1" user="user1" password="{xor}LD08Ki02KyY=/>
<connectionFactory id="wolaCF" jndiName="eis/ola"
  containerAuthDataRef="cauth1" >
  <properties.ola/>
</connectionFactory>

<!-- Local adapters config -->
<zosLocalAdapters wolaGroup="LIBERTY" wolaName2="LIBERTY"
  wolaName3="LIBERTY" />

<!-- WOLA Connect service and z/OS Connect service definitions -->
<zosconnect_localAdaptersConnectService id="wolaOpsCreate"
  registerName="BATCH01"
  serviceName="COBLPGM1"
  connectionFactoryRef="wolaCF"/>
```

The z/OS Connect EE service implementation that allows z/OS Connect EE requests to interact with z/OS assets through WOLA might require a data transformer. If a data transformer is configured, globally or at the service level, input and output payloads are converted using the configured data transformer. If a data transformer is not configured, and the request contains a JSON payload, the service assumes that the backend program will handle the data conversion. The service converts the JSON payload to a byte array using the encoding specified in the request header or the default JSON encoding of UTF-8. Similarly, if the asset returns a payload and no data transformer is configured, the WOLA service implementation expects to receive a JSON payload in byte array form, which is converted to a JSON response payload following the same encoding rules used for transforming a JSON request payload to a byte array.

The WOLA service provider implementation allows for a number of other WOLA connection factory attributes to be specified in its zosconnect_localAdaptersConnectService configuration element.

You can specify the way for a WOLA CICS Link Server to propagate data to target CICS application programs by using the useCICSContainer attribute in the localAdaptersConnectService element. When the useCICSContainer attribute is set to false, the payload is passed to the target CICS application program by using a COMMAREA; otherwise, the payload is passed to the target CICS application program by using CICS containers.

The following options are supported to flow payloads to CICS application programs:

- Use a COMMAREA.
- Use a channel name of IBM-WAS-ADAPTER to flow a single payload container.
- Use a channel name of your choice to flow a single payload container with the HTTP context containers. The context containers pass information about the HTTP request to the CICS application program. See Table 1 for more information.

Table 20. HTTP context container names and descriptions

Name	Description
ZCONReqURL	Contains the URL of the HTTP request. For example, if an invokeURI definition of / banking/deposit and a query parameter country=USA are used, the URL is https://localhost: port/banking/deposit?country=USA.
ZCONReqQParams	Contains the query parameters of the HTTP request. For example, if a request URL is https://localhost:port//banking/deposit?country=USA, the query parameter is country=USA.
ZCONReqMethod	Contains the method of the HTTP request, for example, GET.
ZCONHTTPHeaders	Contains a JSON-formatted set of HTTP header name and value pairs that is configured by using the linkTaskChanCtxContHttpHeaders attribute under the zosconnect_localAdaptersConnectService element.

Configuring CICS to use WebSphere optimized local adapters (WOLA)

z/OS Connect Enterprise Edition includes a WOLA service provider, which uses WOLA to communicate with CICS programs.

About this task

WOLA is provided as part of the Liberty profile that is included in the z/OS Connect Enterprise Edition installation. To configure CICS to use WOLA, copy the WOLA modules from the z/OS Connect Enterprise Edition installation directory to a load library that is referenced in the DFHRPL DD concatenation of the CICS region startup JCL. Next, define WOLA CSD definitions to your CICS region and a SAF profile to control which external user IDs can use WOLA to register to the z/OS Connect EE server.

Note: The Liberty feature that is included in both CICS TS and z/OS Connect EE, includes the BBOA* WOLA modules, but the versions of these modules are potentially different. Always upload the modules that are included with z/OS Connect EE as these modules are tested to ensure compatibility.

Procedure

1. Allocate a PDSE, for example called LIBERTY.WOLA.LOAD, with the following values: Space units = TRACK; Primary quantity = 30; Secondary quantity = 2; Directory blocks = 15; Record format = U; Record length = 0; Block size = 32760 and Data set name type = LIBRARY

```

Data Set Name . . . : LIBERTY.WOLA.LOAD
Management class . . . : STANDARD
Storage class . . . : STANDARD
Volume serial . . . : 3TP001
Device type . . . :
Data class . . . :
Space units . . . : TRACK

Average record unit :
Primary quantity . . . : 30
Secondary quantity . . . : 2
Directory blocks . . . : 15
Record format . . . : U
Record length . . . : 0
Block size . . . : 32760
Data set name type : LIBRARY

```

2. Open a Telnet or ssh session to your z/OS system UNIX System Services:

- a) Change to the z/OS Connect Enterprise Edition <installation_path>/wlp/clients/zos directory that contains the WOLA modules.

For example, /usr/lpp/IBM/zosconnect/v3r0/wlp/clients/zos.

- b) Enter the following command to copy the modules from the UNIX System Services file system to the PDSE you allocated in previous step:

```
cp -Xv ./* //"data.set"
```

Where //"data.set" is the location of the target PDSE. For example,

```
cp -Xv ./* //"LIBERTY.WOLA.LOAD"
```

- c) Check the contents of the PDSE to ensure that all the modules were copied successfully.

3. Edit your CICS region JCL start procedure to add the PDSE containing the WOLA modules.

For example, add LIBERTY.WOLA.LOAD, to the DFHRPL DD concatenation.

4. Define the WOLA resources, transactions, and TD queue to the CICS region DFHCSD.

The CSD definition that is required for the Liberty profile WOLA modules is provided on a GitHub website.

- a) In a web browser, open the URL <https://github.com/WASdev/sample.wola> and click the file CSDUPDAT.jclsamp to open the file.
Click the **Raw** to remove the line numbers.
- b) Copy the contents of the file CSDUPDAT.jclsamp into a new file called CSDUPDAT on your workstation.
- c) Upload file CSDUPDAT in ASCII (record format FB, record length 80) on your z/OS system.
- d) Customize the CSDUPDAT JCL member STEPLIB and DFHCSD DD cards for your CICS installation.
- e) Review and modify the ADD statement at the end of the file to conform with your CICS installation standards for GROUPs and LISTs.
- f) Run the job and check that it completes with RC=0.
- g) If you want the CSD GROUP(BBOACSD) to be installed when CICS starts, add its LIST to the CICS region startup JCL GRPLIST SIT parameter.

5. Set up the Liberty message catalog in the CICS region

The optimized local adapter programs issue messages from a message catalog that is provided with Liberty. For the programs to issue messages, the NLSPATH environment variable in the CICS region must point to the directory that contains the message catalog. This directory is wlp/lib/native/zos/s390x/nls/%N.cat, where wlp is the directory in which the Liberty server is installed.

To set the environment variable, use the Language Environment® ENVAR option, which you can set by editing the CEEROPT CSECT that the CICS region uses. After you edit the CSECT, you can build, compile, link, and copy the CSECT into the DFHRPL data set. For more information about other ways to set Language Environment options, see the documentation for your version of CICS.

Note: Methods for setting Language Environment options that involve editing the application source code or relinking the application are not supported because the optimized local adapter programs cannot be recompiled or relinked.

The following example shows a CEEROPT CSECT that defines the *NLSPATH* environment variable for a Liberty server that is installed in /u/MSTONE1/wlp. The definition spans two lines and includes a continuation character, X, in column 72.

```
CEEROPT CSECT
CEEROPT AMODE ANY
CEEROPT RMODE ANY
*****
*
* Utility: CEEROPT
* Purpose: Set default LE runtime options for CICS region.
*
*****
CEEXOPT ENVAR=(( 'NLSPATH=/u/MSTONE1/wlp/lib/native/zos/s390x/n1X
s/%N.cat' ),OVR)
END
```

6. Define the SAF CBIND profile to allow remote clients (for example, CICS) to register with WOLA.

In these examples, RACF commands are used.

- Enter the command:

```
RDEF CBIND BBG.WOLA.group.name2.name3 UACC(NONE)
```

Where *group*, *name2*, and *name3* match the values that are specified on the *zosLocalAdapters* element in the *server.xml* configuration file to create the three-part name for WOLA. For example,

```
RDEF CBIND BBG.WOLA.LIBERTY1.LIBERTY2.LIBERTY3 UACC(NONE)
```

Note:

- You can use a wildcard in the profile to make it more generic. For example, the CBIND profile **BBG.WOLA.*** would apply to any three-part name you use in Liberty.
- CBIND profiles are stored in uppercase. Specify uppercase values in your *server.xml* *zosLocalAdapters* element because lowercase characters in your three-part name cause a mismatch when the CBIND authority is checked.

- Grant the CICS region user ID (for example, *cics_id*) READ access to the CBIND profile

```
PERMIT BBG.WOLA.group.name2.name3 CLASS(CBIND) ACCESS(READ) ID(cics_id)
```

For example,

SETROPTS RACLIST(CBIND) REFRESH

7. Start the CICS region to ensure that the WOLA definitions are installed successfully.

- Ensure that CSD GROUP(BBOACSD) defined in step 4 is installed (install manually now if it is not included in a List in the GRPLIST SIT parameter).
- Confirm that the WOLA support was added, by checking the MSGUSR job log output for the following messages:

```
Resource definition for BBOACLNK has been added.  
Resource definition for BBOACNTL has been added.  
Resource definition for BBOACSRV has been added.  
Resource definition for BBOATRUE has been added.  
TRANSACTION definition entry for BBO$ has been added.  
TRANSACTION definition entry for BBO# has been added.  
TRANSACTION definition entry for BBOC has been added.
```

8. Stop the CICS region

You will need to restart CICS after the z/OS Connect EE server is configured and started.

What to do next

More advanced configuration options are also available, such as:

- Enabling the WOLA Task Related User Exit (TRUE) to start during CICS region startup.
- Configuring the Link Server Task to start by using either INITPARM or a sequential terminal.

These options are similar to the actions required when WOLA is used in WebSphere Application Server for z/OS. For more information, see the *Full Function WebSphere Application Server z/OS WOLA Quick Start Guide* also available from [Techdoc WP101490](#).

Registering CICS with the WebSphere optimized local adapter (WOLA)

To use the z/OS Connect Enterprise Edition WOLA service provider to connect to CICS, you must first configure both your z/OS Connect EE server and your CICS region. You must then use a CICS transaction to register the CICS region as a client of the WOLA running in the z/OS Connect EE server.

Before you begin

The following tasks must be completed:

1. Configure the SAF definitions required to permit your z/OS Connect EE server the authority to access the z/OS authorized services required by WOLA. Instructions for RACF are documented in [“Configuring the Liberty Angel process and z/OS authorized services” on page 465](#).
2. Configure your z/OS Connect EE server to use the WOLA service provider. Follow the instructions in [“Using the WOLA service provider” on page 231](#).
3. Configure your CICS region to use WOLA. Follow the instructions in [“Configuring CICS to use WebSphere optimized local adapters \(WOLA\)” on page 233](#).

About this task

In this task, you run a CICS transaction to start the WOLA Task Related User Exit (TRUE) and the Link Server task. This registers the CICS region as a client of the WOLA running in the z/OS Connect EE server.

Procedure

1. Ensure that the Angel process is running.

If it is not running, start it by issuing the following command on the z/OS operator console:

S BAQZANGL

Note: The Angel process must be running before a z/OS Connect EE Server that is configured to use WOLA is started, because the server connects to the Angel process to access the z/OS authorized services.

2. Start the z/OS Connect EE server.

Check the server's message log file in `<WLP_USER_DIR>/servers/<serverName>/logs/messages.log`. The following messages indicate that the server was started successfully:

```
CWWKB0103I: Authorized service group LOCALCOM is available.  
CWWKB0103I: Authorized service group WOLA is available  
....  
CWWKB0501I: The WebSphere Optimized Local Adapter channel registered with the Liberty profile server using the following name: <GROUP> <NAME2> <NAME3>  
J2CA7001I: Resource adapter ola installed in 0.327 seconds.
```

where:

- <GROUP>, <NAME2>, and <NAME3> must match the **wolaGroup**, **wolaName2** and **wolaName3** attribute values specified on the `zosLocalAdapters` element in `server.xml`.
- The availability of the **LOCALCOM** service group is a key indicator that the Angel process is running and your Liberty Server ID has READ access to that server profile.
- The availability of the WOLA service group is another key indicator that the configuration was completed successfully.
- Message CWWKB0501I shows the successful use of the WOLA three-part name values you provided in the `server.xml` file. This message indicates that an external address space might register into the Liberty Profile server, in this instance the z/OS Connect EE server, by using this three-part name on the BBOA1REG API.
- The J2CA7001I message indicates that the WOLA JCA resource adapter is available.

3. Ensure that the CICS region is started.

4. Start the WOLA TRUE and Link Server task from CICS, and register with WOLA.

a) Enter the following command on the command line of the CICS region:

BBOC START_TRUE

The following message indicates success:

```
WOLA TRACE 1: Exit enabled successfully.
```

b) Enter the following command to start the link server task and register to your server:

BBOC START_SRVR RGN=<registerName> DGN=<GROUP>
NDN=<NAME2> SVN=<NAME3> SVC=*> SEC=N REU=Y

The following message indicates success:

```
WOLA TRACE 0: Start server completed successfully.
```

Messages are also written to the CICS region job log BBOOUT.

If you get a non-zero return code and reason code, go to the topic [Optimized local adapters APIs on Liberty for z/OS](#) in the *WebSphere Application Server z/OS Liberty* documentation. Search on the string `cdat_olaapis`. Open the link and go to the **Register** section, also known as the BBOA1REG section and look for the RC and RSN codes.

Results

You can now send requests from the z/OS Connect EE server via WOLA to CICS.

What to do next

To unregister the CICS region from WOLA and stop the Link Server task, open a terminal session with the CICS region and issue the following command:

BBOC STOP_SRVR RGN=<registerName>

where:

- <registerName> must match the **registerName** attribute value specified on the `localAdaptersConnectService` element in `server.xml`.

The following message indicates that the server was stopped:

WOLA TRACE 0: Stop server completed successfully.

To stop the WOLA TRUE, issue the following command:

BBOC **STOP_TRUE**

The following message indicates that the exit was stopped:

WOLA TRACE 0: Exit disabled Successfully

Using the IMS service provider

Create a service to enable RESTful access to IMS transactions through the IMS service provider, also known as the IMS service provider (imsmobile-2.0).

To set up the server to use the IMS service provider for the first time, take the following steps:

1. Check the [“Operational requirements” on page 238](#).
2. Review the [“IMS service security process flow” on page 238](#). Determine whether basic or SAF registry must be configured on the server for the initial setup. Identify if RACF is turned on in IMS Connect and if an IMS technical ID, technical group, and technical password need to be configured.
3. Follow the steps in [“Configuring for the IMS service provider” on page 243](#).

For steps to creating REST services to access IMS, see [“Create an IMS service” on page 89](#).

For a step-by-step example of creating a server and connecting it to IMS, see [“Create a server and connect to IMS” on page 72](#).

Operational requirements

Examine the operational requirements to assist the planning and setup of your IMS mobile solution.

- IMS Connect is required. The IMS service provider provides access to IMS transactions through the IMS Transaction Manager with IMS Connect being the TCP/IP gateway. The transactions can then access IMS DB, Db2, and other external subsystems. See [“IMS service security process flow” on page 238](#) for more information about user authentication and authorization from z/OS Connect EE to IMS.
- The z/OS Connect EE API toolkit V3 is required for creating IMS services and REST APIs to act on these services.

Important: IMS Explorer for Development is not compatible with z/OS Connect EE V3. You cannot connect to a z/OS Connect EE V3 server from IMS Explorer for Development. Service creation functions are now provided in z/OS Connect EE API toolkit.

IMS service security process flow

IMS Connect receives the user ID from the z/OS Connect EE server and handles user authentication if RACF setting is turned on.

The following diagram describes the general user authentication and authorization process.

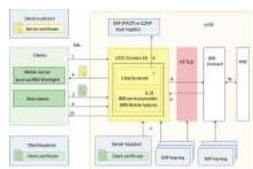


Figure 54. The IMS service security process flow and components involved

1. The client initiates an HTTPS call to z/OS Connect Enterprise Edition.
2. z/OS Connect EE is configured with SSL client authentication and fallback to basic authentication.

3. The client sends to z/OS Connect EE server a client certificate. All clients must send a basic authentication credential. The user ID and password in the credential must be registered in the basic registry or SAF registry on the server.
4. The server verifies the client certificate with the previously imported client certificate that is stored in the sever truststore or key ring. If the client certificate is missing, the server applies basic authentication against the user registry that was configured (SAF or LDAP).
5. The client begins transmitting data over a secure connection.
6. The z/OS Connect EE server authenticates the user credential. Then the server authorizes the user by using a SAF call to validate that the group names in the service configuration matches one of the group names associated with the user ID in the request subject.
7. After authentication and authorization, z/OS Connect EE passes the request to the IMS service provider. If authentication and authorization fail, an error is returned to the client.
8. The IMS service provider receives the service request, identifies the IMS connection profile and interaction profile to use, and processes the service input message.
9. The IMS service provider initiates a request to send the input bytes array and SAF information to IMS Connect. The HWSJAVA0 user exit on IMS Connect is used to manage the messages. The request triggers SSL handshake via AT-TLS, if it is configured, to protect the communication between z/OS Connect EE server and IMS Connect.
10. IMS Connect flows the request with the user ID to IMS. IMS might perform additional authorization, checking if the user can invoke the transaction. IMS transaction runs. IMS returns response (bytes) to IMS Connect.
11. IMS Connect returns response (bytes) to the IMS service provider.
12. The IMS service provider processes the response.
13. The response is returned to the client.

ID propagation

The user ID that is passed to IMS Connect could be the user ID from the originating client (or the RACF ID to which the user ID is mapped, depending on the setup on the z/OS Connect EE server), or the user ID from the IMS connection profile.

- When user authentication data is specified in the IMS connection profile that is used to connect to IMS Connect, all requests from this connection share this same user ID. The authentication data is specified in the **containerAuthDataRef** attribute, which points to an **authData** element with a specified user ID (the **user** attribute) and password (the **password** attribute).
- To pass the user ID from the originating client, ensure that your IMS connection profile does not specify the user ID and password for authentication. Do not specify the **containerAuthDataRef** attribute.

Passing shared user ID and password in the IMS connection profile to IMS Connect with RACF turned on

A common scenario is when users are connecting through a secure gateway, so the users are already authenticated before connecting to the z/OS Connect EE server. This scenario has the following setup:

- Authentication is turned off on z/OS Connect EE.
- RACF is turned on in IMS Connect, so a user ID and a password are required for authentication.

In this case, you can use the user ID and the password that are specified in the IMS connection profile. In the IMS connection profile, configure the **containerAuthDataRef** attribute and related **authData** element with the **user** and **password** attributes. All requests from the z/OS Connect EE server would share the same user ID.

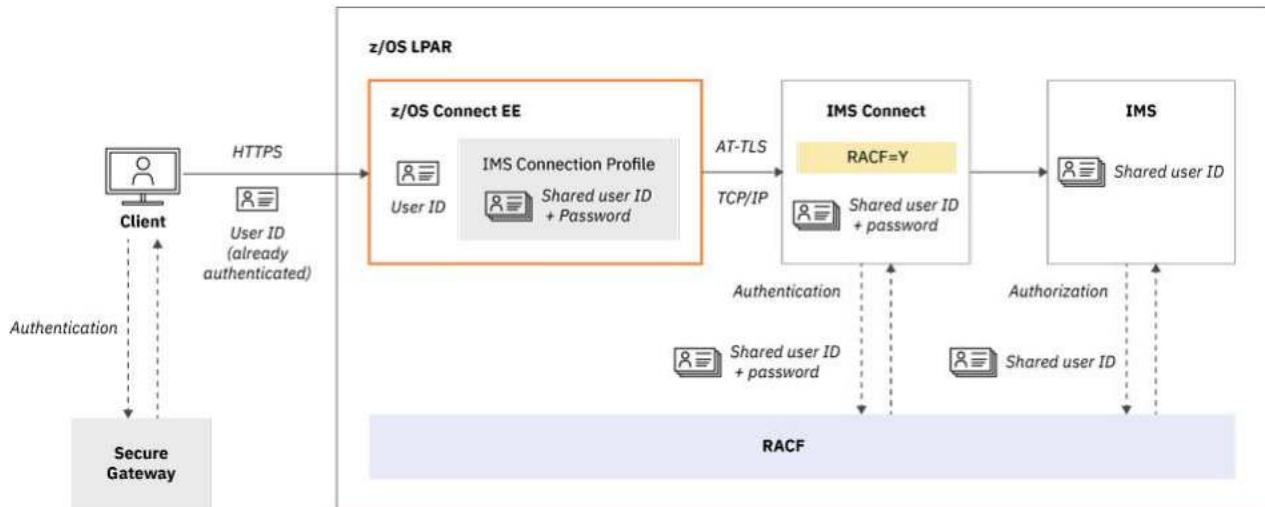


Figure 55. Passing the shared user ID in the connection profile to IMS Connect (RACF=Y)

Passing user ID from the originating client to IMS Connect with RACF turned on

The following diagram shows a common scenario where the initiating client ID is authenticated by the z/OS Connect EE server.

- z/OS Connect EE authenticates the user, and the user ID is mapped to a RACF ID.
- The IMS connection profile does not contain any authentication data (no **containerAuthDataRef** attribute) in order to pass the originating user ID or its mapped RACF ID to IMS Connect.
- A dedicated port (port 5050) is set up in IMS Connect for traffic from z/OS Connect EE. Because only the user ID (no password) is passed to IMS Connect, IMS Connect must be set up to trust the user that connects to the dedicated port by modifying the IMS HWSJAVA0 user exit for trusted users.
- The mapped RACF ID is then sent to IMS for authorization.

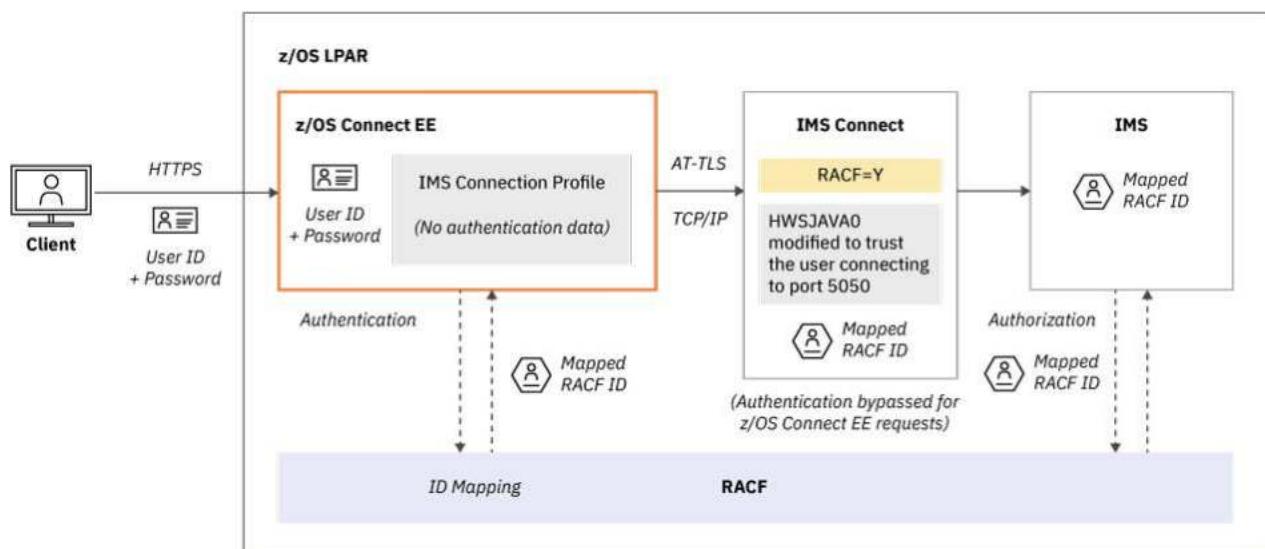


Figure 56. Passing the originating client ID to IMS and bypassing IMS Connect authentication (RACF=Y)

For more information about user authentication in IMS Connect and enabling trusted users, see “[User authentication in IMS Connect](#)” on page 252.

Passing user ID from the originating client to IMS Connect with RACF turned off

In proof-of-concept projects, it is common to turn off IMS Connect authentication (RACF=N) for ease of testing. The following diagram shows such a setup where the IMS connection profile does not contain the

containerAuthDataRef attribute. IMS Connect passes the user ID to IMS for authorization to determine if the user can invoke the transaction.

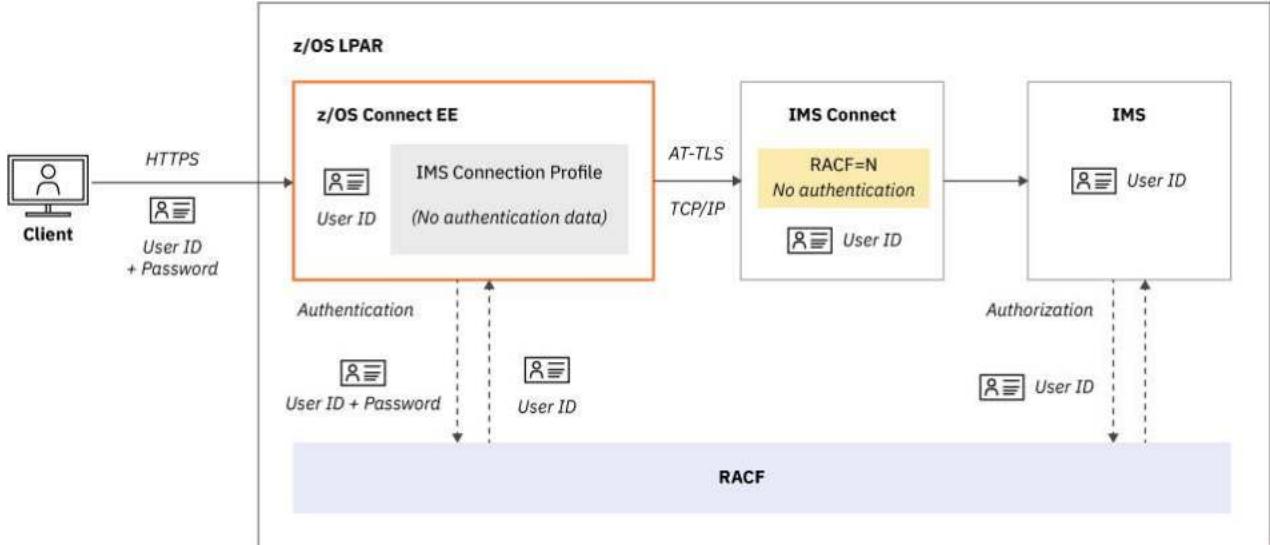


Figure 57. Passing the originating client ID to IMS with IMS Connect authentication turned off (RACF=N) in proof-of-concept projects

Network security credential propagation for IMS 15

When the IMS service provider detects that the IMS Connect it is connecting to is V15, it sends the user ID and its associated network session ID (realm) that are registered in the basic registry or SAF registry to IMS by default (controlled by the **propagateNetworkSecurityCred** IMS interaction property). IMS V15 adds the support for network security credentials. These credentials can be processed and stored for auditing and logging purposes; they are not used for authentication or authorization. For more information about IMS V15 network security credential propagation support, see [Network security credential propagation enhancement \(IMS 15 documentation\)](#).

Related concepts

[“IMS connection profiles” on page 247](#)

An IMS connection profile contains information about the IMS host name, port number, and other security related properties for connection to the specified IMS host system.

[“Security configuration for the IMS service provider” on page 250](#)

You must configure the z/OS Connect EE server and the back-end IMS to ensure secure communications.

Configuring PassTicket support for IMS services

Using PassTickets allows for identity propagation when invoking IMS services. A PassTicket is generated for a specific user ID and application, by z/OS Connect EE, and is then passed to IMS where it is validated.

Before you begin

The value of the **APPL** parameter in the HWSCFG member in IMS must match the **APPL** name that is defined to RACF in the PTKTDATA statement. The **APPL** value must also match the value of the **applicationName** attribute within the **properties.gmoa** element of the **connectionFactory** element in the **ims-connections.xml** configuration file for z/OS Connect EE.

For more information about configuring the DATASTORE statement, see [DATASTORE statement on the IMS documentation](#).

Note: PassTicket generation uses native services, and is therefore not eligible for zIIP processing,

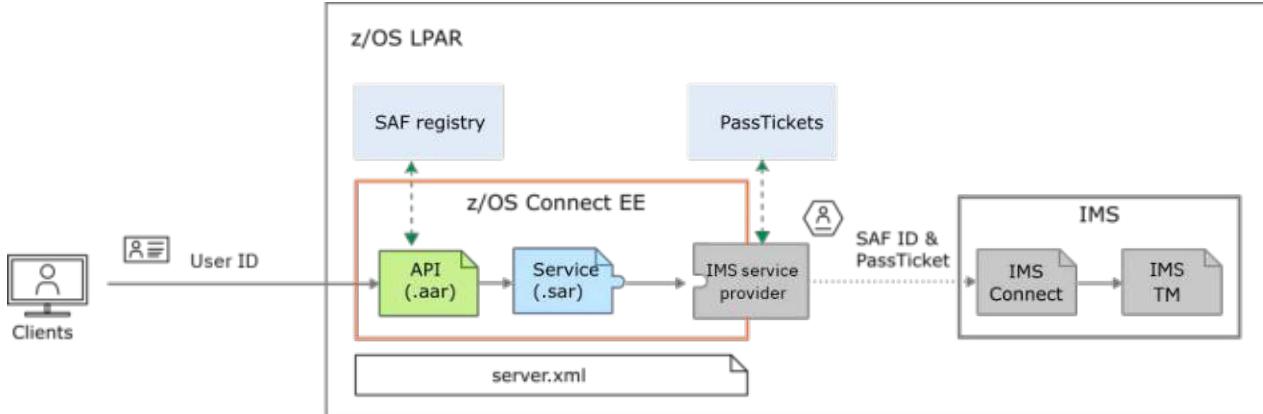
Before you run or test an IMS service that uses PassTicket support, you must first configure the **connectionFactory** element in **ims-connections.xml**.

For more information about how passTickets work on z/OS, see [Using PassTickets in the z/OS documentation](#).

About this task

If authentication is configured in `server.xml`, a user ID is included in the request to z/OS Connect EE.

The following diagram illustrates how to use PassTickets for identity propagation.



The user ID in the request can be a SAF user ID. If the user ID in the request is not a SAF user ID, then it can either be mapped to a SAF user ID, or the SAF user ID can be configured in the `userName` attribute of the `properties.gmoa` element in the `connectionFactory` element in `ims-connections.xml`. The SAF user ID is then propagated to IMS with the generated PassTicket, where further authentication is then carried out by IMS.

Note: When the z/OS Connect EE server and IMS are located in different LPARs that do not share the same RACF database, the commands in steps “1” on page 242, “2” on page 242, and “4” on page 242 must be entered in each LPAR. The commands in step “3” on page 242 need to be entered only on the LPAR in which the z/OS Connect EE server runs.

Procedure

1. From a TSO session, enter the following commands to activate the RACF PassTicket class:

```
SETROPTS CLASSACT(PTKTDATA) RACLIST(PTKTDATA) SETROPTS GENERIC(PTKTDATA)
```

2. Enter the following command to define RACF profiles for the application in PTKTDATA.

```
RDEFINE PTKTDATA <appName> SSIGNON(KEYMASKED(<key>)) APPLDATA('NO REPLAY PROTECTION')
```

- `<appName>` is the name of the application that requests and uses the PassTickets.
- `<key>` is a session key with the value of 16 hexadecimal digits (for an 8-bit or 64-bit key). The session key must be identical to the key in the PassTicket definition in each RACF instance. The key for each application must be the same on all systems in the configuration.

3. Enter the following commands to define RACF profiles that enable z/OS Connect EE PassTicket generation:

```
RDEFINE PTKTDATA IRRPTAUTH.<appName>.* UACC(NONE)  
PERMIT IRRPTAUTH.<appName>.* ID(<zosconnect_id>) ACCESS(UPDATE) CLASS(PTKTDATA)
```

where `<zosconnect_id>` is the user ID under which the z/OS Connect EE started task procedure runs.

4. Enter the following command to refresh the PTKTDATA class:

```
SETROPTS RACLIST(PTKTDATA) REFRESH
```

5. In `ims-connections.xml`, specify the following elements. The value of the `applicationName` attribute must match the `<appName>` value in step “2” on page 242.

The default location for `ims-connections.xml` is `resources/ims-mobile-config/connections/`

```

<imsmobile_imsConnection comments="IMS connection" connectionFactoryRef="ims_cf"
id="Connection1_CF" />
<connectionFactory id="connectionFactoryReference">
<properties.gmoa hostName="ims.example.com" portNumber="1080" userName="myUserId"
applicationName="appName"/>
</connectionFactory>

```

If the **userName** attribute is specified, then this user ID is used with the PassTicket, otherwise the authenticated user ID, the mapped SAF ID, or the technical ID in `server.xml` for the request is used.

Related tasks

["Configuring for the IMS service provider" on page 243](#)

To set up a new server to use the IMS service provider, use the provided template to set up the required configuration, including the IMS mobile service registry. If you have already set up a server (such as with the CICS service provider), or you have specific layout requirements for the XML file, manually configure the `server.xml` file.

Configuring for the IMS service provider

To set up a new server to use the IMS service provider, use the provided template to set up the required configuration, including the IMS mobile service registry. If you have already set up a server (such as with the CICS service provider), or you have specific layout requirements for the XML file, manually configure the `server.xml` file.

Before you begin

The product extensions directory and the required `imsmobile.properties` file must be created by using the **zconsetup install** utility script. Ensure that the product extensions directory exists and the required `imsmobile.properties` file is added by following the steps in ["Setting up the product extensions directory" on page 197](#).

About this task

Use the provided template to set up a server that is configured to use the IMS service provider. Instructions are also provided for manual configuration of the `server.xml` file.

Procedure

1. Ensure a server instance is set up and configured to use the IMS service provider.

- To set up a new server instance with the IMS service provider, run the **zosconnect** command in the `<installation_path>/bin/` directory with the `create` option and specify the `imsmobile` template.

```
zosconnect create <server_name> --template=imsmobile:imsDefault
```

You will get a confirmation that the server `<server_name>` is created.

- If you already have an existing server, manually update the `server.xml` file to use the `imsmobile-2.0` feature by adding the following configuration information in the `<featureManager>` section:

```

<featureManager>
  .
  .
  <feature>imsmobile:imsmobile-2.0</feature>
</featureManager>

```

2. Configure security. Uncomment the related tags and provide the appropriate values in the `server.xml` file.

Note: The Angel process must be started for client authentication.

- a) Uncomment the following tags for SSL and failover to basic authentication:

```

<!-- Define the SSL configuration
<ssl id="defaultSSLConfig" keyStoreRef="defaultKeyStore"
      trustStoreRef="defaultTrustStore" clientAuthentication="false" />
<!-- Define a keystore, use securityUtility to generate encoded password
<keyStore id="defaultKeyStore" password="" />
-->

<!-- Define a truststore, use securityUtility to generate encoded password
<keyStore id="defaultTrustStore" password="" />
-->

<!-- Fail over to basic authentication
<webAppSecurity allowFailOverToBasicAuth="true"/>
-->

```

Use the Liberty server **securityUtility** command (`securityUtility encode userID`) to generate the encoded password. This utility is located in the `<installation_path>/wlp/bin` directory.

For example, the uncommented security configuration might look as follows:

```

<!-- Define the SSL configuration -->
<ssl id="defaultSSLConfig" keyStoreRef="defaultKeyStore"
      trustStoreRef="defaultTrustStore" clientAuthentication="false" />

<!-- Define a keystore, use securityUtility to generate encoded password -->
<keyStore id="defaultKeyStore" password="{xor}PjMzbiw7KjE="/>

<!-- Define a truststore, use securityUtility to generate encoded password -->
<keyStore id="defaultTrustStore" password="{xor}PjMzbiw7KjE="/>

<!-- Fail over to basic authentication -->
<webAppSecurity allowFailOverToBasicAuth="true"/>

```

- Configure your SAF registry or basic registry by uncommenting the related sections in the `server.xml` file.

For more information about SAF or basic registry configuration, see [“User registries” on page 453](#).

3. Save your changes.

You can now start (or restart) the server to verify if you can use the IMS service provider.

4. Start the server, or restart the server if it is currently running.

Starting the server is usually done by running a started task based on the sample JCL `<hlq>.SBAQSAMP(BAQSTRT)`. For information about starting the server, see [“Starting and stopping z/OS Connect EE” on page 471](#).

A `GMOIG7777I` message is issued in the console and logged in the `messages.log` file in `<WLP_USER_DIR>/servers/<server_name>/logs`, indicating that the feature is initialized successfully.

Important: You would receive two warning messages, `CWWKG0011W` and `CWWKG0083W`, indicating that the port value "portNumber" is not a number. These warnings are as expected.

The initial server startup process adds a reference to the generated IMS connection profile template, `ims-connections.xml`. This file has the following entry:

```
<properties.gmoa hostName="hostName_or_IPAddress" portNumber="portNumber"/>
```

These warning messages simply mean that the default IMS connection profile in the template is not yet configured. A connection profile is only required when a service is invoked. This connection information can be configured when you create an IMS service.

What to do next

You can do a quick test to verify server communications to IMS by using the IMS PingService service. For more information, see [“Verifying server communication with IMS” on page 245](#).

Before you start creating an IMS service, see the following information:

- See “[Security configuration for the IMS service provider](#)” on page 250 for security-related configuration, such as setting up secure connections from z/OS Connect EE to IMS.
- See “[IMS connection profiles](#)” on page 247 and “[IMS interaction profiles](#)” on page 248 for more information about configuring these profiles for IMS services.

To learn about how to create an IMS service and an API, see the following IMS scenarios. Each scenario includes a tutorial with detailed steps.

- [“Create an IMS service” on page 89](#)
- [“Create an API to invoke the IMS phone book service” on page 131](#)

Related concepts

[“IMS connection profiles” on page 247](#)

An IMS connection profile contains information about the IMS host name, port number, and other security related properties for connection to the specified IMS host system.

Verifying server communication with IMS

Use the IMSPingService to verify that the IMS service provider can communicate with IMS Connect by specifying the hostname, port number, and IMS data store name.

About this task

The IMSPingService service is included as part of the IMS service provider that you can use to check if the IMS service provider is configured correctly. This service can also be used to verify that the IMS service provider can communicate with IMS Connect.

Procedure

Use the HTTP PUT method to invoke the provided IMSPingService service, and specify host name, the port number, and the IMS data store name.

```
https://hostname:port/zosConnect/services/IMSPingService?action=invoke&HOSTNAME=
my.ims.host.com&PORT=9999&DATASTORE=IMS1
```

The HOSTNAME, PORT, and DATASTORE parameters must be in all uppercase letters, and all three parameters must be specified. If one of the required parameters is missing or the parameter names are misspelled, the parameters are ignored, and the request is treated as a ping request to the z/OS Connect EE server. When these parameters are provided correctly, this service issues a **/DISPLAY OTMA** command to verify that the IMS service provider on the z/OS Connect EE server can access the IMS host system.

Results

You would receive the following response:

```
{
  ...
  "message": "The ping request for the IMS system \\\"HOSTNAME: 9.30.112.170, PORT:
9999,
DATASTORE: IMS1, RACF ID: admuser\\\" through z/OS Connect EE and the IMS service
provider
was successful."
}
  ...
```

What to do next

Before you start creating an IMS service and an API, see the following information:

- See “[IMS connection profiles](#)” on page 247 and “[IMS interaction profiles](#)” on page 248 for configuration information.
- See “[Security configuration for the IMS service provider](#)” on page 250 for security-related configuration, such as setting up secure connections from z/OS Connect EE to IMS.
- To start creating a service and an API, see the following IMS scenarios. Each scenario includes a tutorial with detailed steps.
 - [“Create an IMS service” on page 89](#).
 - [“Create an API to invoke the IMS phone book service” on page 131](#)

IMS mobile services and service registry

Each IMS mobile service (or, simply, IMS service) is a RESTful service with an associated IMS connection profile, an interaction profile, and one or two service interface files that define how the high-level data structures are mapped to the JSON schema. The *IMS mobile service registry* is where the connection profiles and interaction profiles for IMS services are stored.

The IMS connection profile includes information about the IMS host name, port number, data store name and other properties. The IMS interaction profile specifies how each service interacts with IMS, such as the commit mode and sync level. For more information, see [“IMS connection profiles” on page 247](#) and [“IMS interaction profiles” on page 248](#).

The service interface files are defined in the z/OS Connect EE API toolkit.

For each service, optional runtime properties can be specified in the `server.xml` file. The values specified in the `server.xml` file take precedence over the values that are stored in the service archive (`.sar`) file. For more information about IMS service runtime properties, see [Configuration elements \(zosconnect_services > service > property\)](#).

IMS mobile service registry

Upon the initial startup of the created server instance, the IMS mobile service registry is installed into the location that is defined in the `server.xml`. The location is defined in the `imsServiceRegistryHome` attribute in the `<imsmobile_imsserviceManager>` element.

The service registry default location is a `resources/` directory under the `{server.config.dir}` directory (the `<WLP_USER_DIR>/servers/<server_name>`) directory, with a typical location of `/var/zosconnect/servers/<server_name>/resources`). To customize the location, you must specify an absolute path, for example, `<imsmobile_imsserviceManager imsServiceRegistryHome="/usr/lpp/imsmobile/resources"/>`.

The service registry directory contains a subdirectory `imsmobile-config/` that contains the service registry artifacts with the following sub-directories:

Table 21. IMS mobile service registry structure

Directory	Description
<code>connections/</code>	This directory contains the <code>ims-connections.xml</code> file that stores the connection profiles. Each <code><imsmobile_imsConnection></code> tag contains a connection profile that has a unique ID, with associated connection information such as the IMS host name, port number, and connection timeout values. A connection profile can be shared by multiple services.
<code>interactions/</code>	This directory contains the <code>ims-interactions.xml</code> file that stores the interaction profiles. Each <code><imsmobile_interaction></code> tag contains a profile that has an interaction profile ID with associated interaction properties such as commit mode and sync level. An interaction profile can be shared by multiple services.

Related reference

[“Configuration elements” on page 753](#)

IMS connection profiles

An IMS connection profile contains information about the IMS host name, port number, and other security related properties for connection to the specified IMS host system.

IMS connection profiles are stored in the IMS mobile service registry, in the `ims-connections.xml` file in the `connections` directory. In a typical installation, the service registry is located at `/var/zosconnect/servers/<server_name>/resources/imsmobile-config/`. For more information about the IMS mobile service registry, see [“IMS mobile services and service registry” on page 246](#).

Each IMS connection profile is enclosed in a pair of the `<imsmobile_imsConnection>` `</imsmobile_imsConnection>` tags. Each connection profile references the ID of a connection factory that is defined in the `connectionFactory` tag. A connection factory contains the required information (user name and password) to connect to IMS Connect. If security is turned on in IMS Connect, the connection factory references the ID of an `authData` tag that contains the user name and encrypted password for authorization with IMS Connect.

Note: A connection factory (`connectionFactory`) cannot be shared by multiple connection profiles. An authorization data object (`authData`) cannot be shared by multiple connection factories.

Table 22. IMS connection profile: Attributes in the `imsmobile_imsConnection` element

Attribute	Description
<code>id</code>	Specify a unique ID for this connection profile. This is the connection profile value to enter in the z/OS Connect EE API toolkit when you configure the service properties.
<code>connectionFactoryRef</code>	Set this value to the ID of the <code>connectionFactory</code> element.
<code>comment</code>	Optional. Enter your comment for this connection profile.
<code>pingIMSConnectOnInvoke</code>	Ping IMS Connect before attempting to invoke transaction to ensure the connection retrieved from the connection pool is not stale. Throw an exception if z/OS Connect EE is unable to ping IMS before attempting to invoke service.

Table 23. IMS connection profile: Attributes in the `connectionFactory` element

Attribute	Description
<code>id</code>	Specify an ID for the connection factory.
<code>containerAuthDataRef</code>	Set this value to the ID of the <code>authData</code> element. If the user ID from the originating client is to be sent to IMS Connect, remove this attribute, and remove or comment out the <code>authData</code> element. Use this attribute and the corresponding <code>authData</code> element only if you want the user ID and password in the connection profile to be sent to IMS Connect.
<code>hostName</code> (in <code>properties.gmoa</code>)	Specify the host name or IP address of the data store server (IMS Connect).
<code>portNumber</code> (in <code>properties.gmoa</code>)	Specify the port number that is used to connect to IMS Connect.
<code>SSLEnabled</code>	Indicates if SSL is enabled for this connection factory.
<code>SSLEncryptionType</code>	Specifies the type of cipher suite to be used for encryption.

Table 23. IMS connection profile: Attributes in the connectionFactory element (continued)

Attribute	Description
SSLProtocol	The SSL protocol to be used for encryption.
SSLKeyStoreName	Name (full path) of SSL keystore for client certificates/private keys.
SSLKeyStorePassword	Password of SSL keystore for client certificates/private keys.
SSLTrustStoreName	Name (full path) of SSL keystore for trusted certificates.
SSLTrustStorePassword	Password of SSL keystore for trusted certificates.

Table 24. IMS connection profile: Attributes in the authData element

Attribute	Description
id	Specify an ID for this authData element
user	Specify the user name to use to connect to IMS Connect.
password	Specify the encrypted password for the specified user. Use a tool such as securityUtility in Liberty to encrypt plain text.

The following sample shows two connection profiles, each referencing a connection factory. Each connection factory references an authData object for authorization with IMS Connect.

```
<server>
  <imsmobile_imsConnection id="myConnectionID1"
    connectionFactoryRef="myConnection1_CF"
    comment=""
  />

  <connectionFactory id="myConnection1_CF" containerAuthDataRef="myConnection1_Auth">
    <properties.gmoa hostName="123.123.5.19" portNumber="5555"/>
  </connectionFactory>

  <authData id="myConnection1_Auth" user="Fred" password="{xor}DAsEbg0bBm4="/>

  <imsmobile_imsConnection id="myConnectionID2"
    connectionFactoryRef="myConnection2_CF"
    comment=""
  />

  <connectionFactory id="myConnection2_CF" containerAuthDataRef="myConnection2_Auth">
    <properties.gmoa hostName="123.123.5.22" portNumber="1111"/>
  </connectionFactory>

  <authData id="myConnection2_Auth" user="John" password="{xor}DAgsa4sSe2="/>
</server>
```

IMS interaction profiles

The IMS interaction profile specifies how each IMS transaction service interacts with IMS, such as the commit mode and sync level.

IMS interaction profiles are stored in the IMS mobile service registry, in the `ims-interactions.xml` file in the `interactions` directory. In a typical installation, the service registry is located at `/var/zosconnect/servers/<server_name>/resources/imsmobile-config/`. For more information about the IMS mobile service registry, see [“IMS mobile services and service registry” on page 246](#).

Each IMS interaction profile is enclosed in the `<imsmobile_interaction>` tag. Specify the values for the following attributes in the `<imsmobile_interaction>` tag.

Table 25. IMS interaction profile: Attributes in the `imsmobile_interaction` tag

Attribute	Description
<code>id</code>	Specify the ID for this interaction profile. This is the value to specify in the z/OS Connect EE API toolkit as the interaction profile name when configuring the service properties.
<code>commitMode</code>	Specify the commit mode. A value of 0 means commit-then-send (CM0); 1 means send-then-commit (CM1).
<code>imsConnectTimeout</code>	Specify the time in milliseconds to wait for a reply after sending a message to IMS Connect. Starting V3.0.2, the default is 30000, which means to wait for 30 seconds. Prior to V3.0.2, the default is 0, which means to wait indefinitely. Tip: General guidelines for setting the <code>imsConnectTimeout</code> value: <ul style="list-style-type: none"> • This value should be equal or larger than the value for <code>interactionTimeout</code>. • This value should be at least 5 seconds shorter than the value for the <code>asyncRequestTimeout</code> attribute of the <code>zosconnect_zosConnectManager</code> element in <code>server.xml</code>. This ensures that a successful transaction invocation won't be erroneously reported to the API client as a z/OS Connect EE <code>asyncRequestTimeout</code> failure. For more information, see “zosconnect_zosConnectManager” on page 784.
<code>imsDatastoreName</code>	Specify the name of the IMS data store (IMS Connect).
<code>interactionTimeout</code>	Specify the time in milliseconds for the transaction to be processed by IMS. After sending a message to IMS, IMS Connect waits for a reply from IMS until this timeout value is reached. <ul style="list-style-type: none"> • Valid values are -1, 0, or between 1 and 3600000 (one hour), inclusively. • A value of 0 means the timeout value is determined by IMS Connect. • A value of -1 (the default) means to wait indefinitely.
<code>tranExpiration</code>	Sets the TMRA <code>IMSInteractionSpec</code> property <code>transExpiration</code> . Accepted values for this attribute are “true” or “false”. To learn what these properties control, see the TMRA section of the IMS documentation .
<code>propagateNetworkSecurityCred</code>	Specify whether to propagate the network security credential if the IMS Connect is V15 or later. The default is true. The credential consists of the user ID and the network session ID (the realm) that are registered in the basic registry or SAF registry. For more information, see “ Configuring distributed identity propagation to IMS ” on page 253.
<code>syncLevel</code>	Specify the sync level. A value of 0 means None; 1 means Confirm. A <code>commitMode</code> value of 0 (CM0, Commit-then-send) is invalid with sync level 0 (None).
<code>imsConnectCodepage</code>	Specify the code page to use for character string conversion with IMS Connect. The default is Cp1047.

Table 25. IMS interaction profile: Attributes in the `imsmobile_interaction` tag (continued)

Attribute	Description
<code>ltermOverrideName</code>	Optional. Specify a LTERM name to override the value in the LTERM field of the IMS application program's I/O PCB.
<code>comment</code>	Optional. Enter comments about this interaction profile.

The following sample shows one IMS interaction profile in the `ims-interactions.xml` file.

```
<server>
  <imsmobile_interaction
    id="myInteractionProperties1"
    commitMode="1"
    imsConnectTimeout="0"
    imsDatastoreName="IMS1"
    interactionTimeout="-1"
    syncLevel="0"
    imsConnectCodepage="Cp1047"
    ltermOverrideName=""
    comment=""
  />
</server>
```

Security configuration for the IMS service provider

You must configure the z/OS Connect EE server and the back-end IMS to ensure secure communications.

Examine the “IMS service security process flow” on page 238 topic to gain an understanding of how user authentication and authorization are handled, and how the user ID and password for each service request are determined.

Table 26. Required security configurations

Area	Configuration task
z/OS Connect EE server	<p>See “Overview of z/OS Connect EE security” on page 331 and related security configuration topics for details.</p> <p>To configure for secure connections between z/OS Connect EE and IMS, see “Configuring secure connections between z/OS Connect EE and IMS Connect” on page 254.</p>
IMS services	<p>For service-level security, you can set the authority level of a user in the <code><ims_service_registry_home>/services/ims-services.xml</code> file. The following example sets, for the phonebook service, the administrator authority for users in ADMINGRP1 and the Invoke authority for users in USERGRP1.</p> <pre><zosconnect_zosConnectService id="phonebook" invokeURI="/imsmobile/services/phonebook" runGlobalInterceptors="true" adminGroup="ADMINGRP1" invokeGroup="USERGRP1" serviceDescription="" serviceName="phonebook" serviceRef="phonebook"/></pre> <p>See “Overview of z/OS Connect EE security” on page 331 for more information.</p>

Table 26. Required security configurations (continued)

Area	Configuration task
IMS service provider	<p>See “IMS service security process flow” on page 238 for general security process flow and user ID propagation.</p> <p>The IMS service provider also provides an optional attribute imsTechnicalID. If z/OS Connect EE authentication is turned off, or the request subject from which the user ID is extracted is more than 8 bytes, the IMS service provider retrieves the user from the technical ID that you specify in <code>server.xml</code>. Specify also the technical group and technical password in an <code><imsmobile_imsServiceManager></code> element in the <code>server.xml</code> file:</p> <pre><imsmobile_imsServiceManager imsServiceRegistryHome="/usr/lpp/imsmobile/resources" imsTechnicalGroup="IMSGROUP" imsTechnicalID="IMSUSER" imsTechnicalPassword="encoded_password"/></pre> <p>The IMS technical ID and IMS technical group name must be properly configured in SAF (for example, RACF) on the IMS host system.</p> <p>Use WebSphere Application Server Liberty Profile Developer Tools to configure the password and then transfer the file to the server by using FTP. Alternatively you can use the Liberty server securityUtility command (<code>securityUtility encode userID</code>) to generate the encoded password. Copy the encoded password into the <code>server.xml</code> file for the <code>imsTechnicalPassword</code> attribute. The following is a sample of an updated <code><imsmobile_imsServiceManager></code> element in the <code>server.xml</code> file:</p> <pre><imsmobile_imsServiceManager imsServiceRegistryHome=". /resources" imsTechnicalGroup="SYS1" imsTechnicalID="IMSGUEST" imsTechnicalPassword="{xor}PjMzbw7KjE=" ></pre>
IMS Connect	<p>IMS Connect handles authentication for IMS. The IMS Connect HWSJAVA0 exit routine manages the messages for the IMS service provider. If RACF is turned on in IMS Connect, the user ID and the password must be properly configured in RACF.</p> <p>If the user ID from the originating client is passed to IMS Connect, because z/OS Connect EE does not pass the password in the request, IMS Connect must be configured to trust the requests coming from z/OS Connect EE or RACF must be turned off. The HWSJAVA0 exit routine can be modified for trusted users. For more information, see “User authentication in IMS Connect” on page 252.</p> <p>To configure for secure connections between z/OS Connect EE and IMS, see “Configuring secure connections between z/OS Connect EE and IMS Connect” on page 254.</p>
Client applications	All client requests must provide basic authentication credentials in the header. The user ID and password in the credential must be registered in the basic registry or SAF registry on the server.

Important:

- Use of z/OS Communications Server Application Transparent Transport Layer Security (AT-TLS) SSL protection to secure the communication between the z/OS Connect EE server and IMS Connect is recommended.
- Optionally, you can turn on IMS security (**/SECURE OTMA**) for authorization that is based on RACF user ID in the request subject and the associated group name.

Enhancing security through these options are recommended. However, if you turn on IMS security, and you are using the basic user registry for user authentication in z/OS Connect EE server, you must use RACF ID as the user ID in your basic user registry in order for IMS OTMA to authorize the user.

Configuring AT-TLS for IMS Connect and OTMA

[Setting up AT-TLS SSL for IMS Connect](#)
[/SECURE OTMA command](#)

User authentication in IMS Connect

Configure IMS Connect and RACF (if RACF is turned on in IMS Connect) for proper user authentication. The HWSJAVA0 exit routine in IMS can be modified for custom security checking.

The user ID is determined based on the following rules:

- If authentication data (a user name and a password) is specified in the connection profile, the user name and the password are used for SAF authentication with IMS Connect.
- If no user name is specified in the connection profile:
 - The user ID is extracted from the request subject, and the subject must be 8 bytes or less in order for the IMS service provider to retrieve the user ID. The request subject does not contain the user password. If RACF is turned on in IMS Connect, configure the HWSJAVA0 user exit to bypass authentication for requests from z/OS Connect EE. If authentication is not bypassed, the IMS technical password is used as the SAF password for IMS Connect authentication.
 - If the request subject is more than 8 bytes, or authentication is turned off in z/OS Connect EE server, the IMS service provider retrieves the user ID from the technical ID, an IMS mobile global element that is specified during initial installation and setup. The IMS technical password is the SAF password for the user. If the technical ID is left blank, the IMS service provider uses the z/OS Connect EE started job user ID. The IMS technical password, if specified, is the SAF password for the user.
- The IMS technical groupname, if specified, is the SAF groupname.

RACF is turned on

If RACF is turned on in IMS Connect (RACF=Y in the HWS statement), user authentication is enabled for all its TCP/IP client connections. You must specify a user name and a password when you create a connection profile. You can create multiple connection profiles per server instance, each with its user ID and password. If no user ID is specified in the connection profile, ensure an IMS technical ID and a technical password are specified. This IMS technical password is used as the password for RACF authentication.

If RACF is turned on in IMS Connect, but you want to bypass the RACF call and trust connections through a designated port for traffic coming from z/OS Connect EE, modify the IMS Connect user message exit HWSJAVA0 for custom user security. You can add code to the exit to check the port value before marking it trusted. The parameter list passed to the user message exit is mapped by the HWSEXPRM macro. The EXPREA_section contains information about the information that is available to the exit, such as the client IP address (EXPREA_ADDRESS). The trusted user flag is in the HWSOMPFX macro.

See [“IMS service security process flow” on page 238](#) for common security scenarios when RACF is turned on in IMS Connect.

RACF is turned off

If RACF is turned off in IMS Connect, you can either specify a user name and a password in the connection profile (all services using the connection profile would share the same user name for IMS authorization), or leave the user name in the connection profile blank, in which case the user ID coming from the z/OS Connect EE server would be passed to IMS for authorization. Turning RACF off is a common setup in proof-of-concepts projects, as described in [“IMS service security process flow” on page 238](#).

Modifying HWSJAVA0 to enable trusted user support

The following sample code demonstrates how the HWSJAVA0 exit routine in IMS is modified to enable trusted user support in the READ function call when a user connects to IMS Connect through port 5050.

Note: Connections to IMS Connect currently support port values of 32767 or less.

```
USR_CHK DS 0H           CHECK USER DATA
         XR R4,R4          INIT FOR USER DATA
         TM OMCTLPFL,OMCTLUSR  USER DATA PRESENT
         BZ APL_CHK        ...NO, DERAULT ARCH LEVEL
*               LA R4,0(R5,R6)  AND CHECK APPL DATA
*               UD USING HWSOMUSR,R4  SAVE USER DATA ADDRESS
*****
* Determine what type of client should be trusted user *
* which will bypass RACF user authentication.          *
*****
* Example:                                           *
* If client is connecting to IMS Connect port 5050,   *
* then set it to be a trusted user.                   *
*****
LH   R14,EXPREA_LSTNPORT Get listening port
C    R14,=F'5050'      Is it port 5050?
BNE TRUST_X        No, skip trusted user
SETTRUST DS 0H
         OI UD.OMUSR_FLAG2,OMUSR_TRSTUSR
*               Set trusted user
TRUST_X DS 0H
*****
* IF ARCHITECTURE LEVEL OF 1 OR HIGHER, THEN IRM TIMER
* IS SUPPORTED BY IC4J (IC4J 2.2 OR HIGHER).
*****
CLC  UD.OMUSR_ARCLEV,OMUSR_AL01
*               ARCHITECTUAL LEVEL >= 1?
BNL  READ0040        ...YES, VALID FOR TIMER
```

For more information about trusted user support, see [Trusted-user support for IMS Connect messages](#).

For more information about the HWSJAVA0 exit routine, see [HWSJAVA0 user message exit routine](#) and a [HWSJAVA0 sample JCL](#).

Configuring distributed identity propagation to IMS

When the IMS service provider detects that the IMS Connect it is connecting to is IMS Connect V15 or later, by default, it sends the user ID and its associated network session ID (or realm) that are registered in the basic registry or SAF registry to IMS Connect.

About this task

The distributed identity, also known as the *network security credential*, includes a network user ID and a network session ID (or realm). This distributed identity is passed to IMS to be added to the IMS log records for auditing and logging purposes. It is not used by IMS for authentication or authorization.

The network user ID and network session ID are extracted from the z/OS Connect EE server user registry as defined in the `server.xml` file. In the following example, the network user ID is the user name that is registered in the basic registry, and the network session ID is the realm value.

```
<!-- Basic user registry definition -->
<basicRegistry id="basic1" realm="zosConnect">
  <user name="Fred" password="{xor}PjMzbiw7KjE=" />
  <user name="Rosa" password="{xor}LDo8Ki02KyY=" />
</basicRegistry>
```

If the IMS Connect it is connecting to is V14 or earlier, the IMS service provider does not pass the distributed identity to IMS Connect.

To turn off sending the distributed identity to IMS Connect V15 or later, set the `propagateNetworkSecurityCred` property in the IMS interaction profile to false (the default is true).

Related concepts

[“IMS interaction profiles” on page 248](#)

The IMS interaction profile specifies how each IMS transaction service interacts with IMS, such as the commit mode and sync level.

Related information

[Network security credential propagation enhancement \(IMS 15 documentation\)](#)

Configuring secure connections between z/OS Connect EE and IMS Connect

To secure TCP/IP connections to IMS Connect, ensure IMS security is properly defined using RACF and AT-TLS. Create a server key ring for IMS Connect with a server certificate and the necessary certificate authority (CA) certificates for server authentication.

About this task

When you configure secure connections to IMS Connect, IMS Connect is the server, and z/OS Connect EE becomes a client. For z/OS Connect EE to trust and authenticate IMS Connect (server authentication):

- On the subsystem where IMS Connect is run, create a server key ring to store the Certificate Authority (CA) root certificate and the server certificate that is signed by the CA private key. Then configure a server key ring and create the z/OS Communications Server Application Transparent Transport Layer Security (AT-TLS) policy files.
- On z/OS Connect EE, create a truststore and import the IMS Connect certificate.

The following diagram shows the server key ring that is required on IMS Connect and the keystore and truststore on z/OS Connect EE.

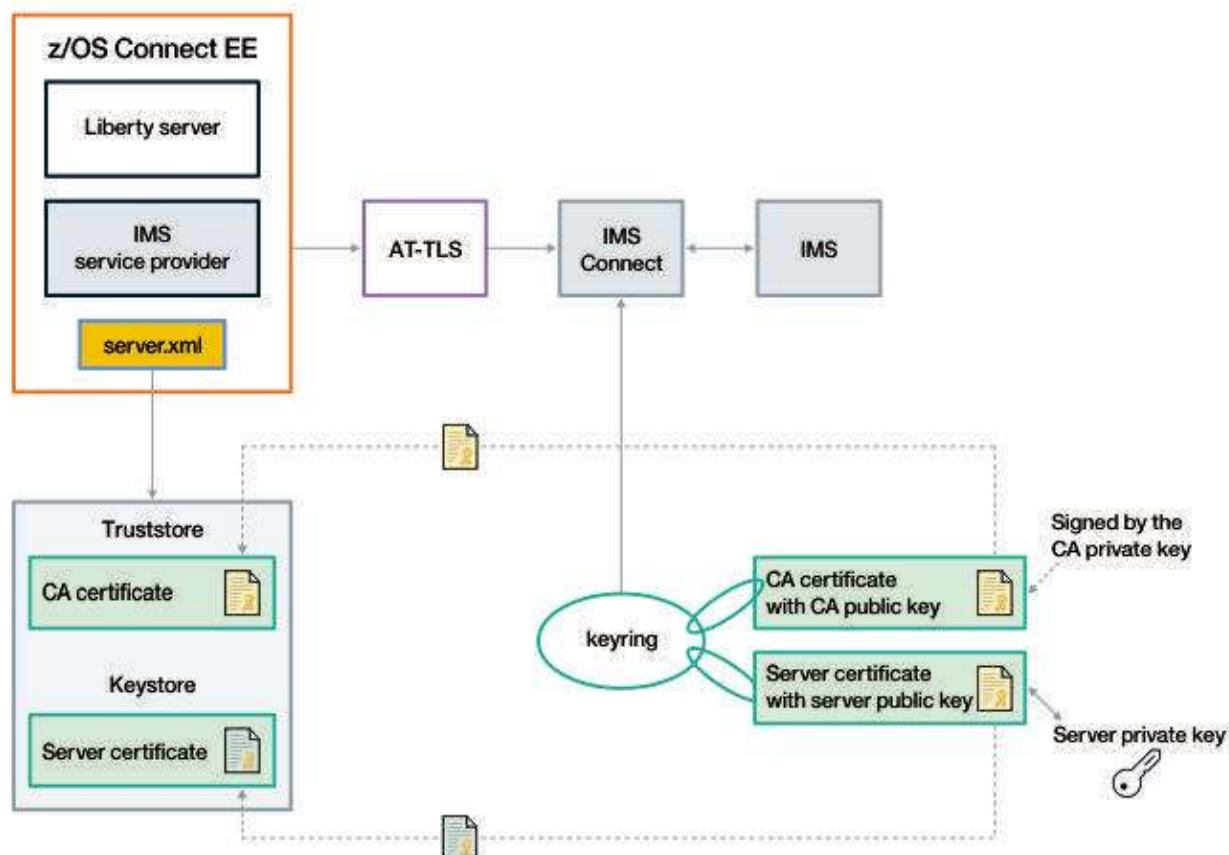


Figure 58. Secure connections with IMS Connect

If z/OS Connect EE is using a SAF key ring, then connect the server certificate to the key ring.

If client authentication is required, create a client certificate that identifies the client (z/OS Connect EE). This certificate is sent to the server (IMS Connect) during SSL handshake for the server to verify the client.

Procedure

The following information demonstrates the steps to set up SSL connections from z/OS Connect EE to IMS Connect by using AT-TLS

1. On each z/OS system where you run IMS Connect, create a server key ring with a server certificate and the necessary certificate authority certificates.
 - a) Create a server key ring to be used by the server (IMS Connect).

```
/*-----*/
/*  Create a key ring (ICON-KEYRING) to be used by          */
/*  IMS Connect.                                            */
/*-----*/
RACDCERT ADDRING(ICON-KEYRING) ID(OMVSADM)
```

- b) Create the Certificate Authority (CA) root certificate that is needed by the client (z/OS Connect EE) to authenticate the server (IMS Connect).

```
/*-----*/
/*  Create a certificate (IMSCA) to use as the Certificate   */
/*  Authority (CA), export to data set, and connect to        */
/*  the server key ring.                                     */
/*-----*/
RACDCERT CERTAUTH GENCERT -
SUBJECTSDN(CN('IMSCA') OU(IMS) O('IBM') C('US')) -
KEYUSAGE(CERTSIGN) WITHLABEL('IMSCA')
RACDCERT CERTAUTH EXPORT(LABEL('IMSCA')) DSN(CERTAUTH.CERT)
RACDCERT CONNECT(CERTAUTH LABEL('IMSCA') -
RING(ICON-KEYRING)) ID(OMVSADM)
```

- c) Create a server certificate that identifies the IMS Connect server. This certificate will be sent to the client during SSL handshake for the client to verify the server.

```
/*-----*/
/*  Create a server certificate signed by the CA (IMSCA),    */
/*  export to data set, and connect to the server key ring   */
/*  (ICON-KEYRING).                                         */
/*-----*/
RACDCERT GENCERT-
SUBJECTSDN(CN(ICON) OU('IMS') O('IBM') C('US'))-
WITHLABEL('CONNECT')-
SIGNWITH(CERTAUTH LABEL('IMSCA'))
RACDCERT EXPORT(LABEL('CONNECT')) DSN(ICON.CERT)
RACDCERT CONNECT(LABEL('CONNECT')) DEFAULT RING(ICON-KEYRING))
```

2. On z/OS Connect EE, connect the server certificate for IMS Connect to the key ring or store the server certificate in the truststore.

- If z/OS Connect EE is using key rings, connect the server certificate to the key ring. For example:

```
RACDCERT ID(ICON) CONNECT (RING(ZCEE.KEYRING.wsc) LABEL('ZCEE-Cert') CERTAUTH)
```

- If z/OS Connect EE is using keystores and truststores:

- a. Locate the truststore in the \${server.config.dir}/resources/security directory. If a truststore does not exist, create one using a tool such as the Ikeyman or the Java keytool.
- b. Store the IMS Connect CA certificate in the truststore.
- c. Ensure the truststore and keystore location is configured in the server.xml file.

```
<!-- Keystore definition (client certificate) -->
<keystore id="defaultKeyStore" password="{xor}NjIsMm89bjM6" location="ZCONN.P12"
type="PKCS12" />

<!-- Define a trust store (certificate authority) -->
<keystore id="defaultTrustStore" password="zosconnect" location="Zmytsts43_ts.jks" />

<!-- Define the SSL configuration -->
```

```
<ssl id="defaultSSLConfig" keyStoreRef="defaultKeyStore"
trustStoreRef="defaultTrustStore"
sslProtocol="SSL_TLSv2" />
```

3. If z/OS Connect EE is using RACF for user authentication, permit user IDs associated with the server to access the certificates.

```
/*-----*/
/* Permit user IDs associated with server and client to      */
/* access the certificates.                                     */
/*-----*/
SETROPTS RACLIST(DIGTCERT) REFRESH
RDELETE FACILITY (IRR.DIGTCERT.LIST    IRR.DIGTCERT.LISTRING)
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LIST   UACC(NONE)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(OMVSADM) ACCESS(ALTER)
PERMIT IRR.DIGTCERT.LIST    CLASS(FACILITY) ID(OMVSADM) ACCESS(ALTER)
SETROPTS CLASSACT(FACILITY)
SETROPTS RACLIST(FACILITY) REFRESH
//
```

4. Create policy agent files for AT-TLS.

When AT-TLS is enabled, the TCP/IP stack uses policies configured through the Policy Agent to apply SSL protection to TCP traffic based on the characteristics of that traffic (such as the local address and port, and the remote address and port).

For more information about configuring AT-TLS for IMS Connect, see [Setting up AT-TLS SSL for IMS Connect](#).

Related concepts

[“User authentication in IMS Connect” on page 252](#)

Configure IMS Connect and RACF (if RACF is turned on in IMS Connect) for proper user authentication. The HWSJAVA0 exit routine in IMS can be modified for custom security checking.

Transaction code override precedence

A transaction code must be specified during IMS service project creation. This transaction code can be overridden in the service interface definitions or at run time.

Precedence for transaction code overrides is in the following order from lowest to highest priority:

1. A transaction code must be specified in the service project editor during service project creation.
2. This transaction code can be overridden in the service interface editor, where the service interfaces for the request and response messages are defined, by specifying a default field value. This override information is stored with the service definition into the service archive (.sar) file.
3. To change the transaction code at run time, a value can be specified in the **imsTranCodeOverride** attribute for the service in the `server.xml` file, this value takes precedence over the transaction code information contained in the service definition. For more information about the **imsTranCodeOverride** attribute and other IMS service properties, see [Configuration elements \(zosconnect_services > service > property\)](#).
4. If a transaction code is passed in the JSON payload, this value overrides all transaction code specifications listed above.

Note: This override works only if the transaction code field is included in the service interface. It is not recommended to have the transaction code included in the service or API interfaces for an external production API service.

5. If an active policy contains a rule that modifies the transaction code, the modified value overrides all other transaction code specifications.

Related reference

[“Configuration elements” on page 753](#)

Using the IMS database service provider

Create a service to enable RESTful access to an IMS database through the IMS database service provider. Services are defined using service archive (.sar) files.

To set up the server to use the IMS database service provider for the first time, take the following steps:

1. Check the “[Operational requirements](#)” on page 257.
2. Review the “[IMS database service security process flow](#)” on page 258. Determine whether basic authentication or the SAF registry must be configured on the server for the initial setup. Identify if RACF is turned on in IMS Connect and if an IMS technical ID, technical group, and technical password need to be configured.
3. Determine if you must [configure PassTicket support for IMS database services](#) or [configure user type converter support for IMS database services](#).
4. Follow the steps in “[Configuring for the IMS database service provider](#)” on page 263.

For steps to creating REST services to access an IMS database, see “[Creating an IMS database service](#)” on page 543.

For a step-by-step example of creating a server, connecting it to an IMS database, and testing a database service, see “[Prepare the sample IMS database](#)” on page 62.

Operational requirements

Examine the operational requirements to assist the planning and setup of your IMS database service.

The IMS database service provider uses the IMS Open Database solution to establish a connection, and make requests to, an IMS database.

Important:

Currently, the following features are supported by the IMS database service provider:

- Create an IMS database service with a SQL SELECT query and a database connection profile by using the provided API toolkit. The following information must be prepared prior to using the API toolkit to create the IMS database service.
 - The SQL SELECT query you use for your IMS database service should be already tested in a tool such as IMS Explorer for Development. The API toolkit does not provide any SQL query development function. To learn more about using IMS Explorer for Development for preparing SQL queries, see [Creating and running SQL queries against an IMS database](#) in the *IMS Enterprise Suite Documentation*.
 - The database connection profile must be prepared in the defined XML format. A database connection profile template is provided.
- Retrieve service request and response schema by using the service administration interface.
- The service editor allows renaming, including and excluding, and setting default values for request and response fields.
 - If the request field is excluded, it must also have a default value specified.
 - If the request field is included, it can optionally have a default value specified. This default value is used when the request field is omitted from the input payload.
 - Default values for response fields are only used in cases when IMS returns null value for such fields.
- Deploy the IMS database service from within the API toolkit, and test the service by using a REST client.

Restriction:

There are multiple restrictions and requirements for the IMS database service provider. Examine the operational requirements, considerations and restrictions to assist the planning and design of your IMS database services.

IMS requirements

- Only IMS version 14 and version 15 are supported.
- The IMS catalog is properly configured.
- For the IMS database service provider, consider applying the following APARs to IMS:
 - IMS v15 PH15107/UI65936 or IMS v14 PH15106/UI65922 to ensure ODBM availability for database services under heavy load. To learn more see the [technical documentation for the IMS v15 APAR](#).
 - IMS v15 PH14651 or IMS v14 PH13796 to support the use of passphrases with IMS database services. To learn more, see the [technical documentation for the IMS v15 APAR](#).
 - IMS v15 PH19383 and IMS v14 PH19382 provide additional support for user defined types.
- Metadata for your IMS databases has been added to the IMS catalog. For more information about how to set up IMS catalog and prepare metadata for your databases, see the following resources:
 - The [IMS open access solution adoption kit](#) contains detailed information about how to set up IMS catalog with sample code.
 - [IMS GitHub](#) includes z/OSMF workflow samples for provisioning IMS catalog.
- Ensure that ODBM and ICON are running on IMS to allow z/OS Connect EE to communicate with an IMS database.

Limitations of use

- SQL INSERT, UPDATE, and DELETE commands are not supported.
- If a PSB or DBD that is used by an existing IMS database service is updated or changed, you must restart the z/OS Connect EE server to update the cached PSB or DBD used by the IMS Universal drivers.
- The use of z/OS Connect EE policies is not supported with IMS database service provider
- The following datatypes are currently unsupported for IMS database services:
 - Binary
- Services created with previous beta versions of the API toolkit which supported IMS database services are not compatible with this release.

IMS database service security process flow

z/OS Connect EE can be configured to securely communicate to IMS through IMS Connect. IMS Connect receives the user ID from the z/OS Connect EE server and handles user authentication if RACF setting is turned on.

The following diagram describes the general user authentication and authorization process. The numbers shown in the diagram correspond to the following steps.

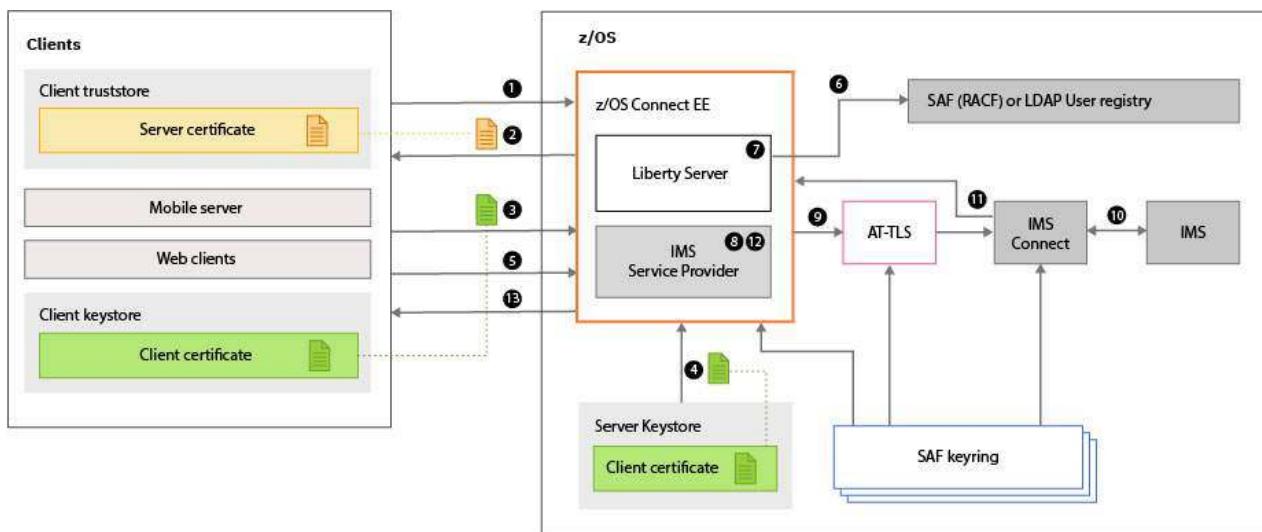


Figure 59. The IMS database service security process flow and components involved

1. The client initiates an HTTPS call to z/OS Connect Enterprise Edition.
2. z/OS Connect EE is configured with SSL client authentication and fallback to basic authentication.
3. The client sends to z/OS Connect EE server a client certificate. All clients must send a basic authentication credential. The user ID and password in the credential must be registered in the basic registry or SAF registry on the server.
4. The server verifies the client certificate with the previously imported client certificate that is stored in the sever truststore or keyring. If the client certificate is missing, the server applies basic authentication against the user registry that was configured (SAF or LDAP).
5. The client begins transmitting data over a secure connection.
6. The z/OS Connect EE server authenticates the user credential. Optionally, the server then authorizes the user by using a SAF call to validate that the group names in the service configuration matches one of the group names associated with the user ID in the request subject.
7. After authentication and authorization, z/OS Connect EE passes the request to the IMS database service provider. If authentication and authorization fail, an error is returned to the client.
8. The IMS database service provider receives the service request, identifies the IMS connection profile, and processes the service input message.
9. The IMS database service provider initiates a request to send the input bytes array and authentication information to IMS Connect. The HWSJAVA0 user exit on IMS Connect is used to manage the messages. The request triggers SSL handshake via AT-TLS, if it is configured, to protect the communication between z/OS Connect EE server and IMS Connect.
10. IMS Connect flows the request with the user ID to IMS DB. IMS might perform additional authorization, after which IMS returns response (bytes) to IMS Connect.
11. IMS Connect returns response (bytes) to the IMS database service provider.
12. The IMS database service provider processes the response.
13. The response is returned to the client.

ID propagation

The user ID that is passed to IMS Connect could be the user ID from the originating client (or the RACF ID to which the user ID is mapped, depending on the setup on the z/OS Connect EE server), or the user ID from the IMS connection profile.

- When user authentication data is specified in the IMS connection profile that is used to connect to IMS Connect, all requests from this connection share this same user ID. The authentication data is specified in the **user** and **password** attributes.
- To pass the user ID from the originating client, ensure that your IMS database connection profile does not specify the user ID for authentication.

Passing shared user ID and password in the IMS connection profile to IMS Connect with RACF turned on

A common scenario is when users are connecting through a secure gateway, so the users are already authenticated before connecting to the z/OS Connect EE server. This scenario has the following setup:

- Authentication is turned off on z/OS Connect EE.
- RACF is turned on in IMS Connect, so a user ID and a password are required for authentication.

In this case, you can use the user ID and the password that are specified in the IMS database connection profile. In the IMS database connection profile, configure the **user** and **password** attributes. All requests from the z/OS Connect EE server would share the same user ID.

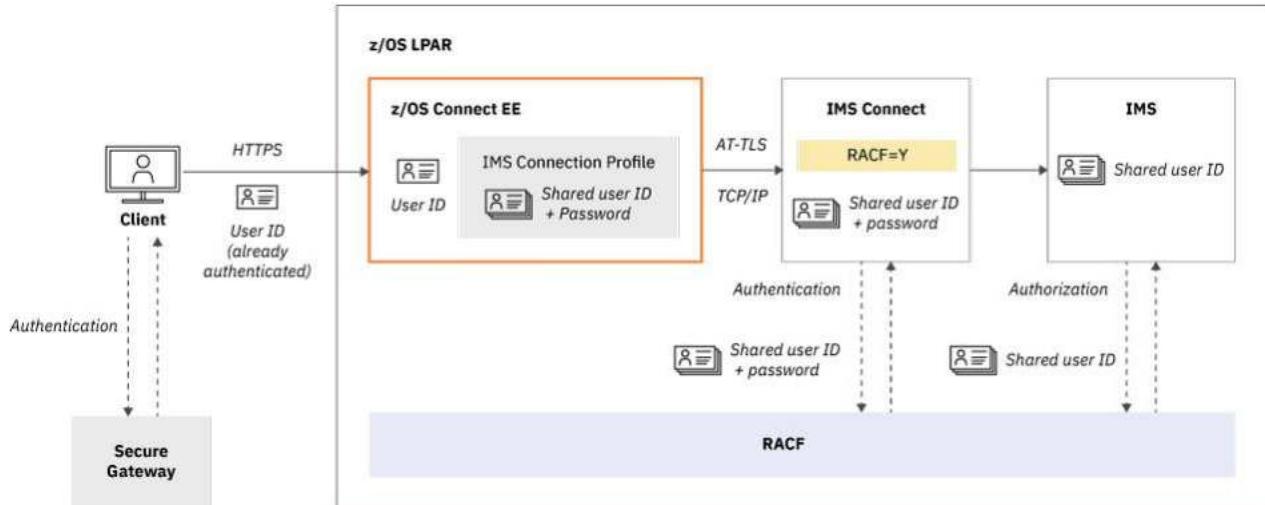


Figure 60. Passing the shared user ID in the connection profile to IMS Connect (RACF=Y)

Passing user ID from the originating client to IMS Connect with RACF turned off

In proof-of-concept projects, it is common to turn off IMS Connect authentication (RACF=N) for ease of testing. The following diagram shows such a setup where the IMS database connection profile does not contain the **user** and **password** attributes. IMS Connect passes the user ID to IMS for authorization to determine if the user can invoke the transaction.

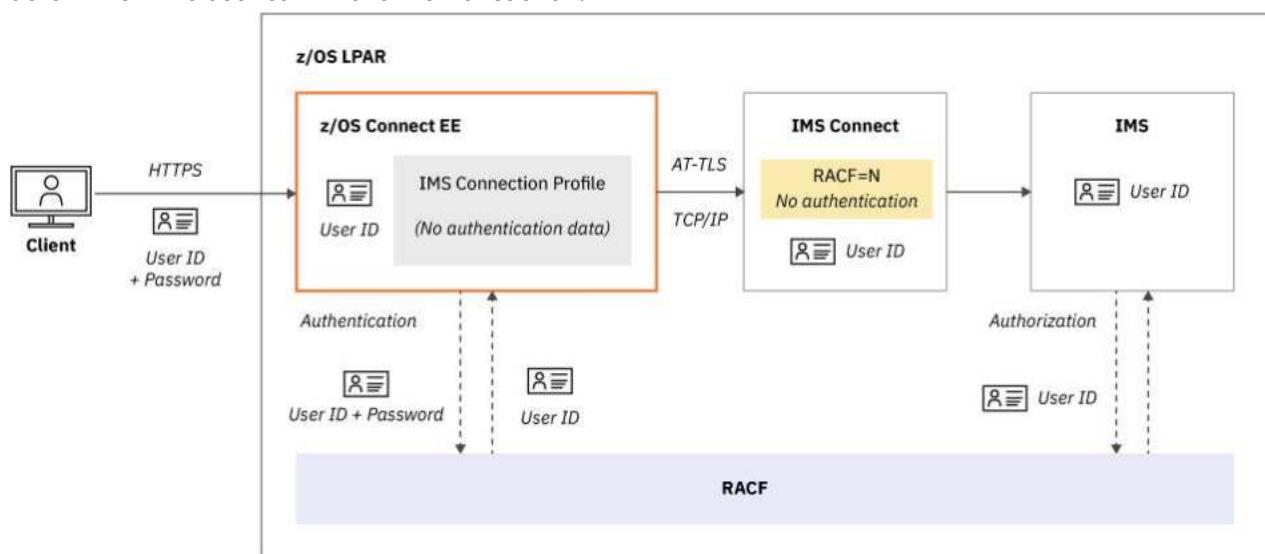


Figure 61. Passing the originating client ID to IMS with IMS Connect authentication turned off (RACF=N) in proof-of-concept projects

Related tasks

[“How to activate and configure the SAF user registry” on page 456](#)

Configuring PassTicket support for IMS database services

Using PassTickets allows for identity propagation when invoking IMS database services. A PassTicket is generated for the specific user ID and application, by z/OS Connect EE, and is then passed to the IMS database where it is validated.

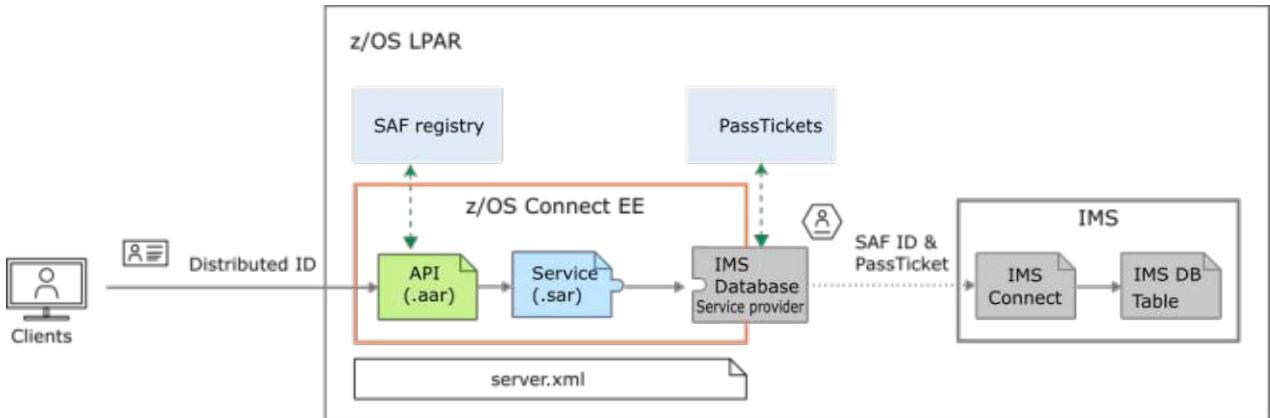
Before you begin

Before you run or test a database service that uses PassTicket support, you must first configure the `connectionFactory` element in `server.xml`. For more information, see “[Configuring for the IMS database service provider](#)” on page 263

Note: PassTicket generation uses native services, and is therefore not eligible for zIIP processing.

About this task

PassTicket generation can be used with any form of z/OS Connect EE supported authentication mechanism. The following diagram illustrates how you can use PassTickets for identity propagation when a distributed ID is used to authenticate to z/OS Connect EE.



A distributed ID that is included in the request to z/OS Connect EE, is then authenticated, and optionally mapped to a SAF user ID for further authentication. The user ID in the request, which can also be a SAF user ID, is then propagated to IMS with the generated PassTicket, where further authentication is then carried out by IMS. If a specific user ID is required for authentication with IMS for all requests, then this user ID can be configured in the **user** attribute of your `server.xml` configuration file.

Note: When the z/OS Connect EE server and the IMS database are located in different LPARs that do not share the same RACF database, the commands in steps “1” on page 261, “2” on page 261, and “4” on page 261 must be entered in each LPAR. The commands in step “3” on page 261 need to be entered only on the LPAR in which the z/OS Connect EE server runs.

Procedure

- From a TSO session, enter the following commands to activate the RACF PassTicket class:

```
SETROPTS CLASSACT(PTKTDATA) RACLIST(PTKTDATA) SETROPTS GENERIC(PTKTDATA)
```

- Enter the following commands to define RACF profiles for the application in PTKTDATA:

```
RDEFINE PTKTDATA <appName> SSIGNON(KEYMASKED(<key>)) APPLDATA('NO REPLAY PROTECTION')/
```

where

- <appName> is the name of the application that requests and uses the PassTickets.
- <key> is a session key with the value of 16 hexadecimal digits (for an 8-bit or 64-bit key). The session key must be identical to the key in the PassTicket definition in each RACF instance. The key for each application must be the same on all systems in the configuration.

- Enter the following commands to define RACF profiles for PassTicket generation:

```
RDEFINE PTKTDATA IRRPTAUTH.<appName>.* UACC(NONE)  
PERMIT IRRPTAUTH.<appName>.* ID(<zosconnect_id>) ACCESS(UPDATE) CLASS(PTKTDATA)
```

where <zosconnect_id> is the z/OS Connect EE user ID that is used for authentication with IMS.

- Enter the following command to refresh the PTKTDATA class:

```
SETROPTS RACLIST(PTKTDATA) REFRESH
```

- Define the **applicationName** attribute in the `connectionFactory` element in `server.xml`. The value of the **applicationName** attribute must match the `<appName>` value in step “2” on page 261. For example,

```
<connectionFactory id="myIMSDBConn">
<properties.imsdbJLocal
  databaseName="BMP255"
  datastoreName="IMS1"
  datastoreServer="myServer.ibm.com"
  driverType="4"
  portNumber="5555"
  user="myUserid"
  password="password"
  flattenTables="True"
  applicationName="imsAppl"/>
</connectionFactory>
```

If the **user** attribute is specified, then this user ID is used with the PassTicket, otherwise the authenticated user ID for the request is used.

Related tasks

[“Configuring for the IMS database service provider” on page 263](#)

Related information

Configuring user type converter support for IMS database services

User type converters are an extensible method of automatically transforming between data types from within the IMS Universal JDBC driver on behalf of IMS database services. If your IMS database service fields reference an IMS database that uses custom user types, configuring z/OS Connect EE for user type converters is required.

Before you begin

- Before running or testing a database service that utilizes user type converter support, you must first configure the `connectionFactory` element in `server.xml`. For more information, see [“Configuring for the IMS database service provider” on page 263](#).

Note: User type converters, and their implementation, are described in more detail on the *IMS product documentation*. To learn more, see [Data transformation support for JDBC](#).

About this task

Note: Using custom user type converters with an IMS database service requires additional elements to be defined in `server.xml`.

Procedure

- Create a Java project in your development environment, such as Eclipse, and copy your existing user type converter Java code into the project.
Save your Java project.
- Export a `.jar` of the project, and upload the file via FTP to the z/OS Connect EE server in a location of your choice.
Remember the location you choose to upload your `.jar` file to, as it is required for step 3.
- Add the following `zosconnect_dbServices` element definition to your `server.xml`:

```
<zosconnect_dbServices imsDbUtcPaths="/<yourFilePath>/<yourFileName>.jar:/<yourFilePath2>/<yourFileName2>.jar"/>
```

Where `yourFilePath` is the location of your `.jar` Java project, and `<yourFileName>` is the name of the `.jar` Java project that you exported in step 2.

4. Restart the z/OS Connect EE server.

For more information about restarting the z/OS Connect EE server, see [“Starting, stopping, or removing a service” on page 529](#).

Results

Upon running and testing your IMS database service, IMS will automatically determine and use the required type converter for the IMS database service that is called.

Related tasks

[“Configuring for the IMS database service provider” on page 263](#)

Configuring for the IMS database service provider

You must create a server instance and configure the `server.xml` file to use the IMS database service provider.

Before you begin

Ensure that the following considerations are made before configuration:

- If you plan to use a high availability database, see [“IMS and IMS Connect high availability” on page 321](#) for additional configuration information.
- If your database service will require data types to be transformed before data is processed by z/OS Connect EE, see [“Configuring user type converter support for IMS database services” on page 262](#).

About this task

Use the provided instructions to set up a server that is configured to use the IMS database service provider.

Procedure

1. Complete the following steps before you create and start your first z/OS Connect EE server instance.

- a) Create the shared directory. Follow the steps in [“Creating the z/OS Connect EE shared directory” on page 196](#).
- b) Set up the product extensions directory. Follow the steps in [“Setting up the product extensions directory” on page 197](#).

2. Set the `WLP_USER_DIR` environment variable to the location where you want your server instances and user features to be stored.

For example: `/var/zosconnect`.

You can specify the `WLP_USER_DIR` environment variable in the shell environment. This variable must be an absolute path. If you need help with this step, see [Creating a z/OS connect EE server](#).

3. Run the `zosconnect` command with the `create` option.

Use the following syntax:

```
zosconnect create <server name>
```

A z/OS Connect EE server instance is created in the `<WLP_USER_DIR>/servers` directory with a `server.xml` configuration file that defines the z/OS Connect EE feature.

The following steps walk through configuring the IMS database service provider feature and at least one connection profile for routing the database service request in the `server.xml` file.

4. Open the `server.xml` file in a text editor.

5. Add `<feature>zosconnect:dbService-1.0</feature>` to the `<featureManager>` element.

```
<featureManager>
  <feature>appSecurity-2.0</feature>
  <feature>osgiconsole-1.0</feature>
  <feature>zosconnect:zosconnect-2.0</feature>
```

```
<feature>zosconnect:dbService-1.0</feature>
</featureManager>
```

6. Add a <connectionFactory> element, with a nested <properties.imsudbJLocal> element that defines the database, data store, data store server, and port number to access. Modify the values based on your environment.

Other attributes can be specified in the <connectionFactory>, such as containerAuthDataRef, if needed.

```
<connectionFactory id=<connection_id>>
<properties.imsudbJLocal
  databaseName=<database_name>
  datastoreName=<datastore_name>
  datastoreServer=<serverName_or_IPAddress>
  driverType="4"
  portNumber=<port_number>
  user=<user_name>
  password=<password>
  flattenTables="True"/>
</connectionFactory>
```

The following example demonstrates what a completed <connectionFactory> element may look like in `server.xml`:

```
<connectionFactory id="myIMSDBConn">
<properties.imsudbJLocal
  databaseName="BMP255"
  datastoreName="IMS1"
  datastoreServer="myServer.ibm.com"
  driverType="4"
  portNumber="5555"
  user="myUserid"
  password="password"
  flattenTables="True"/>
</connectionFactory>
```

For an explanation of database service connection profile attributes for <connectionFactory>, see [Connecting to an IMS database by using the JDBC DriverManager interface](#).

The following table contains the most commonly used connection profile attributes used for configuring IMS database services:

Table 27. Database service connection profile attributes for <properties.imsudbJLocal>

Attribute	Description
applicationName	Specifies the 1- to 8-character application name that is defined to RACF® in IMS that is used by z/OS Connect EE to authenticate an IMS database service. The value that is specified on this parameter is used by z/OS Connect EE to generate a RACF PassTicket to authenticate the IMS database service to access IMS DB.
databaseName	The name of the PSB that your application uses to access the target IMS database.
datastoreName	The IMS datastore alias (IMSA) name that is specified on the DDM message received by IMS Connect. This alias is defined in the CSLDCXXX IMS PROCLIB member.
datastoreServer	The fully qualified name or the IP address of the data store server.
driverType	The JDBC driver connection type. Always set this attribute to 4.

Table 27. Database service connection profile attributes for <properties.imsudbJLocal> (continued)

Attribute	Description
portNumber	The TCP/IP server port number to be used to communicate with IMS Connect.
user	The user name for the connection to IMS Connect provided by your RACF® administrator.
password	The password for the connection to IMS Connect provided by your RACF® administrator.
flattenTables	Indicates whether to produce a flattened view of the database tables containing STRUCT or ARRAY fields. Set the value to "true" to expose the sub-elements of a STRUCT or an ARRAY as additional columns of the table. For more information on flattening tables in IMS, see the <i>IMS product documentation</i> topic Support for flattening complex structures .

7. Save your changes.

You are now ready to start, or restart, your server that is configured to use the IMS database service provider.

8. To start the server, see [Starting and stopping z/OS Connect EE](#).

Results

The z/OS Connect EE server is now configured to run IMS database services. You are now ready to connect to an [IMS database](#).

Related concepts

["IMS and IMS Connect high availability" on page 321](#)

When IMS assets serve as REST API endpoints through the IMS or IMS Database service provider, the first place to configure for IMS Connect high availability is in the TCP/IP network that sends and receives messages.

Creating an IMS database host connection

Create a host connection from the API toolkit to the IMS Connect ODBM port, facilitating a connection between IMS and an IMS database service.

Before you begin

Important:

The IMS database host connections created in the following steps are used during [creation of an IMS database service](#) to connect to an IMS database and collect metadata from the IMS catalog.

In z/OS Explorer, switch to the **z/OS Connect Enterprise Edition** perspective.

About this task

Using the **Host Connections** view, you can add IMS database connections and credentials to store your user IDs and passwords.

Tip: If you don't see the **Host Connections** view, from the menu bar, click **Window > Manage Connections** to open the **Host Connections** view.

Procedure

1. In the **Host Connections** view, click **Add** and select **IMS database**.

Alternatively, if you already have IMS database connections specified, highlight **IMS database** in the **Host Connections** view and then click **add**.

2. Specify the following information:

Table 28. Adding a server connection

Field	Description
Name	A descriptive name for the server connection.
Host name	The name or the IP address of the server.
Port number	The port number. Select the Secure connection (TLS/SSL) checkbox for secure connections.
Connection timeout	The amount of time in milliseconds that the API toolkit will wait for a successful connection to be established with IMS before timing out. The default timeout period is 30 seconds, unless specified otherwise in the z/OS Connect EE API toolkit preferences window. A value of 0 indicates to wait forever.
Read timeout	The amount of time in milliseconds that the API toolkit will read response data from IMS before timing out. The default timeout period is 30 seconds, unless specified otherwise in the z/OS Connect EE API toolkit preferences window. A value of 0 indicates to wait forever.

Tip: You can change the default connection timeout and read timeout values. From the main menu bar, click **Window > Preference > z/OS Connect EE**, and specify your default timeout values.

3. Click **Save and Connect** to save the definition and connect to an IMS database.

You will be prompted to either specify an existing credential to use, or create a credential at this time. To add a credential, click **Add** in the **Credentials** section. For more information, see [Defining connection credentials in IBM Explorer for z/OS documentation](#).

4. After you select an existing credential or create a new one, click **OK** to use the credential to connect.

Results

The defined host connection to your IMS database appears in the **z/OS Connect EE Servers** view. Deployed API packages are listed under the **APIs** folder. You are now prepared to [create an IMS DB service](#).

Using the IBM MQ service provider

The IBM MQ service provider that is supplied with z/OS Connect EE allows REST aware applications to interact with z/OS assets that are exposed by IBM MQ for z/OS queues and topics without the need to use asynchronous messaging APIs.

All the capabilities of IBM z/OS Connect EE to expose APIs, are supported by the IBM MQ service provider.

Setting up the IBM MQ service provider consists mainly of configuration tasks. The REST application does not need to be aware of IBM MQ. However, access to items such as MQMD fields is provided for more advanced applications.

The IBM MQ service provider exposes IBM MQ queues and topics as services. For more information, see [“Service types” on page 268](#).

Supported versions of IBM MQ

The IBM MQ service provider is supported by IBM MQ for z/OS Version 8.0 and later.

The IBM MQ service provider was originally included as part of IBM MQ 9.0.1, and is still available as part of IBM MQ 9.1.0. Customers who are using this earlier version of the IBM MQ service provider should upgrade to the version that is built into z/OS Connect EE. For more information about migrating, see ["Migrating a service to the IBM MQ service provider in z/OS Connect EE" on page 280](#).

This documentation refers to the IBM MQ service provider that is built into z/OS Connect EE. For more information about the IBM MQ service provider that is included with IBM MQ 9.1.0, see [IBM MQ for z/OS Service Provider for IBM z/OS Connect EE](#) in the *IBM MQ* documentation.

Note: The MQ Service provider is supplied with z/OS Connect EE v3.0.21 and later. For earlier versions, you can use the service provider that is supplied with IBM MQ. For more information, see [IBM MQ for z/OS Service Provider for z/OS Connect EE](#) in the *IBM MQ* documentation.

Connection modes

The IBM MQ service provider uses the IBM MQ classes for JMS (Java Message Service) to connect to a queue manager. These classes provide two connection modes:

Bindings mode

In bindings mode, connections are made directly to the queue manager by using the Java Native Interface (JNI).

Bindings mode connections tend to provide better performance than client mode but require the z/OS Connect EE server and IBM MQ queue manager to be running in the same LPAR.

In bindings mode, you must enable the TXRRS authorized service. For more information, see ["Configuring the Liberty Angel process and z/OS authorized services" on page 465](#) and the sample scenario ["Create a server to connect to IBM MQ" on page 77](#).

Client mode

In client mode, connections to the queue manager are made over TCP/IP.

Client mode connections allow greater flexibility than bindings mode as the IBM MQ queue manager can be running on a different z/OS LPAR to the z/OS Connect EE server. However, they require more configuration, which might include setting up TLS. Client mode connections typically do not perform as well as bindings mode connections.

Both connection modes are supported by the IBM MQ service provider, regardless of the version of MQ for z/OS that they connect to.

Important: The IBM MQ service provider does not support client connections to distributed versions of IBM MQ. It supports client connections to IBM MQ for z/OS only.

For the IBM MQ service provider, the connection mode is configured by using the `transportType` property in the `properties.wmqJms` section of the connection factory in the `server.xml` file of the z/OS Connect EE server:

- A `transportType` of `BINDINGS` allows bindings mode connections.
- A `transportType` of `CLIENT` allows client mode connections.

If client mode connections are required, you also need to set the `channel`, `hostName`, and `port` properties and if TLS is being used, the `SSL` attributes. For more information about these properties, see [WebSphere MQ JMS queue connection factory properties](#) in the *WebSphere Application Server Liberty* documentation.

Feature configuration

The following feature must be included in the `server.xml` configuration for the z/OS Connect EE server.

```
<feature>zosconnect:mqService-1.0</feature>
```

By configuring the feature `zosconnect:mqService-1.0`, any dependent features are automatically included. For more information, see the task [“Create a server to connect to IBM MQ” on page 77](#) in the example scenario.

Other considerations

The IBM MQ service provider uses optional HTTP headers to specify overrides, such as MQMD values.

IBM MQ messages consists of both payload and properties. When HTTP requests are sent to the IBM MQ service provider the JSON content of the request is sent as the payload of a message. When a message is returned as part of an HTTP response, only the message payload is returned as part of the HTTP response body. Therefore, applications are not exposed to IBM MQ concepts if all they need is the message data.

The IBM MQ Service Provider uses the IBM MQ resource adapter support in WebSphere Application Server Liberty profile and is based on JMS.

Service types

The IBM MQ service provider exposes IBM MQ queues and topics, and the applications that are behind them, as services.

Services can be one-way or two-way and are created by using the API toolkit. For more information, see [“Build a service archive with the build toolkit” on page 167](#). After a service is created using the API toolkit, the build toolkit can then be used to create this service for use in a DevOps workflow environment.

Services can be created by one of the following methods:

1. Use the API toolkit. This method is preferred, because it provides enhanced capabilities such as enabling MQ messages to be greater than 32K. When a service is created with the API toolkit, it can later be built by the build toolkit. For more information, see [“Build a service archive with the build toolkit” on page 167](#).
2. Manually create a properties file that describes the service. The properties file is then used by the build toolkit to create the service.
3. Configure the `zosConnectService` and `mqService` elements in the `server.xml` file. This method is supported only for compatibility with the IBM MQ service provider that was supplied with IBM MQ. If you use services that were created with this method, consider migrating them to use the API toolkit method.

One-way service

A one-way service can be used to provide a RESTful API for a single IBM MQ queue or topic and supports the sending of messages to either a queue or a topic, or receiving messages from a queue.

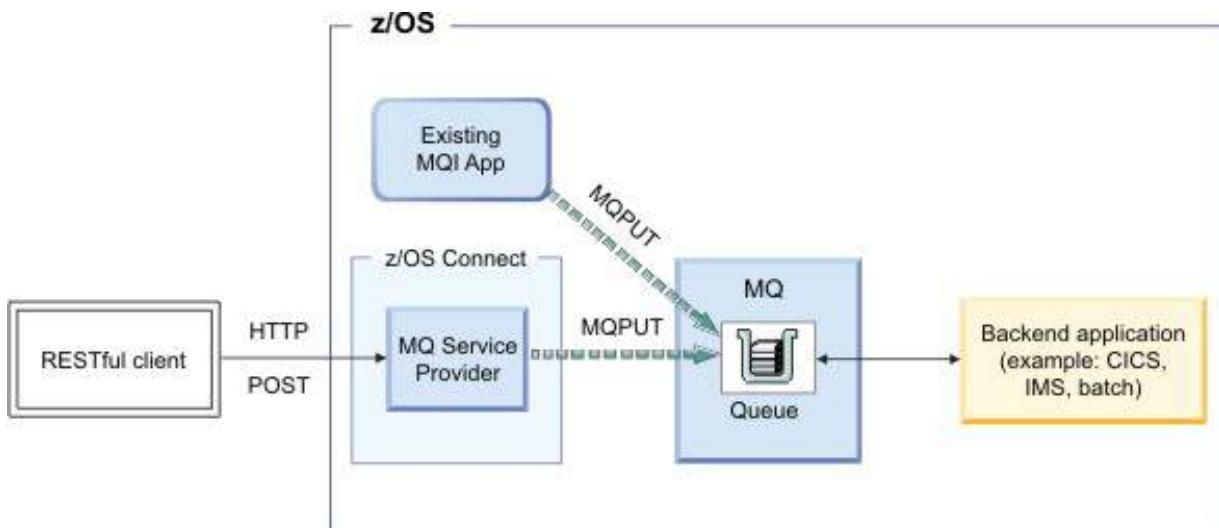


Figure 62. One-way service

One-way services for sending messages can be created in the API toolkit by creating a service project of type MQ One-Way Service for Sending Messages, or by using a properties file with a messagingAction value of mqput. When a RESTful client sends an HTTP request with a JSON payload to a service of this type, the service sends a message with the payload as the message body, to the target queue or topic.

One-way services for receiving messages can be created in the API toolkit by creating a service project of type MQ One-Way Service for Receiving Messages, or by using a properties file with a messagingAction value of mqget. When a RESTful client sends an HTTP request with no payload to a service of this type, the service attempts to destructively get an available message from an IBM MQ queue. If a message was available, the payload of the message is returned as the response body of the HTTP request.

Two-way service

A two-way service allows a RESTful client to perform request-reply messaging on a pair of queues.

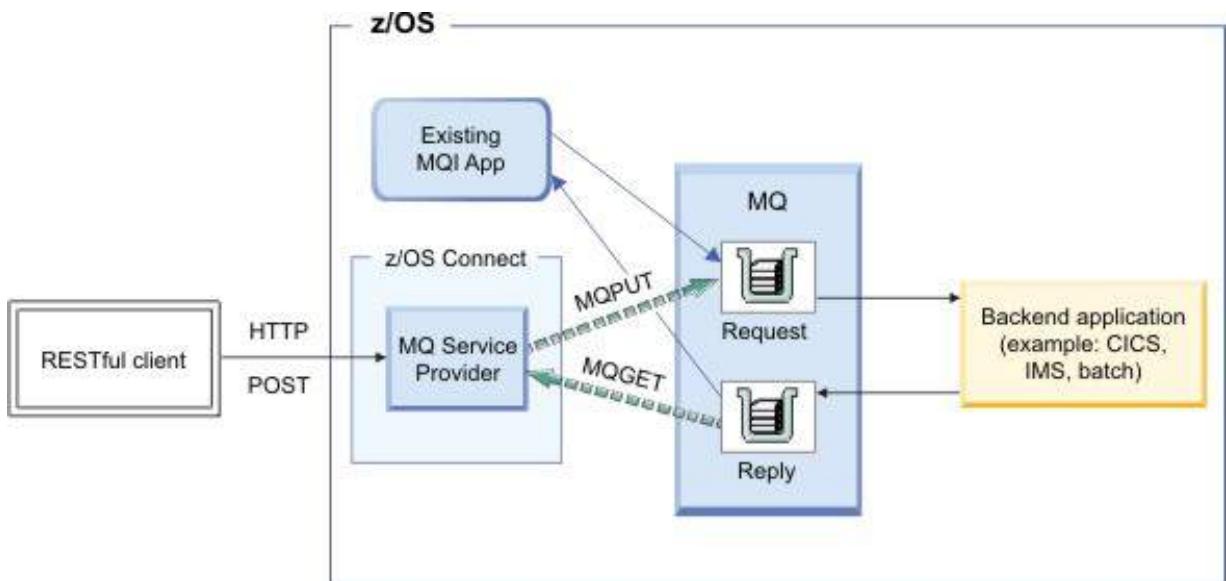


Figure 63. Two-way service

The client sends an HTTP request, specifying a JSON payload. The service takes the payload, optionally converts it to a different format such as a COBOL copybook, and sends it as a message to a request queue.

A backend application gets the message, processes it and generates a response, which is placed on a reply queue. The service locates this message, takes its payload, optionally converts it to JSON and returns it as the response body of the HTTP request.

Note: This description of services that are configured by using service archive files is the best method. Services that are configured in the `zosConnectService` and `mqService` elements of the `server.xml` file work slightly differently.

Use services that are defined by the `zosConnectService` and `mqService` elements only when you migrate from the service provider that was supplied with IBM MQ. For more information, see [“Migrating a service to the IBM MQ service provider in z/OS Connect EE” on page 280](#).

The `mqService` element doesn't support the `messagingAction` element. Instead, callers of the service specify what action to take on the underlying IBM MQ resources by using the HTTP verb. For example,

- For one-way services, an HTTP POST sends a message to an IBM MQ queue or topic, the equivalent of a `messagingAction` of `mqput`. An HTTP GET browses the first message from an IBM MQ queue. There is no `messagingAction` equivalent. An HTTP DELETE destructively gets a message from an IBM MQ queue, the equivalent of a `messagingAction` of `mqget`.
- For two-way services, only the HTTP POST verb can be used.

HTTP considerations for the IBM MQ service provider

When you use the IBM MQ service provider, be aware of the following HTTP-related information.

HTTP verbs

The IBM MQ service provider that is included as part of IBM MQ has a hardcoded mapping of HTTP verbs to messaging operations. These hardcoded mappings made it difficult to build truly RESTful APIs.

The IBM MQ service provider that is included with z/OS Connect EE does not have these hardcoded mappings. Any HTTP verb can be used to invoke either a one-way or a two-way service.

For a one-way service, the actual operation that is performed on the underlying IBM MQ queue or topic depends on the service project type if the service was created in the API toolkit, or the `messagingAction` property if the service was created from a properties file. For more information, see “[Creating an IBM MQ service by using a properties file](#)” on page 813.

A service project type of `MQ One-Way Service for Sending Messages`, or a `messagingAction` of `mqput` allows a message to be sent to either a queue or a topic.

A service project type of `MQ One-Way Service for Receiving Messages`, or a `messagingAction` of `mqget` allows a message to be received from a queue.

HTTP payload considerations

For one-way services configured to send messages:

- The HTTP request payload is optionally transformed by using the configured data transformation, and then sent as the payload of the IBM MQ message.
- The minimum valid payload is `{}`, that is, an empty JSON object.
- If the message is successfully sent, an empty HTTP response is returned to the caller with an HTTP status code of 204 (No Content).
- If a failure occurs, a response is returned as described in the [HTTP status codes](#) section.

For one-way services configured to receive messages:

- Any HTTP request payload is ignored.
- If a message is available on the IBM MQ queue, the payload of the message is optionally transformed by using the configured data transformation and then returned as the HTTP response body. The HTTP status code is 200 (OK).
- If the message has no payload, the HTTP response body is empty and the HTTP status code is 200 (OK).
- If no message is available on the IBM MQ queue, the HTTP response body is empty and the HTTP status code is 204 (No Content).

For two-way services:

- The HTTP request payload is optionally transformed by using the configured data transformation, and then sent as the payload of the IBM MQ message to the request queue or topic.
- The minimum valid payload is `{}`, that is, an empty JSON object.
- If a message is available on the IBM MQ reply queue, the payload of the message is optionally transformed by using the configured data transformation and then returned as the HTTP response body, the HTTP status code is 200 (OK).
- If the message on the reply queue has no payload, the HTTP response body is empty, and the HTTP status code is 200 (OK).
- If no message is available on the IBM MQ reply queue, the HTTP response body is empty, and the HTTP status code is 204 (No Content).

HTTP headers

When a non-blank HTTP request body is sent to either a one-way or a two-way service, it must be a valid JSON object encoded in UTF-8. The Content-Type=application/json HTTP header must also be specified.

Several optional HTTP headers can also be specified to adjust the behavior of the IBM MQ service provider. For more information about the HTTP headers that can be used with the IBM MQ Service Provider, see [“HTTP headers that can be used with the IBM MQ service provider” on page 272](#).

HTTP status codes

If the IBM MQ service provider detects an error, the service sets an HTTP status code in the range of 400-599. Otherwise, the status codes that were documented previously are always returned.

Client-side error

If non-valid data is passed in by the calling application, an HTTP status code in the range 400-499 is returned to the caller. The exact status code depends on the error.

In some cases, an exception stack trace in JSON format is returned to the caller as shown in the server-side example.

Server-side error

If the IBM MQ service provider experiences an unexpected error, then the stack trace is turned into JSON format, and returned to the caller with an HTTP status code of 500 (Internal Server Error). Suitable diagnostics are also be written to the z/OS Connect EE logs.

A sample error response payload is shown in [Figure 64 on page 271](#):

```
{  
  "errorText": "BAQM0006E: An unexpected JMSException occurred while processing a request for  
  service 'mq7'. "  
  "exceptionData": {  
    "exceptions": [  
      {  
        "stackTrace": [  
          "com.ibm.msg.client.jms.DetailedInvalidDestinationException: JMSWMQ2008:  
          Failed to open MQ queue 'ThisQueueDoesNotExist'.",  
          <further content removed for brevity>  
        ]  
      },  
      {  
        "stackTrace": [  
          "com.ibm.mq.MQException: JMSCMQ0001: WebSphere MQ call failed with  
          compcode '2' ('MQCC_FAILED') reason '2085' ('MQRC_UNKNOWN_OBJECT_NAME').",  
          <further content removed for brevity>  
        ]  
      }  
    ]  
  }  
}
```

Figure 64. A simplified example of an error response payload

Note: This description of services that are configured by using service archive files is the best method. Services that are configured in the `zosConnectService` and `mqService` elements of the `server.xml` file work slightly differently.

Use services that are defined by the `zosConnectService` and `mqService` elements only when you migrate from the service provider that was supplied with IBM MQ. For more information, see [“Migrating a service to the IBM MQ service provider in z/OS Connect EE” on page 280](#).

The `mqService` element doesn't support the `messagingAction` element. Instead, callers of the service specify what action to take on the underlying IBM MQ resources by using the HTTP verb. For example,

- For one-way services, an HTTP POST sends a message to an IBM MQ queue or topic, the equivalent of a messagingAction of mqput. An HTTP GET browses the first message from an IBM MQ queue. There is no messagingAction equivalent. An HTTP DELETE destructively gets a message from an IBM MQ queue, the equivalent of a messagingAction of mqget.
- For two-way services, only the HTTP POST verb can be used.

Transactional considerations

HTTP is not a transactional protocol so it cannot support transactional coordination of messaging operations that are performed by the IBM MQ service provider.

This limitation has the following implications:

- If you use a one-way service for sending messages, and the connection fails before an HTTP response is received by the client, the client cannot immediately tell whether the message was successfully sent to the configured queue or topic.
- If you use a one-way service for receiving messages, and the connection fails before an HTTP response is received by the client, then a message might be destructively got from the queue and lost, as it is not possible to roll back the destructive get.
- If you use a two-way service and the connection fails before an HTTP response is received by the client, the client cannot tell where the failure occurred. The request message might have been sent to the request queue, or the reply message might have been got from the reply queue and lost.
- It is not possible to coordinate the outcome of multiple HTTP verbs, to either a one-way or a two-way service.

HTTP headers that can be used with the IBM MQ service provider

The IBM MQ service provider expects specific HTTP headers when it calls either a two-way service, or a one-way service for sending messages.

In these cases, the Content-Type header must be set to application/json. If you specify a character set as part of this header, its value must be utf-8. For example,

```
Content-Type=application/json; charset=utf-8
```

Other HTTP headers can be specified on the HTTP request to change the behavior of the IBM MQ service provider. These headers are listed below. Any other HTTP headers are ignored.

ibm-mq-md-msgID

This header can be specified when requests are sent to a one-way service that is configured to receive messages.

The value of this header is used to generate a message selector to select a message with the specified message ID. If an ibm-mq-md-correlID header is also specified, a message selector that matches both IDs is generated.

ibm-mq-md-correlID

This header can be specified when requests are sent to either a two-way service, or a one-way service that is configured to send messages, in which case it is used to set the MQMD CorrelID field of the message that is sent.

This header can also be specified when requests are sent to a one-way service that is configured to receive messages. The value of this header is used to generate a message selector to select a message with the specified correlation ID. If an ibm-mq-md-msgID header is also specified, a message selector that matches both is generated.

ibm-mq-md-persistence

This header can be used to override the persistence property. It applies to either one-way services that are configured for sending messages or two-way services.

ibm-mq-md-expiry

This header can be used to override the expiry property. It applies to either one-way services that are configured for sending messages, or two-way services.

ibm-mq-gmo-waitInterval

This header can be used to override the waitInterval property if set on the service. It is only relevant for one-way services that are configured for receiving messages or two-way services. For more information about the waitInterval property, see the “[zosconnect_mqService](#)” on page 769 element in the *Reference* section.

ibm-mq-pmo-retain

You can specify this header with a value of TRUE when a request that specifies a topic is sent to a one-way service that is configured for sending messages. For more information, see [Retained publications](#) in the *IBM MQ* documentation.

ibm-mq-usr

Use this header to provide message properties on the IBM MQ messages that are sent as a result of requests to either a two-way service, or a one-way service that is configured for sending messages.

Security requirements for the IBM MQ service provider

Security for the IBM MQ service provider is in two parts.

- A user must be authorized to submit RESTful requests to IBM MQ service provider services that are exposed by z/OS Connect EE. Authorization is controlled by `server.xml` configuration file of z/OS Connect EE. For more information, see “[How to configure the authorization levels](#)” on page 381.
- An IBM MQ service provider service must be authorized to access the queue manager and its resources.

The IBM MQ service provider is fundamentally a JMS (Java Message Service) application, based on the IBM MQ messaging provider that is supplied with z/OS Connect EE. The IBM MQ service provider connects to one or more queue managers by using bindings or client mode connections.

As a result, the IBM MQ service provider can be secured in the same way as applications that share these traits.

The IBM MQ service provider connects to a queue manager and presents a user ID and optional password. These credentials are optionally validated by using connection authentication. For more information, see [Connection authentication](#) in the *IBM MQ* documentation.

Depending on the security configuration of the queue manager, and the validity of the user ID and password, the authenticated user can access the MQ queues or topics. For more information, see [Setting up security on z/OS](#) in the *IBM MQ* documentation.

Configuration attributes specified on the service and the `properties.wmqJMS` subelement of the `jmsConnectionFactory` element in `server.xml`, affect which user ID, and optional password, are presented to the queue manager. For more information, see [IBM MQ override properties](#).



Attention: There is a precedence order for the security configuration properties. The `userName` and `password` attributes override `properties.wmqJms` attributes, and the `useCallerPrincipal` attribute overrides all other attributes.

The possible combinations of attribute values and the resulting behavior are show in [Table 29 on page 274](#).

Table 29. Security configuration attributes

Service property		properties.wmqJms		Resulting behavior
useCallerPrincipal	userName and password	userName	userPassword	
Not set or false	Not set or blank	Not set or blank	Not set or blank	The user name that is associated with the z/OS Connect EE address space is presented to the queue manager for authorization and authentication purposes. No password is presented.
Not set or false	Not set or blank	Set	Not set or blank	The user name in the properties.wmqJms element is presented to the queue manager for authorization and authentication purposes. No password is presented.
Not set or false	Not set or blank	Set	Set	The user name and password in the properties.wmqJms element is presented to the queue manager for authorization and authentication purposes.
Not set or false	Both values set	Ignored if set	Ignored if set	The user name and password in the service property subelement or zosconnect_mqService element are presented to the queue manager for authorization and authentication purposes.
True	Ignored if set	Ignored if set	Ignored if set	The user principal that is authenticated to z/OS Connect EE is taken, and its user name is presented to the queue manager for authorization and authentication purposes. No password is presented.

Both password attributes can be provided in plain text or encoded format. Use the encoded format because anyone with access to the `server.xml` can view the password in plain text. z/OS Connect EE includes a tool called `securityUtility`, which can be used to encode passwords. For more information, see [securityUtility command in the WebSphere Application Server for z/OS Liberty documentation](#).

For more information about how to configure security for z/OS Connect EE, see [Chapter 9, “Securing z/OS Connect EE resources,” on page 331](#).

Data transformations with the IBM MQ service provider

You can use z/OS Connect EE to transform JSON data to an arbitrary format before a z/OS asset, such as a CICS transaction, is called. The response from the z/OS asset can also be transformed back into JSON.

Data transformation for COBOL and PL/I structures to and from JSON is performed by z/OS Connect EE.

Configuring data transformations

Create IBM MQ service provider services with one of the following methods:

1. Use the API toolkit. This is the preferred method. When using the API toolkit data transformation must be used. When a service is created with the API toolkit, it can later be built by the build toolkit. For more information, see [“Build a service archive with the build toolkit” on page 167](#).
2. Manually create a properties file that describes the service. The properties file is then used by the build toolkit to create the service. You can optionally configure data transformations on the service by

setting the `language` and `operationName` properties with at least one of the `requestStructure` or `responseStructure` properties. You can optionally specify the `ccsid` property. For more information, see [“Creating an IBM MQ service by using a properties file” on page 813](#).

3. Configure the `zosConnectService` and `mqService` elements in `server.xml`. Data transformation is optionally provided by using the artifacts that are generated by the BAQLS2JS utility and configured on the service by the `zosConnectDataXform` element in `server.xml`. This method is supported only for compatibility with the IBM MQ service provider that was supplied with IBM MQ. If you use services that were created with this method, consider migrating them to use the API toolkit method.

For information about using data transformations with services that are generated by method “1” on page 274 or “2” on page 274, see [“Considerations for services using the API or build toolkits” on page 275](#).

For information about using data transformations with services that are generated by method “3” on page 275 see [“Considerations for services using the mqService element” on page 276](#).

Considerations for services using the API or build toolkits

This topic describes data transformation considerations for IBM MQ service provider services created with either the API toolkit or the build toolkit.

Data structure size

If you used the API Toolkit to generate the service, or the build toolkit to generate a service project originally from the API toolkit, the maximum COBOL or PL/I data structure size is 100 MB.

If you used the build toolkit and a manually created properties file to generate the service, the maximum COBOL or PL/I data structure size is 32 KB.

CCSID

To use data transformations, a coded character set ID (CCSID) must be configured on the service.

For services generated by the API Toolkit the CCSID is set in the configuration page of the service. For services generated by the build toolkit, the CCSID value is set by using the `ccsid` property. For more information, see [“Creating an IBM MQ service by using a properties file” on page 813](#).

In most cases the value of CCSID should be 37. The following sections describe when the CCSID value is used.

Sending messages to IBM MQ

When an IBM MQ service provider service is configured with a data transformation, and sends a message to a queue manager (such as a one-way service for sending messages, or sending a request from a two-way service), the IBM MQ service provider performs the following steps:

1. Takes the JSON payload from the HTTP request.
2. Passes the payload to the z/OS Connect EE server for data transformation from JSON to a byte array.
Text from the JSON payload is converted into text in the target data structure in the CCSID specified on the service.
3. Takes the result of the data transformation and sends it to the queue manager in the form of a JMS BytesMessage. The `CodedCharSetId` field in the message’s MQMD is set to the CCSID that is configured on the service.

By default, the message that is sent has the MQMD Format field set to MQSTR, indicating that the message is a string message. Optionally, you can set the MQMD Format field to an alternative value using the API Toolkit configuration page. For more information, see [“Configuring service properties” on page 510](#). For services that are generated by the build toolkit you can set the value in the `mqmdFormat` property. For more information, see [“Creating an IBM MQ service by using a properties file” on page 813](#).

When the back-end IBM MQ application receives the message from the IBM MQ destination, it might use the contents of both the MQMD Format and CodedCharSetId field for data transformation or other purposes. Therefore, you must ensure that the `ccsid` and `mqmdFormat` properties are set for the target back-end application.

Receiving messages from IBM MQ

When an IBM MQ service provider service is configured with a data transformation, and is going to receive a message from a queue manager (such as a one-way service for getting messages, or receiving a reply for a two-way service), the IBM MQ service provider performs the following steps:

1. Gets the message from the queue.
2. Checks that the message is either a JMS BytesMessage or a JMS TextMessage. If the message is neither, an error is generated and returned to the caller.
 - If a TextMessage is received, then any text in the contained data structure is automatically converted into the CCSID that is configured on the service.
 - If a BytesMessage is received it is assumed that any text in the data structure is in the same CCSID as is configured on the service.

If it is not the same CCSID, data transformation will not generate the correct data. In this case, it is possible to configure IBM MQ to do data transformation by setting the JMS destination in the `server.xml` so that the `receiveConversion` attribute is set to QMGR and the `receiveCCSID` attribute is set to the same value as the CCSID configured on the service. An example of this is shown in [Figure 65 on page 276](#).

```
<jmsQueue jndiName="jms/sampleQ2Receive">
  <properties.wmqJms
    baseQueueName="SampleQ2Receive"
    receiveCCSID="37"
    receiveConversion="QMGR"/>
</jmsQueue>
```

Figure 65. Sample configuration using IBM MQ data transformation

3. Passes the message payload to the z/OS Connect EE server for data transformation from a byte array to JSON.
4. Takes the result of the data transformation and returns it as the response of the HTTP request.

Considerations for services using the `mqService` element

This topic describes data transformation considerations for the IBM MQ service provider services that are created by using the `zosConnectService` and `mqService` elements in the `server.xml` file.

Data structure size

The maximum data structure size depends on the value of the **PGMINT** parameter that is used with BAQLS2JS. If **PGMINT=COMMAREA**, then the data structure must be less than or equal to 32 KB. If **PGMINT=CHANNEL**, then the data structure must be less than or equal to 100 MB, which is the maximum message size that is supported by IBM MQ.

Sending messages to IBM MQ

When an IBM MQ service is configured with a data transformation, and is going to send a message to a queue manager (for example if it is a one-way or two-way service that receives an HTTP POST containing JSON) the IBM MQ service provider performs the following steps:

1. Takes the JSON payload from the HTTP request.
2. Passes the payload to z/OS Connect EE to perform data transformation from JSON to a byte array.
3. Takes the result of the data transformation and sends it to the queue manager in the form of a JMS BytesMessage.

By default, the message that is sent has the MQMD **Format** field set to MQSTR, indicating that the message is a string message. If not set to MQSTR, you can set the `mqmdFormat` attribute in the `mqzOSConnectService` element to an appropriate value.

The built-in data transformation support always generates output with a CCSID of 37. This information needs to be specified in the MQMD `CodedCharSetId` field, otherwise the application that gets messages from the queue might not be able to decode the message. Set the value in the CCSID attribute on the queue referenced by the `mqzOSConnectService` service element.

The following example illustrates appropriate configuration for a one-way service that is going to be used to send messages to a queue called SampleQ1.

The messages are to be sent with an MQMD Format field of `AFORMAT` and a `CodedCharSetId` field of 37.

z/OS Connect EE uses the `zosConnectDataXform` element to locate the configuration for data transformations. This element is referenced by the `dataXformRef` attribute of the `zosConnectService` element.

For configuration for a two-way service, see [“Receiving messages from IBM MQ” on page 277](#).

```
<jmsConnectionFactory
  id="sampleCF1"
  jndiName="jms/sampleCF1"
  connectionManagerRef="sampleCF1ConnectionManager">

  <properties.wmqJms
    transportType="BINDINGS"
    queueManager="MQ21"/>
</jmsConnectionFactory>

<connectionManager
  id="sampleCF1ConnectionManager"
  maxPoolSize="5"/>

<jmsQueue id="sampleQ1"
  jndiName="jms/sampleQ1">

  <properties.wmqJms
    baseQueueName="SampleQ1"
    CCSID="37"/>
</jmsQueue>

<zosConnectService
  id="samplezOSConnectService1"
  invokeURI="/samplezOSConnectService1"
  serviceName="samplezOSConnectService1_name"
  serviceRef="samplezOSConnectService1_MQ"
  dataXformRef="xformJSON2Byte"/>

<mqzOSConnectService
  id="samplezOSConnectService1_MQ"
  connectionFactory="jms/sampleCF1"
  mqmdFormat="AFORMAT"
  destination="jms/sampleQ1" />

<zosConnectDataXform id="xformJSON2Byte"
  bindFileLoc="/XFORM_ROOT/bindfiles" bindFileSuffix=".bnd"
  requestSchemaLoc="/XFORM_ROOT/json" requestSchemaSuffix=".json"
  responseSchemaLoc="/XFORM_ROOT/json"
  responseSchemaSuffix=".json" />
```

Receiving messages from IBM MQ

When an IBM MQ service provider instance that is configured for data transformation, receives a message from a queue manager (such as a one-way service sending an HTTP GET or DELETE request, or a two-way service that received an HTTP POST) the IBM MQ service provider performs the following steps.

1. Gets the message from the queue.

2. Checks that the message is either a JMS **BytesMessage** or a JMS **TextMessage**. If the message is neither, an error is generated and returned to the caller.
3. Passes the message payload to z/OS Connect EE to perform data transformation from a byte array to JSON.
4. Takes the result of the data transformation and returns it as the response of the HTTP method.

Depending on the type of message received, you might need to configure other attributes. This is because the IBM MQ service provider needs to convert the received message payload into the correct format to pass to the configured data transformation.

The default data transformation expects payload to be in CCSID 37. However, the z/OS asset might not generate messages in this CCSID.

The configuration depends on whether a **BytesMessage** or **TextMessage** is received.

BytesMessage received on z/OS Connect EE

If a **BytesMessage** is to be received, then you can specify the **receiveConversion="QMGR"** and **receiveCCSID="37"** attributes on the queue definition that is used to receive the message.

This is illustrated in the following example. In this case, the sampleQ2Receive definition has both the **receiveConversion** and **receiveCCSID** attributes set.

```
<jmsConnectionFactory
  id="sampleCF2"
  jndiName="jms/sampleCF2"
  connectionManagerRef="sampleCF2ConnectionManager">

  <properties.wmqJms
    transportType="BINDINGS"
    queueManager="MQ21"/>
</jmsConnectionFactory>

<connectionManager
  id="sampleCF2ConnectionManager"
  maxPoolSize="5"/>

<jmsQueue id="sampleQ2Send"
  jndiName="jms/sampleQ2Send">

  <properties.wmqJms
    baseQueueName="SampleQ2Send"
    CCSID="37"/>
</jmsQueue>

<jmsQueue id="sampleQ2Receive"
  jndiName="jms/sampleQ2Receive">

  <properties.wmqJms
    baseQueueName="SampleQ2Receive"
    receiveCCSID="37"
    receiveConversion="QMGR"/>
</jmsQueue>

<zosconnect_zosConnectService
  id="samplezOSConnectService2"
  invokeURI="/samplezOSConnectService2"
  serviceName="samplezOSConnectService2_name"
  serviceRef="samplezOSConnectService2_MQ"
  dataXformRef="xformJSON2Byte"/>

<mqzosconnect_mqzOSConnectService
  id="samplezOSConnectService2_MQ"
  connectionFactory="jms/sampleCF2"
  mqmdFormat="AFORMAT"
  destination="jms/sampleQ2Send"
  replyDestination="jms/sampleQ3Receive" />

<zosconnect_zosConnectDataXform id="xformJSON2Byte"
  bindFileLoc="/XF0RM_ROOT/bindfiles" bindFileSuffix=".bnd"
  requestSchemaLoc="/XF0RM_ROOT/json" requestSchemaSuffix=".json"
```

```
responseSchemaLoc="/XFORM_ROOT/json"
responseSchemaSuffix=".json" />
```

TextMessage received on z/OS Connect EE

If a **TextMessage** is to be received then you need to convert the message to the CCSID expected by the data transformation, by default 37.

If you use a custom data transformation, and the transformation expects a CCSID other than 37, you can specify the `receiveTextCCSID` attribute in the `mqzOSConnectService` element to set an appropriate CCSID.

This is illustrated in the following example, where the `zOSConnectService3_MQ` definition has the `receiveTextCCSID` attribute in the `mqzOSConnectService` element set to 1208 (UTF-8).

```
<jmsConnectionFactory
  id="sampleCF3"
  jndiName="jms/sampleCF3"
  connectionManagerRef="sampleCF3ConnectionManager">

  <properties.wmqJms
    transportType="BINDINGS"
    queueManager="MQ21"/>
</jmsConnectionFactory>

<connectionManager
  id="sampleCF3ConnectionManager"
  maxPoolSize="5"/>

<jmsQueue id="sampleQ3Send"
  jndiName="jms/sampleQ3Send">

  <properties.wmqJms
    baseQueueName="SampleQ3Send"
    CCSID="37"/>
</jmsQueue>

<jmsQueue id="sampleQ3Receive"
  jndiName="jms/sampleQ3Receive">

  <properties.wmqJms
    baseQueueName="SampleQ3Receive"/>
</jmsQueue>

<zosconnect_zosConnectService
  id="samplezOSConnectService3"
  invokeURI="/samplezOSConnectService3"
  serviceName="samplezOSConnectService3_name"
  serviceRef="samplezOSConnectService3_MQ"
  dataXformRef="customDataXForm"/>

<mqzosconnect_mqzOSConnectService
  id="samplezOSConnectService3_MQ"
  connectionFactory="jms/sampleCF3"
  mqmdFormat="AFORMAT"
  destination="jms/sampleQ3Send"
  replyDestination="jms/sampleQ3Receive"
  receiveTextCCSID="1208" />
```

Handling MQRFH2 headers with the IBM MQ service provider

The IBM MQ service provider provides a RESTful interface to existing applications that use IBM MQ. These applications can interact with IBM MQ, by using either the IBM MQ classes for JMS or the Message Queue Interface (MQI)

By default the IBM MQ classes for JMS send messages that include an MQRFH2 header. However, most MQI applications do not use MQRFH2 headers.

If an application that uses IBM MQ classes for JMS interacts with an MQI application that is not designed to work with MQRFH2 headers, configuration is required to prevent the application sending messages that contain an MQRFH2 header. For more information, see [Mapping JMS messages onto IBM MQ messages](#).

If the IBM MQ service provider sends messages to queues, to be consumed by MQI applications that do not expect an MQRFH2 header, you must configure z/OS Connect EE to prevent the sending of an MQRFH2 header.

This configuration consists of adding a `targetClient="MQ"` attribute to the relevant IBM MQ Messaging Provider queue in `server.xml`. An example of this configuration is shown in [Figure 66 on page 280](#).

```
<jmsQueue jndiName="jms/mqiQueue">
  <properties.wmqJms>
    baseQueueName = "MQIQueue"
    targetClient = "MQ"/>
</jmsQueue>
```

Figure 66. Sample configuration to prevent the sending of MQRFH2 headers.

Migrating a service to the IBM MQ service provider in z/OS Connect EE

How to migrate a service that uses the IBM MQ service provider that is supplied with IBM MQ to use the IBM MQ service provider that is built in to z/OS Connect EE.

About this task

The IBM MQ service provider that is supplied with IBM MQ allows services to be defined only by using the `zosConnectService` element in the `server.xml` configuration file.

The IBM MQ service provider that is built in to z/OS Connect EE allows services to be defined by using either a service archive file or the `zosConnectService` element in the `server.xml` configuration file.

All new services should be defined by using the service archive file approach.

For migration purposes, you can continue to use the `zosConnectService` based approach. The service is the same, except for the messages that are generated by the service if it is configured or invoked incorrectly.

This task assumes that the `server.xml` configuration file contains the following elements and that the MQ service provider product feature was installed. Other elements such as security and JMS definitions are not shown:

```
<featureManager>
  <feature>zosconnect:zosconnect-2.0</feature>
  <feature>appSecurity-2.0</feature>
  <feature>jms-2.0</feature>
  <feature>mqzosconnect:zosConnectMQ-2.0</feature>
  <feature>wmqJmsClient-2.0</feature>
  <feature>zostransaction-1.0</feature>
</featureManager>

<zosconnect_zosConnectService invokeURI="/mq1"
  serviceName="mq1_name"
  serviceRef="mq1" />
<mqzosconnect_mqzOSConnectService id="mq1"
  connectionFactory="jms/cf1"
  destination="jms/d1" />
```

Procedure

1. Enable the built-in IBM MQ service provider by adding the following line inside the `featureManager` element:

```
<feature>zosconnect:mqService-1.0</feature>
```

2. Switch the service to the built-in service provider by changing `mqzosconnect_mqzOSConnectService` to `zosconnect_mqService`.
3. Repeat step [“2” on page 280](#) for any other instances of `mqzosconnect_mqzOSConnectService`.
4. Remove the old service provider by deleting the following lines from the `featureManager` element:

```
<feature>jms-2.0</feature>
<feature>mqzosconnect:zosConnectMQ-2.0</feature>
<feature>wmqJmsClient-2.0</feature>
<feature>zosTransaction-1.0</feature>
```

Results

The following sample shows the resulting `server.xml` file.

```
<featureManager>
  <feature>zosconnect:zosconnect-2.0</feature>
  <feature>appSecurity-2.0</feature>
  <feature>zosconnect:mqService-1.0</feature>
</featureManager>

<zosconnect_zosConnectService invokeURI="/mq1"
  serviceName="mq1_name"
  serviceRef="mq1" />
<zosconnect_mqService id="mq1"
  connectionFactory="jms/cf1"
  destination="jms/d1" />
```

Test the service to ensure it is still usable. If it fails, check the `messages.log` file for information to help diagnose the problem.

What to do next

When the IBM MQ service provider product feature is no longer needed, consider uninstalling it by removing the `mqzosconnect.properties` file from the z/OS Connect EE product extensions directory. For example. `/var/zosconnect/v3r0/extensions`.

Overriding IBM MQ service properties

The JNDI name of the IBM MQ resources like connection factory, queues or topics must be specified during the IBM MQ service project creation. The values of the JNDI names for these resources can be overridden in server configuration or at run time.

The override precedence is in the following order from lowest to highest priority:

1. The JNDI name of the IBM MQ resources must be specified in the service project editor. For more information, see [Properties for IBM MQ](#) in the *Configuring service properties* section.
2. The JNDI names of the IBM MQ resources can be overridden by the `zosconnect_services` setting in the `server.xml` file. For example, to override the JNDI name of the connection factory, specify the property name as `connectionFactory` and the value as the new connection factory. The following `server.xml` definition overrides connection factory for the IBM MQ service named `MqExample`:

```
<zosconnect_services>
  <service name="MqExample">
    <property name="connectionFactory" value="newValue"/>
  </service>
</zosconnect_services>
```

For more information, see [“zosconnect_services”](#) on page 774 in the *Reference* section.

3. If an active policy contains a rule that modifies the JNDI names of the IBM MQ resources, the modified value overrides all other specifications. For the properties that you can use to modify the JNDI name of an IBM MQ resource in a policy rule, see [“Valid properties for use in policy rules”](#) on page 687.

Using Db2 services

Use the REST client service provider that is supplied with z/OS Connect EE to connect to a Db2 endpoint. Services are defined using service archive (.sar) files.

Configuring a REST client connection to Db2

Follow these steps to configure a REST client connection to a Db2 endpoint.

Before you begin

Before you begin this task, ensure that a server instance is set up, see [“Creating a z/OS Connect Enterprise Edition server” on page 217](#).

About this task

Edit the `server.xml` file to configure a REST client connection element to communicate with a Db2 endpoint. You can also define security elements.

Note: If z/OS Connect EE connects to a System of Record (SoR) that is HTTP 1.1 compliant, then connections are persistent by default, but controlled by the server. The underlying HTTP connection is cached in the JVM and can be reused by another client.

Procedure

1. Configure a `zosconnect_zosConnectServiceRestClientConnection` element in `server.xml`.
The `host` and `port` attributes must be specified. For example:

```
<zosconnect_zosConnectServiceRestClientConnection id="db2Conn"  
host="db2.example.com" port="8080"/>
```

For more information about the available configuration attributes and default values, see [“zosconnect_zosConnectServiceRestClientConnection” on page 790](#).

2. If the Db2 endpoint requires security, see [“Configuring security for a REST client connection to Db2” on page 282](#).

Configuring security for a REST client connection to Db2

REST connections to Db2 can be secured by using an HTTPS connection, with the addition of a user credential that is authenticated with a password.

HTTPS connections

HTTPS can be used to secure a connection to ensure that the data transferred between the two parties is encrypted and that each party is able to validate the identity of the other party.

The client side of the TLS connection can be configured either in the z/OS Connect EE `server.xml` configuration file or by using Application Transparent Transport Layer Security (AT-TLS). For more information about configuring AT-TLS, see [Using Application Transparent Transport Layer Security \(AT-TLS\)](#) in the *IBM z/OS Communication Server* documentation.

For information about configuring the server side of the TLS connection, see the Db2 product documentation.

User authentication

A user ID and password or PassTicket can be used to authenticate that the user is able to call the Db2 endpoint.

User authentication can be used with HTTP or HTTPS connections. Db2 RESTful services support authentication by using [basic authentication](#), [PassTicket authentication](#), or for TLS client authentication, the client certificate can be mapped to a SAF user ID.

Note: Db2 for z/OS RESTful services do not support the use of RACF-protected user IDs for PassTicket authentication. This is a permanent restriction that is documented by APAR PH12603 titled *ENABLE REST SERVICE REQUESTS TO UTILIZE RACF PASSTICKETS WITH PROTECTED USERIDS FOR AUTHENTICATION*.

However, if you use TLS client authentication to authenticate, then Db2 for z/OS does support the client certificate that is mapped to a RACF-protected user ID.

Configuring HTTPS on a REST client connection to Db2

Follow these steps to configure HTTPS on a REST client connection to a Db2 endpoint.

About this task

Update `server.xml` to configure SSL on a REST client connection definition.

Procedure

1. Edit the z/OS Connect EE server configuration file to define an **SSL** element or **SSLDefault** element.
2. Add an **sslCertsRef** attribute to the `zosconnect_zosConnectServiceRestClientConnection` element to reference the SSL element.

For example:

```
<featureManager>
  <feature>zosconnect:zosconnect-2.0</feature>
</featureManager>

<!-- Define the SSL configuration. -->
<ssl id="defaultSSLConfig" keyStoreRef="defaultKeyStore"
trustStoreRef="defaultTrustStore" clientAuthentication="false" />

<!-- Define a keystore. Contains the z/OS Connect EE server's personal certificate to be
sent on SSL handshake. -->
<keyStore id="defaultKeyStore" password="zosconnect" location="${server.config.dir}/
resources/security/serverKey.jks" />

<!-- Define a truststore. Contains the CICS region's public certificate expected to be
sent on the SSL handshake. -->
<keyStore id="defaultTrustStore" password="zosconnect" location="${server.config.dir}/
resources/security/serverTrust.jks"/>

<!-- Define the connection to the Db2 endpoint -->
<zosconnect_zosConnectServiceRestClientConnection id="db2Conn"
sslCertsRef="defaultSSLConfig"
host="db2.example.com" port="8080" />
```

Configuring basic authentication for a REST client connection to Db2

Follow these steps to configure user security credentials for a REST client connection to Db2.

Before you begin

1. Configure a `zosconnect_zosConnectServiceRestClientConnection` element in `server.xml`. For more information, see “[Configuring a REST client connection to Db2](#)” on page 282.
2. Locate the WebSphere Liberty profile server **securityUtility** command tool, which can be found in the `<installation_path>/wlp/bin` directory.

Procedure

1. Use the **securityUtility encode** command to encode the password for the user ID that is to be defined for the connection. For more information, see the [securityUtility command](#).
2. Define a `zosconnect_zosConnectServiceRestBasicAuth` element in `server.xml`. The **userName** and **password** attributes must both be specified. For the password, specify the entire encoded string output by running the **securityUtility** command.
3. Add the **basicAuthRef** attribute to the `zosconnect_zosConnectServiceRestClientConnection` element to reference the `zosconnect_zosConnectServiceRestBasicAuth` element. For example:

```

<zosconnect_zosConnectServiceRestClientBasicAuth id="authABC"
    userName="SYSUSER" password="{xor}0jIvb18oMzs="/>
<zosconnect_zosConnectServiceRestClientConnection id="db2Conn"
    host="db2.example.com" port="1110" basicAuthRef="authABC"/>

```

Configuring PassTicket support for Db2 services

When invoking Db2 services, use PassTickets for identity propagation. A PassTicket is generated for the specific user ID and application, by z/OS Connect EE, and is then passed in the HTTP Authorization header to Db2 where it is validated.

Before you begin

- Configure a `zosconnect_zosConnectServiceRestClientConnection` element in `server.xml`. For more information, see “[Configuring a REST client connection to Db2](#)” on page 282.
- Define `<safCredentials mapDistributedIdentities="true" />` in the `server.xml` configuration file. For more information about configurations, see “[How to activate and configure the SAF user registry](#)” on page 456 or “[How to configure an LDAP user registry](#)” on page 455

Note: PassTicket generation uses native services, and as such can reduce zIIP-eligible processing.

About this task

PassTicket generation can be used with any form of z/OS Connect EE supported authentication mechanism except basic user registry. Figure 67 on page 284 illustrates how you can use PassTickets for identity propagation when a distributed ID is used to authenticate to z/OS Connect EE.

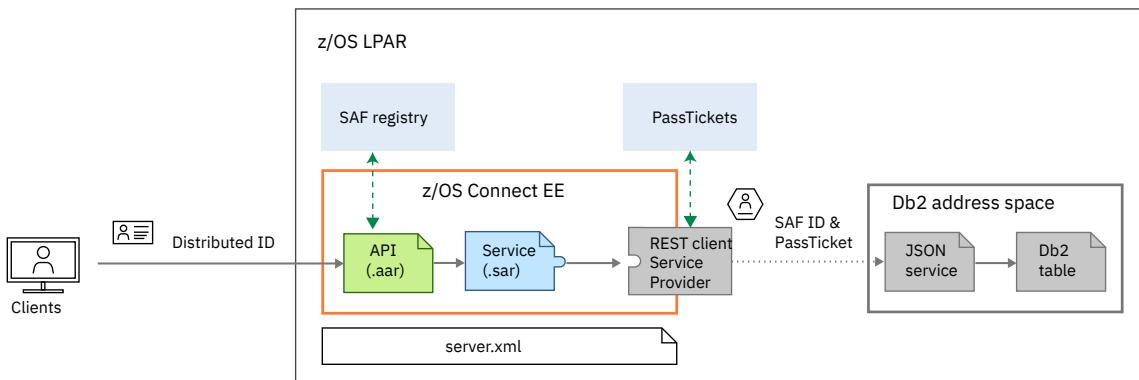


Figure 67. How PassTickets are used for identity propagation

A distributed ID is included in the request to z/OS Connect EE and then authenticated and mapped to a SAF user ID. The user ID in the request, which can also be a SAF user ID, is then propagated to Db2 with the generated PassTicket, where further authentication is carried out by Db2. If a specific user ID is required for authentication with Db2 for all requests, then that user ID can be configured in your `server.xml` configuration file.

Note:

- To map LDAP distributed IDs to SAF IDs, add `mapDistributedIdentities="true"` to the `server.xml` configuration file.
- When the z/OS Connect EE server and Db2 are located in different LPARs that do not use the same RACF database, the commands in steps 1, 2, and 4 must be issued to each LPAR. The commands in step 3 need to be entered only on the LPAR in which the z/OS Connect EE server runs.

Procedure

- From a TSO session, enter the following commands to activate the RACF PassTicket class:

```
SETROPTS CLASSACT(PTKTDATA) RACLIST(PTKTDATA)
SETROPTS GENERIC(PTKTDATA)
```

- Issue the following commands to define RACF profiles for the application in PTKTDATA:

```
RDEFINE PTKTDATA <appName> SSIGNON(KEYMASKED(<key>))
    APPLDATA('NO REPLAY PROTECTION')
```

where

- <appName> is the name of the application that requests and uses the PassTickets. If necessary, issue the Db2 **-DISPLAY DDF** command to find out the application name as the value for <appName>.
 - If GENERICCLU is defined, use the second part of GENERICCLU for <appName>.
 - If GENERICCLU is not defined, use the second part of LUNAME for <appName>.
 - If neither GENERICCLU or LUNAME is defined, use the value of the IPNAME for <appName>.
- <key> is a session key with the value of 16 hexadecimal digits (for an 8-bit or 64-bit key). The session key must be identical to the key in the PassTicket definition in each RACF instance. The key for each application must be the same on all systems in the configuration.
- APPLDATA (' NO REPLAY PROTECTION ') is the option that you can use to permit reuse of the same PassTicket multiple times.

- Issue the following commands to define RACF profiles for PassTicket generation:

```
RDEFINE PTKTDATA IRRPTAUTH.<appName>.* UACC(NONE)
PERMIT IRRPTAUTH.<appName>.* ID(<zosconnect_id>) ACCESS(UPDATE) CLASS(PTKTDATA)
```

where <zosconnect_id> is the user ID under which the z/OS Connect EE started task procedure runs.

- Issue the following command to refresh the PTKTDATA class:

```
SETROPTS RACLIST(PTKTDATA) REFRESH
```

- Define a zosconnect_zosConnectServiceRestClientBasicAuth element in `server.xml` and set the **appName** attribute. The value of the **appName** attribute must match the <appName> value in step 2.

If the **userName** attribute is specified then this user ID is used with the PassTicket, otherwise the authenticated user ID for the request is used. The authenticated user ID can be either a distributed ID mapped to SAF or the SAF ID that was sent on the API request.

- Add the **basicAuthRef** attribute to the `zosconnect_zosConnectServiceRestClientConnection` element to reference the `zosconnect_zosConnectServiceRestClientBasicAuth` element.
For example,

```
<zosconnect_zosConnectServiceRestClientBasicAuth
    id="authABC" appName="db2appl"/>
<zosconnect_zosConnectServiceRestClientConnection id="db2Conn"
    host="db2.example.com" port="8080" basicAuthRef="authABC"/>
```

Related tasks

[“How to map an LDAP user ID to a RACF user ID” on page 357](#)

Overriding Db2 service properties in the server configuration

Collection ID and connection reference must be specified during the Db2 service project creation. The values of collection ID and connection reference can be overridden in the service project editor or at run time.

The override precedence is in the following order from lowest to highest priority:

1. A collection ID and a connection reference must be specified in the service project editor.
2. The collection ID and connection reference can be overridden by the `zosconnect_services` setting using the service and property.

To override the collection ID, specify the property name as `collectionId` and the value as the new collection ID. To override the connection reference, specify the property name as `connectionRef` and the value as the new connection reference ID. For example, the following `server.xml` definition overrides both the collection ID and connection reference for the Db2 service named `exampleDb2`:

```
<zosconnect_services>
  <service name="exampleDb2" >
    <property name="collectionId" value="newValue" />
    <property name="connectionRef" value="newConnection" />
  </service>
</zosconnect_services>
```

For more information, see “[zosconnect_services](#)” on page 774 in the *Reference* section.

3. If an active policy contains a rule that modifies the collection ID or the connection reference, the modified value overrides all other specifications. For the properties that you can use to modify the collection ID or the connection reference in a policy rule, see “[Valid properties for use in policy rules](#)” on page 687.

Using the REST client service provider

This REST client service provider is supplied with z/OS Connect EE and connects to an HTTP endpoint.

Services are defined using service archive (.sar) files.

Configuring a REST client connection in z/OS Connect EE

Follow these steps to configure a REST client connection to an HTTP endpoint.

Before you begin

Before you begin this task, ensure that a server instance is set up, see “[Creating a z/OS Connect Enterprise Edition server](#)” on page 217.

About this task

Edit the `server.xml` file to configure a REST client connection element. You can also define security elements.

Note: If z/OS Connect EE connects to a System of Record (SoR) that is HTTP 1.1 compliant, then connections are persistent by default, but controlled by the server. The underlying HTTP connection is cached in the JVM and can be reused by another client.

Procedure

1. Configure a `zosconnect_zosConnectServiceRestClientConnection` element in `server.xml`. The `host` and `port` attributes must be specified. For example:

```
<zosconnect_zosConnectServiceRestClientConnection id="remoteHttp"
  host="example.com" port="8080"/>
```

For more information about the available configuration attributes and default values, see “[zosconnect_zosConnectServiceRestClientConnection](#)” on page 790.

2. If the HTTP endpoint requires security, see “[Configuring security for a REST client connection](#)” on page 287.

Configuring security for a REST client connection

REST Connections can be secured by using an HTTPS connection, with the addition of a user credential that is authenticated with a password.

HTTPS connections

HTTPS can be used to secure a connection, ensuring that the data transferred between the two parties is encrypted and that each end is able to validate the other end is who they say they are.

User authentication

A user ID and password or PassTicket can be used to authenticate that the user is able to call the HTTP endpoint.

Configuring HTTPS on a REST client connection

Follow these steps to configure HTTPS on a REST client connection to a HTTP endpoint.

About this task

Update `server.xml` to configure SSL on a REST client connection definition.

Procedure

1. Edit the z/OS Connect EE server configuration file to define an **SSL** element or **SSLDefault** element.
2. Add an **sslCertsRef** attribute to the `zosconnect_zosConnectServiceRestClientConnection` element to reference the SSL element.

For example:

```
<featureManager>
    <feature>zosconnect:zosconnect-2.0</feature>
</featureManager>

<!-- Define the SSL configuration. -->
<ssl id="defaultSSLConfig" keyStoreRef="defaultKeyStore"
trustStoreRef="defaultTrustStore" clientAuthentication="false" />

<!-- Define a keystore. Contains the z/OS Connect EE server's personal certificate to be
sent on SSL handshake. -->
<keyStore id="defaultKeyStore" password="zosconnect" location="${server.config.dir}/
resources/security/serverKey.jks" />

<!-- Define a truststore. Contains the CICS region's public certificate expected to be
sent on the SSL handshake. -->
<keyStore id="defaultTrustStore" password="zosconnect" location="${server.config.dir}/
resources/security/serverTrust.jks"/>

<!-- Define the connection to the REST API -->
<zosconnect_zosConnectServiceRestClientConnection id="restConn"
sslCertsRef="defaultSSLConfig"
host="example.com" port="8080" />
```

Configuring basic authentication for a REST client connection

Follow these steps to configure user security credentials for a REST client connection.

Before you begin

1. Configure a `zosconnect_zosConnectServiceRestClientConnection` element in `server.xml`.
For more information, see “[Configuring a REST client connection in z/OS Connect EE](#)” on page 286.
2. Locate the WebSphere Liberty profile server **securityUtility** command tool, which can be found in the `<installation_path>/wlp/bin` directory.

Procedure

1. Use the **securityUtility encode** command to encode the password for the user ID that is to be defined for the connection. For more information, see the [securityUtility command](#).
2. Define a `zosconnect_zosConnectServiceRestBasicAuth` element in `server.xml`. The **userName** and **password** attributes must both be specified. For the password, specify the entire encoded string output by running the **securityUtility** command.
3. Add the **basicAuthRef** attribute to the `zosconnect_zosConnectServiceRestClientConnection` element to reference the `zosconnect_zosConnectServiceRestClientBasicAuth` element. For example:

```
<zosconnect_zosConnectServiceRestClientBasicAuth id="authABC"
  userName="SYSUSER" password="{xor}0jIvb18oMzs="/>
<zosconnect_zosConnectServiceRestClientConnection id="restConn"
  host="example.com" port="1110" basicAuthRef="authABC"/>
```

Using other service providers

Using the service providers that are supplied with other products.

CICS, WOLA, IMS, and MQ service providers are included with z/OS Connect EE. You can also write your own service provider or you can use one that is supplied with the System of Record (SoR) to plug into the framework.

Note: The MQ Service provider is supplied with z/OS Connect EE v3.0.21 and later. For earlier versions, you can use the service provider that is supplied with IBM MQ. For more information, see [IBM MQ for z/OS Service Provider for z/OS Connect EE](#) in the *IBM MQ* documentation.

Configuring interceptors

z/OS Connect EE provides a framework that enables interceptors (exit programs) to be invoked for API, service, API requester, and administration requests. Interceptors are OSGi services that implement the `com.ibm.zosconnect.spi.Interceptor` Service Provider Interface (SPI) that is provided by z/OS Connect EE.

About this task

Interceptors can be configured either globally in a z/OS Connect EE server by setting the **globalInterceptorsRef** attribute of the `zosConnectManager` element, or specifically for APIs, services, or API requesters by setting the **interceptorsRef** attribute on the appropriate `zosConnectAPI`, `service`, `zosConnectService`, or `apiRequester` element. If interceptors are configured globally, by default they apply to all APIs, services, and API requesters, because the `zosConnectAPI`, `service`, `zosConnectService`, and `apiRequester` elements all have the default setting of `runGlobalInterceptors="true"`. To prevent global interceptors run for an individual API, service, or API requester, specify `runGlobalInterceptors="false"` on the relevant element.

Note:

1. Interceptors that are configured for services are only triggered if the service is invoked directly from an HTTP or HTTPS request, they are not triggered if the service is invoked from an API.
2. If an interceptor is configured at both the global level and specifically for an API, service, or API requester, then the interceptor is called twice. For example, if this configuration is used for the audit interceptor, then two SMF123 records are written for each request.

z/OS Connect EE provides three interceptors:

1. The file system logger interceptor enables z/OS Connect EE users to log information about API and service requests in a file. This interceptor is not called for API requester invocations.
2. The audit interceptor is used to audit and track requests. For more information, see [“Using SMF records to monitor requests” on page 475](#)
3. The authorization interceptor controls access to z/OS Connect EE artifacts by assigning users one of four authorization levels, Admin, Operations, Invoke and Reader. These authorization levels are set in the global, API, service, or API requester configuration elements. For more information, see [“Overview of z/OS Connect EE security” on page 331](#) and the subtopics relating to security authorization.

You can use interceptors for many purposes. z/OS Connect EE provides the same information to all interceptors regardless of their purpose. For example, an interceptor might be written to perform some infrastructure setup that is based on the message payload before the request is processed. z/OS Connect EE provides a copy of the input request payload to all interceptors.

This task describes how to configure custom z/OS Connect EE interceptors and how to enable those interceptors globally for the z/OS Connect EE server.

If defined, interceptors are called in a specific order:

1. Global interceptors `preInvoke` methods
2. API, service, or API requester interceptors `preInvoke` methods
3. The requested API, service, or API requester operation is performed.
4. API, service, or API requester interceptors `postInvoke` methods
5. Global interceptors `postInvoke` methods

Within each category, the **sequence** attribute determines the order in which the interceptors run. For `preInvoke` methods, the lowest number runs first. For `postInvoke` methods, the lowest number runs last. If the sequence numbers are the same, there is no set order. If the sequence is not defined or incorrectly defined, these interceptors run their `preInvoke` methods after all other interceptors of the same type, and their `postInvoke` methods before all other interceptors of the same type. For example, if you want the audit interceptor to capture failed authorization requests then you need to ensure that the sequence number of the audit interceptor is lower than that of the authorization interceptor.

For more information, see [Interface Interceptor](#).

Procedure

Create a global interceptor list and enable it in the `zosconnect_zosConnectManager` element.

The **globalInterceptorsRef** attribute is the ID of the configuration element that describes the list of z/OS Connect EE interceptors that apply globally within the server. The **interceptorsRef** attribute is a comma-separated list of the interceptors to be applied. In this example, two user-created interceptors are configured.

```
<zosconnect_zosConnectManager globalInterceptorsRef="globalInterceptors"/>

<!-- User Interceptor definitions -->
<usr_userInterceptorOne id="userI1" sequence="1"/>
<usr_userInterceptorTwo id="userI2" sequence="2"/>

<!-- Global interceptor definitions -->
<zosconnect_zosConnectInterceptors id="globalInterceptors" interceptorRef="userI1, userI2"/>
```

Configuring the file system logger interceptor

The file system logger interceptor enables z/OS Connect EE users to log information about API and service requests in a file. It is not called for API requester invocations.

About this task

This function is available when you configure z/OS Connect EE.

Table 30. Descriptions of the entries that are in the generated log file

Entry	Description
DateTime	The date and time that is calculated by the logger interceptor before the service invocation and after the service response. From V3.0.16.0, the time is changed from 12 to 24 hour format and extended to millisecond precision.
ThreadID	The ID of the thread under which the service request is being processed.
UserName	The user name for which the request is being processed.
RequestID	The request tracking ID that is generated by z/OS Connect EE.
RemoteAddress	The Internet Protocol (IP) address of the client who originated the request or last proxy that sent the request.
LocalAddress	The Internet Protocol (IP) address of the interface on which the request was received.
MessageType	Identifies whether the payload is from a request or a response.
MessageSize	The character size of the payload.
MessageData	A request or response payload.

The file system logger interceptor is configured by a `zosconnect_fileSystemloggerInterceptor` entry in `server.xml`. For more information, see “[zosconnect_fileSystemloggerInterceptor](#)” on page 783 in the *Reference* section.

Procedure

1. Configure the `zosconnect_fileSystemloggerInterceptor` element globally.

```

<zosconnect_fileSystemloggerInterceptor id="globalFileSystemLogger" logName="globalLog_%SERVERNAME%" />
  <zosconnect_zosConnectInterceptors id="globalInterceptorList"
interceptorRef="globalFileSystemLogger" />
  <zosconnect_zosConnectManager globalInterceptorsRef="globalInterceptorList" />

```

In the example, all z/OS Connect EE API and service requests are logged in a file called `globalLog_myServer1_yyyy-MM-dd_HH_mm_ss_SSS.log`, where `myServer1` is the name of the server. The only required configuration element is the `logName` attribute definition. The configuration element accepts a `%SERVERNAME%` string that is replaced with the name of the server when the log is created. The default log file location is `${server.output.dir} /logs/zosConnect`. For more information about available configuration attributes and default values, see [“Configuration elements” on page 753](#) in the *Reference* section.

You can also configure the `fileSystemloggerInterceptor` element for specific APIs or services. For example,

```

<zosconnect_fileSystemloggerInterceptor id="serviceYFileSystemLogger"
  logName="service1Log"
  logPath="/zosConnect/logs"
  logOption="RESPONSE"
  maxPayloadSize="30720" />
<zosconnect_zosConnectInterceptors id="serviceYInterceptorList" interceptorRef="serviceYFileSystemLogger" />
<zosconnect_zosConnectService serviceName="serviceY" serviceRef="serviceY"
interceptorsRef="serviceYInterceptorList"/>
<usr_myService id="serviceY"/>

```

In the example, information is logged in a file called `service1Log_yyyy-MM-dd_HH_mm_ss_SSS.log`, which is located in the path `/zosConnect/logs`. The log path is a fully qualified path. Only response data is logged for all incoming requests that target `serviceY`. The maximum JSON payload is configured to be 30,720 characters, so any JSON response payload that is greater than 30,720 characters is truncated to the configured maximum payload size.

2. Optional: Configure the `bufferedLogging` and the `bufferSize` attribute definitions to enable buffered logging. The default buffer size is 8 KB. All records in the buffer are flushed to the disk when the buffer becomes full. Using buffered logging is appropriate when performance is a concern and when it is acceptable for a possible loss of records from the buffer during a failure condition.

```

<zosconnect_fileSystemloggerInterceptor id="globalFileSystemLogger"
  logName="globalLog_%SERVERNAME%"
  bufferedLogging="true"
  bufferSize="16384" />

```

3. Optional: Configure the `rollOffLogPolicy` attribute. This policy states that when a new file is created based on when the active log file reaches the specified or default file size of 50 MB, or when the specified or default time of 24 hours expires since the active log file was created. In the following example, a new file is created when the file reaches 16 KB. The naming schema is the same: `globalLog_myServer1_yyyy-MM-dd_HH_mm_ss_SSS.log`. The difference between the newly created log files is the timestamp that is used when the new file is created:

```

<zosconnect_fileSystemloggerInterceptor id="globalFileSystemLogger"
  logName="globalLog_%SERVERNAME%"
  rollOffLogPolicy="SIZE"
  rollOffLogPolicySize="16384" />

```

Configuring the audit interceptor

You can use the audit interceptor that is included with z/OS Connect EE to write SMF 123 records, which contain data that can be used to audit and analyze requests.



CAUTION: SMF 123 subtype 1 version 2 records are not a compatible extension of the version 1 records. Therefore, before you select version 2 of the record, check that the tools you use to format SMF 123 records support the version 2 format, or formatting will fail.

Enabling recording of SMF 123 records

To generate SMF 123 records, you must include record type 123 in your SMF configuration options or the `_smf_record` function fails with return code JRSMFNotAccepting (0x406). For example, include "123" in your `SYS(TYPE(...,123,...))` statement, in PARMLIB. For more information, see [Statements and parameters for SMFPRMxx](#).

Note: For z/OS Connect EE to write to SMF, the user ID under which z/OS Connect EE runs, must have READ access to the RACF BPX.SMF profile of the FACILITY class. An example of the required command syntax is shown here:

```
PERMIT BPX.SMF CLASS(FACILITY) ACCESS(READ) ID(USERID)
```

For more information, see [Defining z/OS UNIX users to RACF](#) in the *z/OS UNIX System Services Planning* documentation.

Configuring the audit interceptor

By default, the audit interceptor writes SMF subtype 1 version 1 records. Set the attribute `apiProviderSmfVersion` to 2 to record version 2 records for API provider and administration requests. You can enable the audit interceptor globally for the z/OS Connect EE server or specifically for individual APIs, services, and API requesters.

Note: If the audit interceptor is enabled both globally and specifically on an individual API, service, or API requester, then two SMF 123 records are written for that request, unless the `runGlobalInterceptors` attribute is set to false on the relevant specific definition.

The following example shows how to enable the audit interceptor globally to capture records SMF 123 subtype 1 version 2 records for all API provider and administration requests, and by default version 1 records for all API requester requests:

```
<!-- Audit interceptor configuration -->
<zosconnect_auditInterceptor id="auditInterceptor" sequence="1" apiProviderSmfVersion="2"/>

<!-- Interceptor list configuration -->
<zosconnect_zosConnectInterceptors id="interceptorList1" interceptorRef="auditInterceptor"/>

<!-- Enable audit interceptor globally -->
<zosconnect_zosConnectManager globalInterceptorsRef="interceptorList1"/>
```

The following example shows how to enable the audit interceptor for a specific API, service, or API requester:

```
<!-- Enable interceptor for API "myAPI" -->
<zosconnect_zosConnectAPIs>
  <zosConnectAPI name="myAPI" interceptorsRef="interceptorList1"/>
</zosconnect_zosConnectAPIs>

<!-- Enable interceptor for service "myService" -->
<zosconnect_services>
  <service name="myService" interceptorsRef="interceptorList1"/>
</zosconnect_services>

<!-- Enable the interceptor for API Requester myAPIrequester -->
<zosconnect_apiRequesters>
  <apiRequester name="myAPIrequester" interceptorsRef="interceptorList1"/>
</zosconnect_apiRequesters>
```

Configuring interceptor sequence

Use the **sequence** parameter to control the sequence in which interceptors are called. For more information, see ["Configuring interceptors" on page 288](#).

Capturing request and response headers in SMF data

If you set `apiProviderSmfVersion` to 2, then it is possible to capture request and mapped response headers for a given request. Only response headers that are defined in response data mapping can be captured. For more information, see [“Defining and mapping headers, query parameters, or path parameters” on page 581](#). The following example shows how to specify a comma-separated list of three request header names and two response header names. A maximum of four header names can be specified for each. Replace the header names in this example with the names of the headers whose values you want to capture:

```
<zosconnect_auditInterceptor id="interceptorList1" sequence="1" apiProviderSmfVersion="2"
    apiProviderRequestHeaders="Request-Name-1, Request-Name-2, Request-Name-3"
    apiProviderResponseHeaders="Response-Name-1, Response-Name-2"/>
```

If each specified header is present on the request or response, the header name and header value are stored in the relevant SMF record field, `SMF123S1_REQ_HDRn` or `SMF123S1_RESP_HDRn` in the following format:

```
<headerName>:<headerValue>
```

For more information, see [“zosconnect_auditInterceptor” on page 757](#) in the *Reference* section of this documentation.

Configuring the authorization interceptor

The authorization interceptor is used by the z/OS Connect EE security functions.

The authorization interceptor controls access to z/OS Connect EE artifacts by assigning users one of four authorization levels, Admin, Operations, Invoke and Reader. These authorization levels are set in the global, API, service, or API requester configuration elements.

For more information, see [“Overview of z/OS Connect EE security” on page 331](#) and the sub-topics relating to security authorization.

Configuring service-specific interceptors

The z/OS Connect EE administrator can configure a user-defined set of interceptors for a service.

The `zosconnect_service` and `zosconnect_zosConnectService` elements, each represent a single service definition that is made available at run time by the server that is associated with a `server.xml` configuration file.

The `zosconnect_service` and `zosconnect_zosConnectService` elements each support an **interceptorsRef** attribute, which can be used by the z/OS Connect EE administrator to configure the service with a user-defined set of interceptors. These interceptors are invoked when a request is matched to that service for invocation (`preInvoke`), and before the result is returned to the caller (`postInvoke`).

The following example shows two methods for configuring services with user-defined interceptors.

```
<zosconnect_services>
    <service name="HealthService" interceptorsRef="medstaffInterceptorList" />
    <service name="FooService" interceptorsRef="adminInterceptorList" />
    <service name="BarService" interceptorsRef="patientInterceptorList" />
</zosconnect_services>
```

The following example shows how to configure interceptors for specific old style services that are not defined by service archive files.

```
<zosconnect_zosConnectService id="HealthService"
    serviceName="recordOpsCreate"
    serviceRef="HealthServiceEndpoint"
    interceptorsRef="medstaffInterceptorList" />
<zosconnect_zosConnectService id="FooService"
    serviceName="recordOpsUpdate"
    serviceRef="FooServiceEndpoint"
    interceptorsRef="adminInterceptorList" />
```

```
<zosconnect_zosConnectService id="BarService"
    serviceName="recordOpsDelete"
    serviceRef="BarServiceEndpoint"
    interceptorsRef="patientInterceptorList" />
```

This example implies the existence of three `zosConnectInterceptor` definitions that are located in `server.xml`:

```
<zosconnect_zosConnectInterceptors id="medstaffInterceptorList" interceptorRef="...."
<zosconnect_zosConnectInterceptors id="adminInterceptorList" interceptorRef="...."
<zosconnect_zosConnectInterceptors id="patientInterceptorList" interceptorRef="...."
```

For more information, see [Defining interceptors](#).

Note: Interceptors that are configured for services are only triggered if the service is invoked directly from a HTTP(S) request, they are not triggered if the service is invoked from an API.

Excluding specific services from the Global interceptors

The `runGlobalInterceptors` attribute accepts the values `true` or `false`, and defaults to `true`. You can use this attribute to exclude individual services from a configured global interceptor. For example, the following configurations would exclude the `HealthService` service from any global interceptor definition.

```
<zosconnect_services>
    <service name="HealthService" interceptorsRef="medstaffInterceptorList"
        runGlobalInterceptors="false" />
    <service name="FooService" interceptorsRef="adminInterceptorList" />
    <service name="BarService" interceptorsRef="patientInterceptorList" />
</zosconnect_services>
```

or

```
<zosconnect_zosConnectService id="HealthService"
    runGlobalInterceptors="false"
    serviceName="recordOpsCreate"
    serviceRef="HealthServiceEndpoint"
    interceptorsRef="adminInterceptorList" />
<zosconnect_zosConnectService id="FooService"
    serviceName="recordOpsUpdate"
    serviceRef="FooServiceEndpoint"
    interceptorsRef="adminInterceptorList" />
<zosconnect_zosConnectService id="BarService"
    serviceName="recordOpsDelete"
    serviceRef="BarServiceEndpoint"
    interceptorsRef="patientInterceptorList" />
```

Configuring API-specific interceptors

The z/OS Connect EE administrator can configure a user-defined set of interceptors for an API.

The `zosConnectAPI` element represents a single API definition that is made available at run time by the server that is associated with a `server.xml` configuration file.

The `zosConnectAPI` element accepts an `interceptorsRef` attribute, which can be used by the z/OS Connect EE administrator to configure the API with a user-defined set of interceptors. These interceptors are invoked whenever a request is matched to that API for invocation (`preInvoke`), and before the result is returned to the caller (`postInvoke`). For example:

```
<zosconnect_zosConnectAPIs location="">
    <zosConnectAPI name="Health" interceptorsRef="medstaffInterceptorList" />
    <zosConnectAPI name="Foo" interceptorsRef="adminInterceptorList" />
    <zosConnectAPI name="Bar" interceptorsRef="patientInterceptorList" />
</zosconnect_zosConnectAPIs>
```

This example implies the existence of three `zosConnectInterceptor` definitions that are located in `server.xml`:

```
<zosconnect_zosConnectInterceptors id="medstaffInterceptorList" interceptorRef="....."
<zosconnect_zosConnectInterceptors id="adminInterceptorList" interceptorRef="....."
<zosconnect_zosConnectInterceptors id="patientInterceptorList" interceptorRef="....."
```

For more information, see [Defining interceptors](#).

Excluding specific APIs from the Global interceptors

The `runGlobalInterceptors` element accepts values `true` or `false`, and defaults to `true`. You can use this element to exclude individual APIs from a configured global interceptor. For example, the following configuration would exclude the `Health` API from any global interceptor definition:

```
<zosconnect_zosConnectAPIs location="">
  <zosConnectAPI name="Health" interceptorsRef="medstaffInterceptorList" runGlobalInterceptors="false" />
  <zosConnectAPI name="Foo" interceptorsRef="adminInterceptorList" />
  <zosConnectAPI name="Bar" interceptorsRef="patientInterceptorList" />
</zosconnect_zosConnectAPIs>
```

Configuring API requester-specific interceptors

The z/OS Connect EE administrator can configure a user-defined set of interceptors for an API requester.

Interceptors that are called for an API requester must implement the `com.ibm.zosconnect.spi.InterceptorRequester` Service Provider Interface (SPI) that is provided by z/OS Connect EE.

The `apiRequester` element represents a single API requester definition that is made available at run time by the server that is associated with a specific `server.xml` configuration file. This element accepts an `interceptorsRef` attribute, which can be used by the z/OS Connect EE administrator to configure the API requester with a user-defined set of interceptors. These interceptors are invoked whenever a request is matched to that API requester for invocation (`preInvokeRequester`), and before the result is returned to the caller (`postInvokeRequester`). For example,

```
<zosconnect_apiRequesters>
  <apiRequester name="apiRequester1" interceptorsRef="interceptorList1" />
  <apiRequester name="apiRequester2" interceptorsRef="interceptorList2" />
  <apiRequester name="apiRequester3" interceptorsRef="interceptorList3" />
</zosconnect_apiRequesters>
```

This example implies the existence of three `zosConnectInterceptor` definitions, which are located in `server.xml`:

```
<zosconnect_zosConnectInterceptors id="interceptorList1" interceptorRef="....."
<zosconnect_zosConnectInterceptors id="interceptorList2" interceptorRef="....."
<zosconnect_zosConnectInterceptors id="interceptorList3" interceptorRef="....."
```

For more information, see [Defining interceptors](#).

Excluding specific API requesters from the Global interceptors

The `runGlobalInterceptors` element accepts values `true` or `false`, and defaults to `true`. You can use this element to exclude individual API requesters from a configured global interceptor. For example, the following configuration would exclude the `apiRequester1` API requester from any global interceptor definition:

```
<zosconnect_apiRequesters>
  <apiRequester name="apiRequester1" interceptorsRef="interceptorList1"
runGlobalInterceptors="false" />
  <apiRequester name="apiRequester2" interceptorsRef="interceptorList2" />
  <apiRequester name="apiRequester3" interceptorsRef="interceptorList3" />
</zosconnect_apiRequesters>
```

Configuring Data Transformers

IBM z/OS Connect EE provides the ability to optionally transform request and response payloads that are used for calling a business asset on z/OS operating systems. You can create message payload transformers to satisfy specific needs by implementing the `com.ibm.zosconnect.spi.DataXform` Service Provider Interface (SPI), which is included with z/OS Connect EE.

About this task

z/OS Connect EE provides an implementation that requires the request and response message format be JSON. This product supports the conversion of the request to a byte array, which can be mapped by a native language COBOL, PL/I, or C structure. This language structure of the target program, or copy book, includes a description of the `in` and `out` parameters. The language structure is used by a supplied utility to generate a binding file and JSON request and response schema files. The bind file that is generated by this utility is used by z/OS Connect EE to complete the data conversion to and from JSON and native data formats as requests arrive and responses are returned. You can retrieve the JSON schemas for the request and response message with a RESTful API call that is provided by z/OS Connect EE.

z/OS Connect EE provides the `zosConnectService` configuration element that enables the administrator to configure a set of attributes that apply to a particular service. One of these attributes is `dataXformRef`, which points to a data transformation configuration that is to be used for a specific service. This task describes how the data transformer supplied with z/OS Connect EE is used.

Note:

1. In the examples shown in steps 1 & 2, the `serviceName` and `bindFileSuffix` must match the `bindfile` found in `bindFileLoc`. For example: `record0psCreate.wsbind` resides in `/u/bindfiles`.
2. The request and response schema file names must match the `serviceName` specified in the `zosConnectService` element with `_request` or `_response` appended. The values, including the file type, must match the `requestSchemaSuffix` or `responseSchemaSuffix` respectively. For example: `record0psCreate_request.json` and `record0psCreate_response.json`. These files must reside in `/u/json`.

Procedure

1. Update the `zosConnectService` element for each service in your `server.xml` configuration for which you want to enable the data transformation supplied by z/OS Connect EE.

```
<!-- z/OS Connect service definition -->
<zosconnect_zosConnectService id="zcs1"
    serviceName="record0psCreate"
    serviceRef="wola0psCreateService"
    dataXformRef="xformJSON2byte"/>
```

2. Create the associated `zosConnectDataXform` element.

```
<!-- z/OS Connect data transformation provider -->
<zosconnect_zosConnectDataXform id="xformJSON2byte"
    bindFileLoc="/u/bindfiles" bindFileSuffix=".wsbind"
    requestSchemaLoc="/u/json" responseSchemaLoc="/u/json"
    requestSchemaSuffix=".json" responseSchemaSuffix=".json"/>
```

3. Optional: Configure a data transformer that applies to all services.

Set the `globalDataXformRef` of the `zosConnectManager` element to the configured data transformer's ID that is intended for global use. If both global and service data transformers are defined and requests come in for a service with a configured data transformer, z/OS Connect EE will use the data transformer configured specifically for the service.

```
<zosconnect_zosConnectManager globalDataXformRef="globalDataXform"/>

<!-- z/OS Connect data transformation provider -->
<zosconnect_zosConnectDataXform id="globalDataXform"
    bindFileLoc="/u/bindfiles" bindFileSuffix=".wsbind"
```

```
requestSchemaLoc="/u/json" responseSchemaLoc="/u/json"  
requestSchemaSuffix=".json" responseSchemaSuffix=".json"/>
```

Creating bind and schema files

z/OS Connect EE provides the ability to optionally transform request and response payloads that are used for calling a business asset on z/OS operating systems. z/OS Connect EE supplies two new utilities called **BAQLS2JS** and **BAQJS2LS**.

Before you begin

Before you create your binding and schema files, make sure that your setup complies with these conditions:

- Your high-level language data structures must meet the following criteria:
 - The data structures must be defined separately from the source program. For example, in a COBOL copybook.
 - If your PL/I or COBOL application program uses different data structures for input and output, the data structures must be defined in two different members in a partitioned data set. If the same structure is used for input and output, the structure must be defined in a single member.
 - For C and C++, your data structures can be in the same member in a partitioned data set.
- The language structures must be available in a partitioned data set.
- You must define to Open Multiple Virtual Storage (OMVS) the user ID that **BAQLS2JS** or **BAQJS2LS** uses to run.
- The user ID must have read permission to z/OS UNIX and PDS libraries, and write permission to the directories that are specified on the LOGFILE, WSBIND, and **JSON-SCHEMA-REQUEST** and **JSON-SCHEMA-RESPONSE** output parameters.
- The user ID must have a sufficiently large storage allocation to run Java. You can use any supported version of Java. The **BAQLS2JS** and **BAQJS2LS** utilities use the Java version that is specified by **JAVA_HOME** in BAQLS2JS and BAQJS2LS sample JCL. Otherwise, the ID uses the Java version specified on the **PATH** statement.

About this task

The **BAQLS2JS** utility reads a COBOL copybook, PLI structure, or C structure file and generates a binding file and JSON schema files. The **BAQJS2LS** utility reads a JSON schema and generates the corresponding binding file and language structure file. The utilities are similar to the existing **DFHLS2JS** and **DFHJS2LS** tools, which are part of the CICS Transaction Server Mobile Extensions feature pack.

Procedure

1. Copy the BAQLS2JS or BAQJS2LS sample JCL to a writeable directory or dataset
2. Use the **BAQLS2JS** procedure to generate a z/OS Connect EE service binding file from a language structure.

You will need to provide JCL to invoke the **BAQLS2JS** procedure with the input parameters. Refer to the **BAQLS2JS** reference documentation for information on the input parameters and an example job to help you use the procedure. When you submit the **BAQLS2JS** procedure, the utility generates the service binding file to the location that you specified with the WSBIND parameter. The generated JSON schemas are placed in the location that you specified with the **JSON-SCHEMA-REQUEST** and **JSON-SCHEMA-RESPONSE** parameters. For more information, see [“Conversion for z/OS Connect Enterprise Edition data transformation” on page 606](#).

Note: In order to generate a service archive file, **SERVICE-ARCHIVE** and **SERVICE-NAME** must be specified in **BAQLS2JS**.

3. Review the generated JSON schema.

These schemas are used to define the input and output data formats for interacting with the z/OS Connect EE service. The application developer must use these schemas when creating an application to call the service and pass the JSON request payload.

Note: Changing the generated schema invalidates the generated binding file at WSBIND. If you want to change the schema, for example, to rename the fields within the schema, you must use **BAQJS2LS** to generate a new binding file, and a new set of language structures. The application program must be re-compiled to use the new language structures.

Configuring service archives

You can add the `zosconnect_services` element to `server.xml` and configure where service archive files for the REST client service provider are stored.

You copy a service archive (.sar) file into a directory where it can be accessed during API development. The definition of the service must be added to the `server.xml` configuration file before it can be used.

The `zosconnect_services` element in the configuration file defines the z/OS Connect EE services to be deployed. For example,

```
<zosconnect_services location="/var/zosconnect/services" updateTrigger="polled" pollingRate="100ms">
  <service name="AddCustomer" />
  <service name="InquireCustomer" />
  <service name="UpdateCustomer" />
</zosconnect_services>
```

The `server.xml` configuration file can contain the following elements:

`zosconnect_services`

- Only one `zosconnect_services` element is allowed.
- The **location** parameter is optional and specifies where the service archive files are deployed. If it is not defined, the `${server.config.dir} /resources/zosconnect/services` directory is used as the service archive file installation directory. When the server is started, all the service archive files are deployed in the server.
- The **updateTrigger** parameter is optional and controls when z/OS Connect EE is notified about changes in the services directory. The default value of the **updateTrigger** parameter is disabled. For more information, see [zosconnect_services](#) in the *Reference* section.

`service`

- One or more `service` elements can be contained in the `zosconnect_services` element.
- Supports other optional attributes that, for example, can be used to define authentication, authorization, or interceptors specific to a service. For more information, see [“Configuration elements” on page 753](#).
- The attributes `adminGroup`, `invokeGroup`, `operationsGroup`, `readerGroup`, `requireAuth`, and `requireSecure` are available to specify authentication and authorization for a service. For more information, see [Chapter 9, “Securing z/OS Connect EE resources,” on page 331](#).
- The attributes `interceptorsRef` and `runGlobalInterceptors` are available to specify interceptors for a service. For more information, see [“Configuring API-specific interceptors” on page 294](#).

`property`

- One or more `property` elements can be contained in the `service` element.
- Defines optional properties for the service provider.
- Properties can only be defined by editing the `server.xml` file.
- Properties remain in the configuration file after the service archive file is moved or deleted.

Collection ID and connection reference can be specified in `server.xml` to override the service archive properties in different server environments. For more information, see [“Overriding Db2 service properties in the server configuration” on page 286](#).

Related concepts

[“Administering service archives” on page 670](#)

Use this information to learn how to manage your service archives.

Configuring APIs

How to add the `zosconnect_zosConnectAPIs` element to `server.xml` and configure where API archive files are stored.

Note: For information about deploying APIs with the Administration Interface, see [Chapter 23, “How to manage APIs,” on page 673](#).

The `zosconnect_zosConnectAPIs` element can override the default settings for APIs including the location where the API archive files are stored. This location is where they reside on the file system, not where they are deployed. For example,

```
<zosconnect_zosConnectAPIs location="/var/zosconnect/apis" updateTrigger="polled">
  <zosConnectAPI name="Health" />
  <zosConnectAPI name="Inventory" />
  <zosConnectAPI name="Ordering" />
</zosconnect_zosConnectAPIs>
```

The `server.xml` configuration file can contain the following elements:

zosconnect_zosConnectAPIs

- Only one `zosconnect_zosConnectAPIs` element is allowed.
- The **location** parameter is optional and specifies where the API archive files are deployed. If it is not defined, the `${server.config.dir} /resources/zosconnect/apis` directory is used as the API archive file installation directory. When the server is started, all the API archive files in the location directory are deployed in the server.

Note:

1. The server does not create the API archive file installation directory. If the directory does not exist, APIs cannot be dynamically deployed by the API toolkit or RESTful administration interface. Either change the location to a directory that exists and restart the server, or create the required directories.
 2. The **location** parameter cannot be changed while the server is running; its value is set when the server is started.
- The **updateTrigger** parameter is optional and controls when z/OS Connect EE is notified about changes in the `apis` directory. The `apis` directory is specified on the **location** parameter. For more information, see [“Configuration elements” on page 753](#).
 - You can use the nested `zosConnectAPI` elements to specify the APIs that are expected to be in the location directory. If a defined API is not found in the location directory, a message is written to the log. If an API archive file is invalid, then the API fails to load and an error is written to the log.

zosConnectAPI

- One or more `zosConnectAPI` elements can be contained in the `zosconnect_zosConnectAPIs` element.
- Supports other optional attributes that, for example, can be used to define authentication, authorization, or interceptors specific to an API. For more information, see [“Configuration elements” on page 753](#).
- The attributes `adminGroup`, `invokeGroup`, `operationsGroup`, `readerGroup`, `requireAuth`, and `requireSecure` are available to specify authentication and authorization for an API. For more information, see [Chapter 9, “Securing z/OS Connect EE resources,” on page 331](#).

- The attributes `interceptorsRef` and `runGlobalInterceptors` are available to specify interceptors for an API. For more information, see “[Configuring API-specific interceptors](#)” on page 294.

Note: The `zosConnectAPI` element can be dynamically updated while the server is running, provided the server is enabled for dynamic configuration. For more information about enabling the server for dynamic configuration, see [Administering Liberty](#) in the *WebSphere Application Server Liberty* documentation.

Related information

[“How to manage APIs” on page 673](#)

Configuring Cross-Origin Resource Sharing on a z/OS Connect Enterprise Edition Server

z/OS Connect EE supports Cross-Origin Resource Sharing (CORS). CORS is a mechanism that allows access to a resource from a different domain than the one in which the resource is located.

CORS is enabled in z/OS Connect EE by using the Liberty `cors` configuration element. For more information, see [Configuring Cross Origin Resource Sharing on a Liberty server](#) in the *WebSphere Application Server for z/OS Liberty* documentation.

To enable API toolkit connections, you must have a `cors` definition in your `server.xml`. When you create a server by using one of the server templates supplied with the product, a `cors` element that enables API toolkit connections, is automatically added to the `server.xml` configuration file. The following excerpt from the `server.xml` shows a `cors` element definition that allows API toolkit connections.

```
<cors id="defaultCORSConfig"
      domain="/"
      allowedOrigins="*"
      allowedMethods="GET, POST, PUT, DELETE, OPTIONS"
      allowedHeaders="Origin, Content-Type, Authorization, Cache-Control, Expires, Pragma"
      allowCredentials="true"
      maxAge="3600" />
```

You may want to customize this configuration further to, for example, restrict the allowed origins.

If you have JavaScript clients that access z/OS Connect EE resources, you may need to further customize your `cors` element definition. For example, the `allowedHeaders` attribute, on the `cors` element, must specify all headers that you want to be permitted on any request. The CORS access control headers themselves do not need to be listed.

Tip: If any problems occur, ensure that you gather all relevant diagnostic information by including the Liberty trace specification: `*=info:CorsService=all:GenericBNF=all`. For more information, see [“Enabling trace in z/OS Connect EE server” on page 715](#).

Configuration updates on demand

A running z/OS Connect EE server uses information from various files that are stored in UNIX System Services.

These files can be modified and the changes activated while the server is running. The following z/OS Connect EE files can be configured dynamically:

- Server configuration file.
- Data transformation files, if any of your z/OS Connect EE services use DataXForms.
- API archive files.
- API Requester archive files.
- Service archive files.
- Policies and rule sets.

If you need to modify, create or delete these files while the server is running, you can use one of these methods to activate the changes:

1. Use polling. Continuous polling uses CPU resources. This method can be useful in a development or test environment.
2. Use an MBean to trigger an update on demand. This method can be useful in a production environment.
3. Use the **Refresh Modify** command to activate all changes in all the files. This method can be useful in a production environment. For more information, see “[The MODIFY command](#)” on page 473.

The update mechanism is specified on the following server configuration elements. For more information, see the element description in the *Reference* section.

- Updates to the server configuration files are controlled by the `config` element.
- Updates to data transformation files are controlled by the “[zosconnect_zosConnectDataXform](#)” on page 782 element.
- Updates to APIs are controlled by the “[zosconnect_zosConnectAPIs](#)” on page 780 element.
- Updates to API Requesters are controlled by the “[zosconnect_apiRequesters](#)” on page 754 element.
- Updates to services are controlled by the “[zosconnect_services](#)” on page 774 element.
- Updates to policies and rule sets are controlled by the “[zosconnect_policy](#)” on page 773 element.

Using an MBean to trigger updates

An MBean called the `FileNotificationMBean` is provided, which can be called to trigger an update of a running server.

The MBean can make the following updates to the server:

- Use new, modified, or deleted configuration files.
- Use new, modified, or deleted data transformation files.
- Use new, modified, or deleted API archive files.
- Use new, modified, or deleted API requester archive files.
- Use new, modified, or deleted service archive files.
- Use new, modified, or deleted policy rule sets.

MBeans are invoked by using a Java Management Extensions (JMX) connector. Two supported JMX connectors are available:

- The local connector is enabled through the Liberty feature `localConnector-1.0`. Access through the local connector is protected by the policy that is implemented by the SDK in use. The client must run on the same host as the server, and under the same user ID.
- The REST connector is enabled through the Liberty feature `restConnector-1.0`. Remote access through the REST connector is protected by a single administrator role. In addition, SSL is required to keep the communication confidential.

You must configure the JMX connectors as described in the topics [Configuring local JMX connection to Liberty](#) or [Configuring secure JMX connection to Liberty](#) in the *WebSphere Application Server* documentation.

Note: If you intend to use the REST connector with SAF authentication, the following steps are required.

- Instead of following the link to the topic *Map to the administrator role for Liberty*, see the alternative topic [Mapping the administrator role for Liberty on z/OS](#).
- Define the following RACF definitions (or equivalent definitions for your SAF provider).

```
RDEFINE APPL <SAF_profilePrefix> UACC(NONE) SETROPTS CLASSACT(APPL)  
PERMIT <SAF_profilePrefix> CLASS(APPL) ACCESS(READ) ID(<id>)
```

Where `<id>` is the user or group name of the SAF user ID you are granting administrator authority to, and `<SAF_profilePrefix>` is the value of the `profilePrefix` attribute specified on the server configuration file element `safCredentials`. The default `profilePrefix` value is `BBGZDFLT`.

For more information, see [Liberty: Accessing z/OS security resources using WZSSAD in the WebSphere Application Server documentation](#).

MBeans can be called from Java-based client applications. For example, a Java program, Jython script, or a REST API.

Invoking the FileNotificationMBean from a Java program

Javadoc for the `FileNotificationMBean` is provided in the file `<installation_path>/wlp/dev/api/ibm/javadoc/com.ibm.websphere.appserver.api.basics_1.2-javadoc.zip`.

Before studying this topic, you should be familiar with the information in [“Using an MBean to trigger updates” on page 301](#).

The following Java code snippet provides an example of using the REST connector to invoke the `FileNotificationMBean` method `notifyFileChanges`. This example updates the running server configuration with changes that are made to either of two server configuration files: `/var/zosconnect/servers/serverName/server.xml` and `/var/zosconnect/servers/serverName/includeConfig.xml`.

```
/** z/OS Connect server hostname */
private final static String hostname = "companyhost.company.com";
/** z/OS Connect server HTTPS port */
private final static String httpsPort = "9443";
/** z/OS Connect server administration role user */
private final static String adminUserId = "adminUser";
/** z/OS Connect server administration role user's password */
private final static String adminPassword = "adminPswd";
/** Truststore containing z/OS Connect server's public certificate */
private final static String trustStoreLocation = "/mydir/trust.jks";
/** Truststore password */
private final static String trustStorePassword = "truststorePswd";
/** Absolute paths of configuration files to monitor */
private final static String configFilePath1 = "/var/zosconnect/servers/serverName/server.xml";
private final static String configFilePath2 = "/var/zosconnect/servers/serverName/includeConfig.xml";

/** FileNotificationMBean name */
private final static String MBEAN_FILE_NOTIFICATION =
        "WebSphere:service=com.ibm.ws.kernel.filemonitor.FileNotificationMBean";
/** FileNotificationMBean method notifyFileChanges signature */
private final static String[] MBEAN_FILE_NOTIFICATION_NOTIFYFILECHANGES_SIGNATURE = new String[] {
    Collection.class.getName(),
    Collection.class.getName(),
    Collection.class.getName() };
/** JMX service URL*/
private static String jmxServiceURL = "service:jmx:rest://" + hostname + ":" + httpsPort +
        "/IBMJMXConnectorREST";
...
public void invokeFileNotificationMBean() {
...
    // Get secure remote JMX MBean server connection
    System.setProperty("javax.net.ssl.trustStore", trustStoreLocation);
    System.setProperty("javax.net.ssl.trustStorePassword", trustStorePassword);
    HashMap<String, Object> environment = new HashMap<String, Object>();
    environment.put("jmx.remote.protocol.provider.pkgs", "com.ibm.ws.jmx.connector.client");
    environment.put("com.ibm.ws.jmx.connector.client.disableURLHostnameVerification",
Boolean.TRUE);
    environment.put(JMXConnector.CREDENTIALS, new String[] { adminUserId , adminPassword });
    JMXServiceURL url = new JMXServiceURL(jmxServiceURL);
    JMXConnector jmxConnector = JMXConnectorFactory.newJMXConnector(url, environment);
    jmxConnector.connect();
    MBeanServerConnection mbsc = jmxConnector.getMBeanServerConnection();

    // Invoke FileNotificationMBean
    ObjectName fileMonitorMBeanName = new ObjectName(MBEAN_FILE_NOTIFICATION);
    if (mbsc.isRegistered(fileMonitorMBeanName)) {
        // Create a list of absolute paths of each file to be checked
    }
}
```

```

List<String> modifiedFilePaths = new ArrayList<String>();
modifiedFilePaths.add(configFilePath1);
modifiedFilePaths.add(configFilePath2);

// Set MBean method notifyFileChanges parameters (createdFiles, modifiedFiles, deletedFiles)
Object[] params = new Object[] { null, modifiedFilePaths, null };

// Invoke FileNotificationMBean method notifyFileChanges
mbsc.invoke(fileMonitorMBeanName, "notifyFileChanges", params,
MBEAN_FILE_NOTIFICATION_NOTIFYFILECHANGES_SIGNATURE);
}
else {
    System.err.println("MBean invoke request failed " + MBEAN_FILE_NOTIFICATION + " is
not registered.");
}
...
// Close the JMX connection
jmxConnector.close();
...
}

```

Note: The client-side REST connector, which is provided in `wlp/clients/restConnector.jar`, must be in the class path that is used to compile and run a Java application that invokes an MBean.

Invoking FileNotificationMBean from the REST API

MBeans can also be invoked by using a REST interface.

Before studying this topic, you should be familiar with the information in [“Using an MBean to trigger updates” on page 301](#).

This method is described in the [WASdev article Accessing Liberty's JMX REST APIs](#). As described in that article, if the JMX Connector is installed, you can get help on the syntax that is required to invoke an MBean by making an HTTP GET call to `http://<hostname>:<port>/IBMJMXConnectorREST` and selecting the MBean operation invocation and JSON grammar for POJO options from the menu.

The following instructions provide an example of using the REST API to invoke the `FileNotificationMBean` method `notifyFileChanges`. This example updates the running server configuration with changes that are made to either of two server configuration files: `/var/zosconnect/servers/serverName/server.xml` and `/var/zosconnect/servers/serverName/includeConfig.xml`.

Issue an HTTP POST request to `http://<hostname>:<port>/IBMJMXConnectorREST/mbeans/WebSphere%3Aservice%3Dcom.ibm.ws.kernel.filemonitor.FileNotificationMBean/operations/notifyFileChanges` with a HTTP header `: Content-Type application/json` and a request JSON payload as in the following sample.

```
{
  "params": [
    {
      "value" : [],
      "type" : {"className":"java.util.ArrayList","items":["java.lang.String"]}
    },
    {
      "value" : ["/var/zosconnect/servers/serverName/server.xml", "/var/zosconnect/servers/serverName/includeConfig.xml"],
      "type" : {"className":"java.util.ArrayList","items":["java.lang.String", "java.lang.String"]}
    },
    {
      "value" : [],
      "type" : {"className":"java.util.ArrayList","items":["java.lang.String"]}
    }
  ],
  "signature": [
    "java.util.Collection",
    "java.util.Collection",
    "java.util.Collection"
  ]
}
```

Using the FileTransferMBean

The FileTransferMBean can be used to transfer your API archive files from a remote system to the z/OS Connect EE API location directory.

Before studying this topic, you should be familiar with the information in [“Using an MBean to trigger updates” on page 301](#).

The FileTransferMBean contains methods that you can use to download, upload, or delete files on the z/OS Connect EE server instance. This MBean is available as part of the `restConnector-1.0` feature. Javadoc is provided in the file `<installation_path>/wlp/dev/api/ibm/javadoc/com.ibm.websphere.appserver.api.restConnector_1.1-javadoc.zip`.

Note: To use this MBean, you must also define the `remoteFileAccess` server configuration element attributes `readDir` and `writeDir` to define the directories to which the `restConnector` within that server is authorized to read to and write from.

For more information about this element, see [Remote File Access \(remoteFileAccess\)](#) in the *WebSphere Application Server* documentation.

Configuring for API requesters

You must configure both the z/OS Connect EE server and the z/OS subsystem to enable a z/OS application to call RESTful APIs through z/OS Connect EE.

You can also configure a high availability (HA) environment to handle RESTful API calls from z/OS applications.

Note: The API requester supports code page conversion between the runtime code page and UTF-8. The code page is automatically detected and no configuration is required.

Configuring z/OS Connect EE to support API requesters

To enable the z/OS application to call RESTful API through z/OS Connect EE, you must add related elements in `server.xml`.

Procedure

1. Open the `server.xml` configuration file for editing.
2. Add the `zosconnect:apiRequester-1.0` in the `featureManager` element.

For example,

```
<featureManager>
  <feature>zosconnect:apiRequester-1.0</feature>
</featureManager>
```

3. Add the `zosconnect_apiRequesters` element in the `server.xml` file.

The `zosconnect_apiRequesters` element defines the z/OS Connect EE API requesters that are to be deployed and, optionally, the API requesters that are expected to be deployed. For example,

```
<zosconnect_apiRequesters
  location="/var/zosconnect/servers/ServerA/resources/zosconnect/apiRequesters"
    updateTrigger="disabled">
  <apiRequester name="Book_Inventory" />
</zosconnect_apiRequesters>
```

The following elements can be configured in `server.xml`:

zosconnect_apiRequesters

- Only one `zosconnect_apiRequesters` element is allowed.

- The **location** parameter is optional and specifies where the API requester archive is deployed. If it is not defined, the `${server.config.dir} /resources/zosconnect/apiRequesters` directory is used as the API requester archive installation directory. When the server is started, all the API requester archives in the location directory are deployed in the server.

Note: The server does not create the API requester archive installation directory. Either change the location to a directory that exists and restart the server, or create the required directories.

- The **updateTrigger** parameter is optional and controls when z/OS Connect EE is notified about changes in the `apiRequesters` directory. The `apiRequesters` directory is specified on the **location** parameter. The default value of the **updateTrigger** parameter is disabled. If you want to enable the servers to automatically pick up the changes to the `apiRequesters` directory without the need to restart the server or invoke the `FileNotificationMbean`, the **updateTrigger** parameter must be set to polled. For more information, see “[zosconnect_apiRequesters](#)” on page 754.

Note: When polling is enabled, high CPU and I/O usage might be observed.

- You can use the nested `apiRequester` elements to specify the API requesters that are expected to be in the location directory. If a defined API requester is not found in the location directory, then a message is written to the log. If an API requester archive is invalid, then the API requester fails to load and an error is written to the log.

Note: The `zosconnect_apiRequesters` element cannot be changed while the server is running; its value is set when the server is started.

apiRequester

- One or more `apiRequester` elements can be contained in the `zosconnect_apiRequesters` element.
- The attribute **name** is available to specify the name of an API requester.

Important:

The name of an API requester is generated with an API requester archive file by the build toolkit. The following information from the Swagger definition of the API to be called is used to generate the name of an API requester:

- Info title: The name of the API
- Info version: The version of the API

For example, if the API you want to call has the following `info` section in the Swagger file:

```
"info": {
    "title": "My Pet store",
    "version": "2.1.0"
}
```

Then the API requester will have the name `My-Pet-store_2.1.0`. This name is associated with the API requester archive file to be used.

- The attributes `adminGroup`, `invokeGroup`, `operationsGroup`, `readerGroup`, `requireAuth`, and `requireSecure` are available to specify authentication and authorization for an API requester. For more information, see “[Overview of z/OS Connect EE security](#)” on page 331.
- The attribute `interceptorsRef` is available to specify interceptors for an API requester. For more information, see “[Configuring API requester-specific interceptors](#)” on page 295.
- The attribute **connectionRef** is available to specify the connection to the RESTful endpoint for an API requester. When the values of **connectionRef** both in the `apiRequester` element and in the build toolkit properties file are set, the former one is used to identify the connection to a request endpoint. For more information, see `apiRequester`.

Note: The `apiRequester` element can be dynamically updated while the server is running, provided the server is enabled for dynamic configuration. For more information about enabling the

server for dynamic configuration, see [Administering Liberty in the WebSphere Application Server Liberty documentation](#).

For information about deploying API requesters, see “[Deploying the API requester to z/OS Connect EE](#)” on page 631.

Configure CICS to access z/OS Connect EE to call APIs

To enable CICS applications to call APIs through z/OS Connect EE, enable the communication stub in the CICS region.

About this task

The z/OS Connect EE communication stub is a module that establishes HTTP connections with the z/OS Connect EE server, transfers data between z/OS applications and z/OS Connect EE, and handles status and return codes that are issued by z/OS Connect EE. Before your CICS applications can make RESTful API calls, you must configure the communication stub in your CICS region.

Tip: z/OS provides functions to optimize performance for HTTP connections. If z/OS Connect EE and the z/OS subsystem are on the same LPAR, the fast local socket is automatically available. If z/OS Connect EE and the z/OS subsystem are on different LPARs, you must enable the following functions:

- HiperSocket, a zSeries hardware feature that provides high-performance internal communications between LPARs within the same central processor complex (CPC). For more information, see [HiperSockets concepts and connectivity](#) in the z/OS documentation.
- Shared Memory Communication (SMC), which requires z/OS V2.2 or later on z13® in the same CPC. For more information, see [Shared Memory Communications](#) in the z/OS documentation.

To set up the z/OS Connect EE communication stub in CICS, you must have the following resources in your CICS region:

A TDQUEUE resource with the name BAQQ

This resource defines the transient data queue that is used by the z/OS Connect EE communication stub to log communication stub error messages. By default, messages are printed to DD BAQOUT in the CICS job log.

A URIMAP resource with the name BAQURIMP

This resource handles HTTP client requests from the communication stub to the z/OS Connect EE server.

A PROGRAM resource with the name BAQCSTUB

This resource defines attributes for the z/OS Connect EE communication stub program BAQCSTUB. By default, BAQCSTUB is defined as threadsafe.

Sample resources are supplied with z/OS Connect EE and are provided in a CSD group with the name BAQAPIR.

It is recommended that you use the resources as supplied. Based on the topology you want to use and your CICS and z/OS Connect EE environment, you might need to modify the attributes of the BAQURIMP resource definition.

Procedure

1. Import the BAQAPIR CSD group into your CICS system by using the DFHCSDUP utility program.
The supplied BAQAPIR CSD group is in the h1q.SBAQSAMP data set.
 - a) Edit the job control statements that you can use to invoke DFHCSDUP. Include a DD statement that references the h1q.SBAQSAMP data set.
 - b) Run the DFHCSDUP utility program.
2. Modify the BAQURIMP URIMAP resource definition, based on the topology you want to use and system environment.
 - a) Specify the host and port of z/OS Connect EE.

Note: The **HOST** attribute and the **PORT** attribute are mandatory attributes that you must specify based on your situation.

- b) To secure this connection with basic authentication or client authentication, see “[How to configure basic authentication from CICS](#)” on page 410 or “[How to configure TLS from CICS](#)” on page 397.
 - c) By default, connection pooling is enabled to keep the connection between CICS and the z/OS Connect EE server open as a persistent connection. It is set up with the **SOCKETCLOSE** attribute value set to 30s, which means CICS closes the socket when the socket keeps idle for 30s. If needed, you can specify the **SOCKETCLOSE** value of 0 to disable connection pooling. For more information about connection pooling in CICS, see [Connection pooling for HTTP client performance](#) in the CICS documentation.
3. Install the BAQQ, BAQURIMP, and BAQCSTUB resource definitions in your CICS system.
 4. Define and install a new CICS LIBRARY to include the h1q.SBAQLIB1 data set.

You have two options to do this:

- Add a DD statement that references the h1q.SBAQLIB1 data set in the CICS static load LIBRARY concatenation, DFHRPL.

Note that when CICS is running, changes to DFHRPL are not possible without stopping and restarting CICS.

- Use a dynamic LIBRARY concatenation to add the load library to CICS.
5. Setting timeout for CICS to send a request to and receive the response from the z/OS Connect EE server.

The z/OS Connect EE communication stub uses CICS web API to communicate with the z/OS Connect EE server. You can specify DTIMOUT on the TRANSACTION definition to control the timeout of sending a request to z/OS Connect EE and specify RTIMOUT on the PROFILE definition to control the timeout of waiting a response from z/OS Connect EE. For more information, see [WEB CONVERSE](#) in the CICS documentation.

Related concepts

[“Exposing z/OS assets as REST APIs” on page 487](#)

To access and act on a resource on a z/OS subsystem through REST APIs, create a service that defines how the JSON schemas for the request and response messages map to the resource. Then, design a REST API to define how an HTTP action such as GET, PUT, POST, or DELETE would act on the service.

Related tasks

[“Configuring z/OS Connect EE to support API requesters” on page 304](#)

To enable the z/OS application to call RESTful API through z/OS Connect EE, you must add related elements in `server.xml`.

Related information

[“Calling RESTful APIs from z/OS applications” on page 617](#)

Configuring enhanced monitoring for CICS tasks

To allow CICS applications to pass enhanced monitoring information to z/OS Connect EE, enable the CICS Task-Related User Exit (TRUE) in the CICS region. Enhanced monitoring information is then available to any interceptors that run in the z/OS Connect EE server and use the API requester interceptor interface.

About this task

The z/OS Connect EE provided CICS TRUE allows the UOWID and NETUOWID of the CICS task to be sent to the z/OS Connect EE server as enhanced monitoring information. Before your CICS applications can pass enhanced monitoring information, you must configure the TRUE in your CICS region.

To set up the z/OS Connect EE provided TRUE in CICS, you must have the following resources in your CICS region:

A PROGRAM resource with the name BAQCTENA

This program is used to enable the TRUE in the CICS region.

A PROGRAM resource with the name BAQCTDIS

This program is used to disable the TRUE in the CICS region.

A PROGRAM resource with the name BAQCTRUA

This program is called by the TRUE to gather the enhanced monitoring information.

A PROGRAM resource with the name BAQCTTRUE

This program is the implementation of the TRUE.

A TRANSACTION resource with the name BAQE

This transaction runs the program BAQCTENA.

A TRANSACTION resource with the name BAQD

This transaction runs the program BAQCTDIS.

Sample CICS CSD resources are supplied with z/OS Connect EE in the hlq.SBAQSAMP member BAQTRUE. These sample definitions use a CSD group with the name BAQAPIR. For best results, use the resources as supplied.

Procedure

1. Import the TRUE resources in the BAQAPIR CSD group into your CICS system by using the DFHCSDUP utility program.
The supplied TRUE resources are located in the BAQTRUE member in the hlq.SBAQSAMP data set.
 - a) Edit the job control statements that you can use to invoke DFHCSDUP. Include a DD statement that references the BAQTRUE member in the hlq.SBAQSAMP data set.
 - b) Run the DFHCSDUP utility program.
2. Install the TRUE resource definitions into your CICS system by installing the group BAQAPIR.
3. Ensure that SBAQLIB1 data set is available to CICS either in the DFHRPL concatenation or as a dynamic library. This step was completed as part of configuring CICS to access z/OS Connect EE to call APIs.
4. Enable the TRUE by running the BAQE transaction. This provides enhanced monitoring for any CICS applications that are calling z/OS Connect EE by using the BAQCSTUB in this CICS region.
 - a) If the TRUE can be enabled, then the BAQT0034I message is returned. If it cannot be enabled, then a warning or error message is returned and also written to the BAQQ TDQ.
5. CICS can be configured to enable the TRUE automatically at startup by using the sample program list table BAQPLTBQ located in the hlq.SBAQSAMP data set. Choose one of the following PLTPI configuration options:
 - If you do not have a program list table defined:
 - a. Add the **PLTPI=BQ** parameter to the CICS system initialization parameters.
 - b. Copy the BAQPLTBQ sample to a new data set and rename it to DFHPLTBQ.
 - c. Complete the necessary steps for your CICS version to make DFHPLTBQ available to CICS at startup.
 - If you already have a program list table defined and specified in your CICS system initialization parameters, add the BAQCTENA program to the list by including the following statement after the DFHDELIM entry:

```
DFHPLT TYPE=ENTRY,PROGRAM=BAQCTENA
```

This ensures that BAQCTENA is run in the second pass of PLTPI.

6. The TRUE can be disabled by running the BAQD transaction, which returns the BAQT0039I message if it completes successfully. If the TRUE can't be disabled, then a warning or error message is returned and also written to the BAQQ TDQ.

Configuring IMS to access z/OS Connect EE for API calls

To enable RESTful API calls for IMS applications through z/OS Connect EE, you must configure the API requester communication stub in your IMS system. The communication stub is a load module that uses the z/OS Web Enablement toolkit, which requires POSIX(ON) to be specified.

About this task

Restriction: Depending on the type of programs used, some restrictions apply to accessing z/OS Connect EE for API calls. Consider the following restrictions before you configure your API requester communication stub:

- Use of the RTEREUS (COBOL only) option has multiple restrictions, and should not be used in this context. To learn more, see [RTEREUS \(COBOL only\)](#).
 - Pre-loading the Message Processing Program (MPP) can make the COBOL program work with RTEREUS(ON) as a last resort. While unsupported, this technique might offer you a temporary work around to use RTEREUS(ON).
- z/OS Connect EE requires the z/OS Language Environment (LE) option POSIX(ON) for signal handling. There can be only one LE enclave, the first enclave, with POSIX(ON) per TCB. To avoid invalid configuration, specify POSIX(ON) at the application level by using CEEUOPT. If there are any nested enclaves, the main program in the nested enclave must use CEEUOPT to set POSIX(OFF).
- If the first or main program for a child enclave is COBOL, CEEUOPT cannot be used to set POSIX(OFF). Child enclaves for these programs can inherit z/OS Language Environment options only from their parent enclave, consequentially resulting in invalid configurations due to inheriting POSIX(ON) from the parent. Therefore, if the use of nested enclaves is needed for your program, the main program of the child enclave cannot be COBOL. To learn more about this restriction, see [Child enclave has a COBOL main program](#).

The load module for the communication stub must be added to the IMS STEPLIB concatenation. The z/OS Connect EE server name and port number must be specified for the communication stub. Use the CEEOPTS DD (data definition) statement for z/OS Language Environment to specify these options.

Procedure

1. Include the load module BAQCSTUB from the hlq.SBAQLIB data set into the IMS STEPLIB concatenation.
2. Specify connection options for the communication stub for IMS.
 - a) In your message processing region (MPR) job, specify the URI and port number for the z/OS Connect EE server in the CEEOPTS DD statement.

```
CEEOPTS DD *  
ENVAR("BAQURI=MYSERVER.MY.COM", "BAQPORT=10053")
```

- b) You can also specify the environment variable in a member, and then specify the member in the MPR job. In the following example, MPRLEOPT is the PROCLIB member that contains the environment variable:

```
CEEOPTS DD DSN=IMSVS.IMA1.&SYS2.PROCLIB(MPRLEOPT),DISP=SHR
```

The MPRLEOPT member contains the following information:

```
ENVAR("BAQURI=MYSERVER.MY.COM", "BAQPORT=10053")
```

Tip: z/OS provides functions to optimize performance for HTTP connections. If z/OS Connect EE and the z/OS subsystem are on the same LPAR, the fast local socket is automatically available. If z/OS Connect EE and the z/OS subsystem are on different LPARs, you must enable the following functions:

- HiperSocket, a zSeries hardware feature that provides high-performance internal communications between LPARs within the same central processor complex (CPC). For more information, see [HiperSockets concepts and connectivity](#) in the z/OS documentation.
- Shared Memory Communication (SMC), which requires z/OS V2.2 or later on z13 in the same CPC. For more information, see [Shared Memory Communications](#) in the z/OS documentation.

The following connection options are supported by the communication stub for IMS.

BAQPASSWORD

Specifies the password, in clear text, for the specified BAQUSERNAME to be authenticated with the z/OS Connect EE server. The user name and password are used for basic authentication, when the SSL **clientAuthentication** attribute is set to **false** but the service **requireSecure** and **requireAuth** attributes are set to **true**.

The value cannot exceed 255 characters.

BAQPORT

Specifies the port number for the z/OS Connect EE server.

BAQTIMEOUT

An optional 4-byte integer to set a timeout value in seconds for waiting for an API response. Valid range is 1 - 2,678,400 seconds. The default timeout value is 10 seconds.

Important: If CEEOPTS is specified for a message processing region, all outbound applications that run in the region use the same timeout value.

BAQURI

Specifies either an IPv4 or IPv6 address, or a hostname for the z/OS Connect EE server to connect to. To configure secure connections with IMS, specify the server without the HTTPS prefix (URI scheme). You can use basic authentication or client certificate to authenticate your IMS applications.

The value cannot exceed 255 characters.

BAQUSERNAME

Specifies the user name for connections if basic authentication is used. BAQPASSWORD must be specified.

The value cannot exceed 255 characters.

BAQVERBOSE

An optional value to turn on verbose messages to assist debugging of runtime and configuration issues.

Trace messages are directed to the standard output for the application environment. By default, verbose messages are turned off.



CAUTION: If you use the settings ON or ALL, the trace output might contain sensitive information.

Available levels and their usage are:

OFF

Turns off verbose trace messages (the default). No trace messages are produced.

ON

Turns on verbose trace messages. Most messages are written, including messages for entry, exit, errors, and auditing. To investigate any issues in the API requester stub code for IMS and batch, enable this level of trace.

ERROR

If an error occurs within the stub code, whether an internal error or by bad data being passed, messages are written to the output destination.

AUDIT

Records the entry and exit to stub and third-party libraries such as the HWT. This level is useful to track the timing of events such as entry and exit to the stub code, and timing of request and response through the HWT.

ALL

All diagnostic messages are written, including previous levels and intensive trace. This can include string conversions, setters, lookup tables etc. This level may cause a performance impact so it is not advisable unless there is an issue in one of the areas that writes at this level.

3. Compile and link your IMS application with CEEUOPT to set POSIX(ON). The following example demonstrates how to assemble a CEEUOPT CSECT to set POSIX(ON), and link the CEEUOPT CSECT to the load module of the IMS application being run.

```
//APIPRGM JOB ...
/*
/* STEP 1: ASSEMBLE CEEUOPT CSECT
/*
//CEEUOPT EXEC PGM=ASMA90,COND=(4,LT),REGION=0M,
// PARM='OBJECT'
//SYSLIB DD DSN=SYS1.MACLIB,DISP=SHR
// DD DSN=CEE.SCEEMAC,DISP=SHR
//SYSIN DD *
CEEUOPT CSECT
CEEUOPT AMODE ANY
CEEUOPT RMODE ANY
CEELOPT POSIX=(ON)
END
/*
//SYSLIN DD DSN=<HLQ>.ASM.OBJECT(CEEUOPT),DISP=SHR
//SYSPRINT DD DSN=<HLQ>.ASM.LISTING(CEEUOPT),DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
/*
/* STEP 2: COMPILE AND LINK COBOL PROGRAM
/*
//APIPRGM EXEC IGYWCL,LNGPRFX='IGYV5R20',
// LIBPRFX='SYS1',
// PARM.COBOL='TEST SOURCE',
// PARM.LKED='LIST MAP XREF'
//COBOL.SYSIN DD DSN=<HLQ>.COBOL.SOURCE(APIPRGM),DISP=SHR
//COBOL.SYSLIB DD DSN=<HLQ>.COBOL.COPYLIB,DISP=SHR
//COBOL.SYSLIN DD DSN=<HLQ>.COBOL.OBJECT(APIPRGM),DISP=SHR
//COBOL.SYSDEBUG DD DSN=<HLQ>.COBOL.SYSDEBUG(APIPRGM),DISP=SHR
//COBOL.SYSPRINT DD DSN=<HLQ>.COBOL.LISTING(APIPRGM),DISP=SHR
//LKED.OBJECT DD DSN=<HLQ>.ASM.OBJECT,DISP=SHR
// DD DSN=<HLQ>.COBOL.OBJECT,DISP=SHR
//LKED.RESLIB DD DSN=IMS.SDFSRESL,DISP=SHR
//LKED.SYSLIN DD *
INCLUDE OBJECT(CEEUOPT)
INCLUDE OBJECT(APIPRGM)
ENTRY APIPRGM
NAME APIPRGM(R)
/*
//LKED.SYSLMOD DD DSN=IMS.PGMLIB(APIPRGM),DISP=SHR
```

Important:

- Ensure that POSIX is set to ON to allow Language Environment to access the UNIX System Services environment for the required HTTP-enabling services. A POSIX(ON) environment also requires the user ID associated with the address space that uses these HTTP-enabling services to have an OMVS segment defined and associated with it.
- Persistent connections are automatically enabled between the z/OS Connect EE server and the communication stub in the IMS region, which requires z/OS Connect EE V3.0.10 or later.

Configuring other z/OS applications to access z/OS Connect EE for API calls

Before your z/OS applications that are not running on CICS or IMS can make RESTful API calls through z/OS Connect EE, you must configure the communication stub.

About this task

Restriction: Depending on the type of programs used, some restrictions apply to accessing z/OS Connect EE for API calls. Consider the following restrictions before you configure your API requester communication stub:

- z/OS Connect EE requires the z/OS Language Environment (LE) option POSIX(ON) for signal handling. There can be only one LE enclave, the first enclave, with POSIX(ON) per TCB. To avoid invalid configurations, specify POSIX(ON) at the application level by using CEEUOPT. If there are any nested enclaves, the main program in the nested enclave must use CEEUOPT to set POSIX(OFF).
- If the first or main program for a child enclave is COBOL, CEEUOPT cannot be used to set POSIX(OFF). Child enclaves for these programs can inherit z/OS LE options only from their parent enclave, consequentially resulting in invalid configurations due to inheriting POSIX(ON) from the parent. Therefore, if you need to use nested enclaves for your program, the main program of the child enclave cannot be COBOL. To learn more about this restriction, see [Child enclave has a COBOL main program](#).

The z/OS Connect EE communication stub is a load module to configure in your z/OS system to handle the HTTP connection and communication with the z/OS Connect EE server.

The load module for the communication stub must be added to the STEPLIB concatenation. In addition, the z/OS Connect EE server name and port number must be specified for the communication stub. Use the CEEOPTS DD statement for the Language Environment (LE).

Tip: z/OS provides functions to optimize performance for HTTP connections. If z/OS Connect EE and the z/OS subsystem are on the same LPAR, the fast local socket is automatically available. If z/OS Connect EE and the z/OS subsystem are on different LPARs, you must enable the following functions:

- HiperSocket, a zSeries hardware feature that provides high-performance internal communications between LPARs within the same central processor complex (CPC). For more information, see [HiperSockets concepts and connectivity](#) in the z/OS documentation.
- Shared Memory Communication (SMC), which requires z/OS V2.2 or later on z13 in the same CPC. For more information, see [Shared Memory Communications](#) in the z/OS documentation.

Procedure

1. Include the load module BAQCSTUB from the hlq.SBAQLIB data set into the STEPLIB concatenation. Use STEPLIB rather than LINKLST for library concatenation to avoid runtime issues after applying maintenance to the communication stub.
2. In your batch job, specify the URI and port number for the z/OS Connect EE server in the CEEOPTS DD statement.
3. To configure secure connections, you can use basic authentication or client certificate to authenticate your z/OS applications. For more information, see [“How to configure basic authentication from IMS or a z/OS application” on page 410](#) or [“API requester client certificate authentication to z/OS Connect EE” on page 412](#).

```
CEEOPTS DD *  
  POSIX(ON),  
  ENVAR("BAQURI=MYSERVER.MY.COM", "BAQPORT=10053")
```

Note:

- Ensure that POSIX is set to ON to allow Language Environment to access the UNIX System Services environment for the required HTTP-enabling services. A POSIX(ON) environment also requires the user ID associated with the address space that uses these HTTP-enabling services to have an OMVS segment defined and associated with it.

- Persistent connections are automatically enabled between the z/OS Connect EE server and the communication stub in the non-CICS z/OS applications, which requires z/OS Connect EE V3.0.10 or later.

The following connection options are supported by the communication stub.

BAQPASSWORD

Specifies the password, in clear text, for the specified BAQUSERNAME to be authenticated with the z/OS Connect EE server. The user name and password are used for basic authentication, when the SSL **clientAuthentication** attribute is set to **false** but the service **requireSecure** and **requireAuth** attributes are set to **true**.

The value cannot exceed 255 characters.

BAQPORT

Specifies the port number for the z/OS Connect EE server.

BAQTIMEOUT

An optional 4-byte integer to set a timeout value in seconds for waiting for an API response. Valid range is 1 - 2,678,400 seconds. The default timeout value is 10 seconds.

Important: If CEEOPTS is specified for a message processing region, all outbound applications that run in the region use the same timeout value.

BAQURI

Specifies either an IPv4 or IPV6 address, or a hostname for the z/OS Connect EE server to connect to. To configure secure connections with IMS, specify the server without the HTTPS prefix (URI scheme). You can use basic authentication or client certificate to authenticate your IMS applications.

The value cannot exceed 255 characters.

BAQUSERNAME

Specifies the user name for connections if basic authentication is used. BAQPASSWORD must be specified.

The value cannot exceed 255 characters.

BAQVERBOSE

An optional value to turn on verbose messages to assist debugging of runtime and configuration issues.

Trace messages are directed to the standard output for the application environment. By default, verbose messages are turned off.



CAUTION: If you use the settings ON or ALL, the trace output might contain sensitive information.

Available levels and their usage are:

OFF

Turns off verbose trace messages (the default). No trace messages are produced.

ON

Turns on verbose trace messages. Most messages are written, including messages for entry, exit, errors, and auditing. To investigate any issues in the API requester stub code for IMS and batch, enable this level of trace.

ERROR

If an error occurs within the stub code, whether an internal error or by bad data being passed, messages are written to the output destination.

AUDIT

Records the entry and exit to stub and third-party libraries such as the HWT. This level is useful to track the timing of events such as entry and exit to the stub code, and timing of request and response through the HWT.

ALL

All diagnostic messages are written, including previous levels and intensive trace. This can include string conversions, setters, lookup tables etc. This level may cause a performance impact so it is not advisable unless there is an issue in one of the areas that writes at this level.

Configuring an HA environment for RESTful API calls from z/OS applications

You can configure a highly available z/OS Connect EE environment that handles RESTful API calls from z/OS applications through API requesters.

Note: For more information about the concepts of high availability and the options you can choose, see Chapter 8, “High availability,” on page 317.

The following diagram demonstrates the following topological characteristics:

- Two z/OS Connect EE servers run in each LPAR.
- Sysplex Distributor provides high availability and workload balancing.
- Port sharing is configured for each LPAR.
- Sysplex Distributor is configured with the OPTLOCAL feature to take advantage of co-location optimization. For more information about configuring sysplex distributor to prefer local connections, see “[Sysplex distributor](#)” on page 317.

The following configuration steps demonstrate how to configure two servers in one LPAR for HA.

Prerequisites

1. Create the first server, and configure the server properly to support API requesters, as described in [“Configuring z/OS Connect EE to support API requesters” on page 304](#).
2. On the first server, go through the steps as outlined in the high-level workflow in [“z/OS applications to call REST APIs” on page 9](#) to configure, develop, deploy, and test all the required artifacts to call a REST API from your z/OS application. Ensure everything works properly on the first server.

General steps to configure for HA

The following steps demonstrate setting up HA with two z/OS Connect EE servers in one LPAR, sharing a TCP/IP port.

1. Set up HA for the first server:

- a. Set up TCP/IP port sharing.

Note: Choose an unreserved port number. Use a different port number from the number that is used as the HTTP listener for incoming HTTP requests for services and APIs to access the backend z/OS assets.

For more information about port sharing, see [“TCP/IP port sharing” on page 318](#).

- b. Create a shared configuration file. This step involves moving the `server.xml` to a location that can be shared with the second server and creating another copy of the `server.xml` for the first server that points to the shared location.

- i) Stop the server.

- ii) Create directories under `/var/zosconnect` to store the shared configuration file. For example, run the following command from the `/var/zosconnect` directory:

```
mkdir -p shared/config
```

- iii) Move the `server.xml` to a shared configuration directory with a new name. For example, rename it as `haserver.xml`.

iv) Create a `server.xml` for the server with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="z/OS Connect EE HA server1">
    <include location="${shared.config.dir}/haserver.xml"/>
</server>
```

This configuration file includes the shared `haserver.xml` by using the Liberty property `$
{shared.config.dir}` to reference the directory that contains the shared configuration files. This property points to the directory `WLP_USER_DIR/shared/config`. The include statement points to the renamed file, `haserver.xml`, in the shared location.

- c. Restart the server.
2. Add a second server.
3. Replace the default `server.xml` file on the second server with the `server.xml` file from the first server, so both the first server and the second server point to the shared location for the server configuration file.
4. Start your second server.

The two server instances now share the same resources, and the HA environment for API requesters is configured.

Chapter 8. High availability

With a high availability solution, you can minimize the impact on daily operations of the failure of one or more components within the overall system. The key concepts of a high availability (HA) environment and how they can be implemented with z/OS Connect EE are discussed.

This information is aimed at architects and z/OS system programmers who need to implement a z/OS Connect EE HA environment. Implementation requires z/OS system programming skills, such as the use of TCP/IP workload balancing software and creation of zFS file systems.

You can use various IBM or third-party products with z/OS Connect EE to build such a system. You have a number of options available so that you can choose a solution that suits the needs of your organization. The solution might be just two z/OS Connect EE servers that run in a single LPAR. Or you can design a more complex parallel sysplex solution with many z/OS Connect EE servers that are spread across multiple LPARs.

It is also necessary to plan for variations in workloads, especially increased workloads, when it might be necessary to increase the capacity of the system. The system must respond in a predictable way to workload variation to meet SLAs. For z/OS Connect EE, scalability can be achieved by manually increasing the number of servers.

Connectivity for high availability

A z/OS Connect EE HA environment can take advantage of TCP/IP load balancing technologies.

Both the requests to invoke APIs and services in z/OS Connect EE servers, or RESTful API requests from z/OS applications, are made by using the HTTP protocol, which means that an HA environment can use known TCP/IP load balancing technologies to distribute connections from clients across multiple servers.

- Sysplex Distributor can be used to distribute requests across multiple z/OS Connect EE servers in a Parallel Sysplex, with TCP/IP port sharing within each LPAR.
- TCP/IP port sharing can be used to load balance requests across multiple z/OS Connect EE servers in the same z/OS LPAR.
- WLM classification can be used to classify HTTP traffic in a z/OS Connect EE server so that it can be managed by z/OS workload management (WLM) along with other work in the system.

By configuring the `httpEndpoint` in the server configuration file of multiple z/OS Connect EE servers to listen on the same shared TCP/IP port, clients can send requests to a single hostname and port and allow the TCP/IP load balancing technologies to distribute the request to one of the available servers. z/OS Connect EE APIs, services, or API requesters have no persistent state and do not use HTTP sessions, so no affinities exist between client applications and z/OS Connect EE servers, simplifying both workload distribution and the retry of failed requests.

Sysplex distributor

Sysplex Distributor is the strategic IBM solution for connection workload balancing across an IBM Parallel Sysplex.

Sysplex Distributor is a component of IBM z/OS Communications Server and provides TCP/IP load balancing across multiple LPARs. It can be combined with TCP/IP port sharing. Sysplex distributor is a combination of the high availability features of distributed DVIPA and the workload optimization capabilities of WLM. The TCP/IP stacks can be configured to request workload information from WLM, enabling the distributing stack to forward those connections that are based on the workload of each of the target stacks. The WLM workload information is based on a comparison of available general CPU capacity for each target system.

z/OS Connect EE is a Java-based product, so it can use System z® Integrated Information Processor (zIIP) capacity. You can configure the TCP/IP stacks to consider the available zIIP CPU capacity. You can also configure WLM-based forwarding based on server-specific workload.

High availability with preferred local connections

Where z/OS Connect EE makes an *onward* TCP/IP-based connection to another system, the z/OS Sysplex distributor optimization capability can be applied to ensure that connections remain local to an LPAR whenever possible. This capability is enabled by use of the OPTLOCAL keyword on the VIPADISTRIBUTE statement in the associated VIPADYNAMIC block of the z/OS TCP/IP profile. For more information, see [Sysplex distributor optimization with the OPTLOCAL keyword in the z/OS Communications Server: IP Configuration Guide](#).

This capability is useful in the following situations:

- Preferring z/OS Connect EE to connect to a local CICS region (over IPIC) by using the CICS service provider, but failing-over to a CICS region on another LPAR when no local peer region is available. Each peer CICS region uses a common listener port that is defined to z/OS Communications Server as shared, and specifying the OPTLOCAL keyword.
- Preferring z/OS Connect EE to connect to a local IMS Connect instance by using the IMS mobile service provider, but failing-over to an IMS Connect instance on another LPAR when no local instance is available. Each peer IMS Connect instance uses a common listener port that is defined to z/OS Communications Server as shared, and specifying the OPTLOCAL keyword.
- Preferring z/OS Connect EE to connect to a local Db2 server that provides the Db2 REST service by using the REST client service provider, but failing-over to a Db2 server on another LPAR when no local Db2 server that provides the Db2 REST service is available. Each Db2 server that provides the Db2 REST service uses a common listener port that is defined to z/OS Communications Server as shared, and specifying the OPTLOCAL keyword.
- Preferring to connect to a z/OS Connect EE server in the local LPAR for an application by using the API requester, but failing-over to a z/OS Connect EE server in another LPAR when no local z/OS Connect EE server that is configured with the same API requester is available. Each peer z/OS Connect EE server users a common listener port that is defined to z/OS Communications Server as shared, specifying the OPTLOCAL keyword.

In each case, z/OS Connect EE relies upon facilities of the z/OS Sysplex distributor. Specifically, the distributed VIPA keyword, OPTLOCAL, expresses the preference to keep connections within the local LPAR when possible.

Related information

[Configuring distributed DVIPAs - sysplex distributor](#)

[Sysplex distributor optimization with the OPTLOCAL keyword](#)

TCP/IP port sharing

The TCP/IP port sharing component of the z/OS TCP/IP Communications subsystem allows multiple listeners to listen on the same combination of port and IP address.

Port sharing can be configured to distribute HTTP requests across multiple z/OS Connect EE servers in a single LPAR.

Note: z/OS Connect EE servers that use a shared port must have the same APIs and services available.

Shared ports are defined by the PORT statement in the TCP/IP profile. A shared port definition reserves a specific port number to be shared by specific z/OS address space job names. Two options of the PORT statement control port sharing:

SHAREPORT

Requests are distributed by a weighted round-robin distribution method based on the Servers' accept Efficiency Fractions (SEFs) of the listeners that share the port.

SHAREPORTWLM

Requests are distributed based on WLM server-specific recommendations, which are modified by the SEF values for each listener. Load balancing can be more efficient if you consider the status that is provided by dynamic feedback from the z/OS Workload Manager (WLM).

When you configure port sharing for z/OS Connect EE, define shared ports for any ports that receive HTTP or HTTPS requests to invoke APIs or services. For example, define the port values specified on the `httpEndpoint` element with `id="defaultHttpEndpoint"` in your server's configuration file. For multiple z/OS Connect EE server address spaces to share a port, their job names must start with the same characters. It is the common portion of the job name that must be specified on the PORT definition. For example, to uniquely identify two servers:

1. Use the BAQSTRT JCL started task procedure provided in `<hlq>.SBAQSAMP`.
2. Specify the `JOBNAME` parameters as `JOBNAME=ZCHA1` and `JOBNAME=ZCHA2`. The common portion of the job name in this instance is "ZCHA".

Note: Use the `JOBNAME` parameter rather than an identifier for started tasks. See [Starting a system task from a console](#) in the *z/OS MVS System Commands* documentation. If you choose to use identifiers, be aware that it is the common portion of the identifier value that is used as the job name in the shared port definition.

Configure a separate port to handle outgoing RESTful API requests from z/OS subsystems if the same z/OS Connect EE server is used to handle both outgoing RESTful API requests and incoming requests that access z/OS subsystems. Using separate ports allows for better security and audit control.

Related information

[Considerations for multiple servers sharing a TCP port](#)

[TCP/IP profile \(PROFILE.TCPIP\) and configuration statements](#)

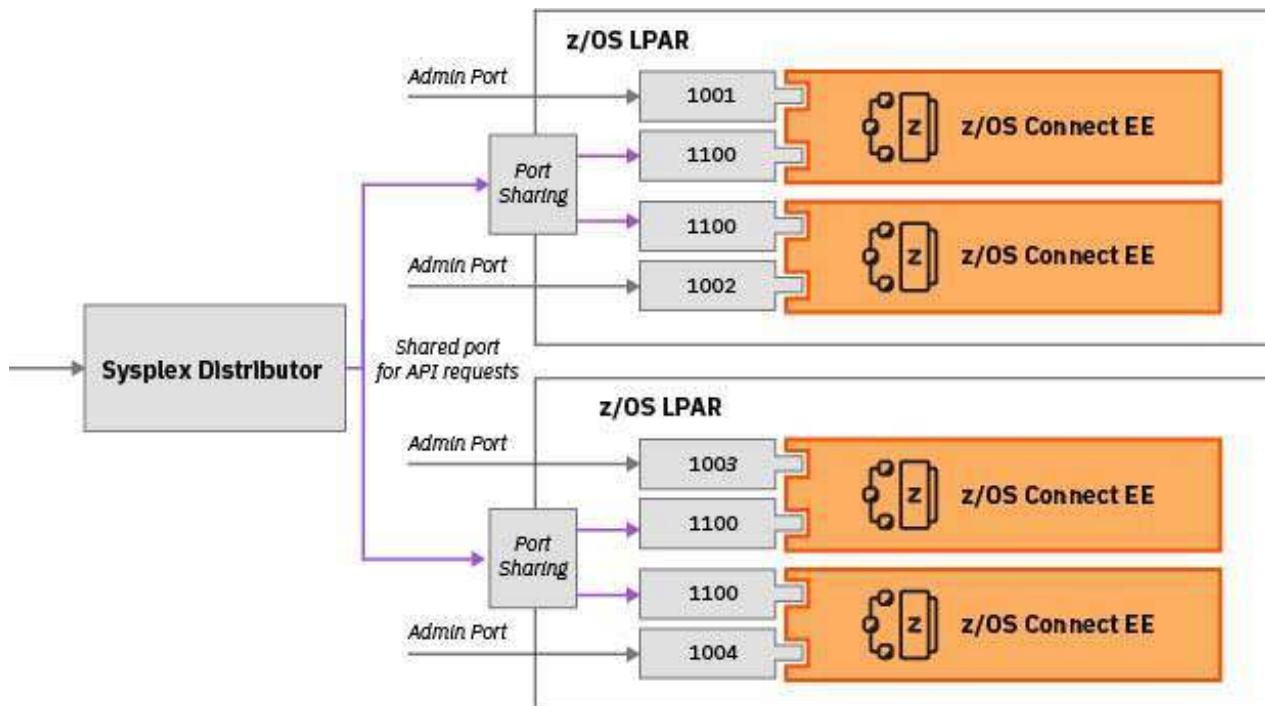
[PORT statement](#)

High availability of z/OS Connect EE servers

If you have multiple z/OS Connect EE servers configured, you can implement a high availability solution to provide scalability and backup in case of a server failure

HA can be implemented with just two z/OS Connect EE servers that run in a single LPAR. Or you can design a more complex parallel sysplex solution with many z/OS Connect EE servers that are spread across multiple LPARs.

The following diagram illustrates how z/OS Connect EE works within an HA environment across multiple LPARs.



High availability can be achieved with z/OS Connect EE by the workload distribution of HTTP and HTTPS API and service requests from clients across several servers with shared configuration. For more information, see “[Connectivity for high availability](#)” on page 317 and “[Sharing server configuration](#)” on page 325. To ensure that workload distribution can route requests to any configured z/OS Connect EE server, the servers must be maintained as clones of each other. That is, the servers must have the same configuration with the same z/OS Connect APIs, services, or API requesters deployed.

For a step-by-step example of how to implement high availability for inbound requests to two z/OS Connect EE servers that use port sharing and shared configuration data, see “[Configure HA for connections to z/OS Connect EE](#)” on page 170.

WLM classification

Classify HTTP traffic to improve workload distribution.

You can classify HTTP traffic in a z/OS Connect EE server so that it can be managed by z/OS workload management (WLM) along with other work in the system by enabling Liberty feature **zosWlm-1.0**. This identifies work by classification rules and pass that into z/OS WLM for mapping to service classes and reporting classes. Different URIs can have different priority and performance criteria. Classification of HTTP requests is based on host, port, method, and resource and is configured in the server configuration file.

A WLM enclave is associated with the thread that the request is dispatched on. It is also associated with a WLM Service Class. A WLM Service Class is assigned to the WLM enclave by WLM, based on rules that you define in the WLM configuration. The WLM Service Class indicates the WLM goals for each class of client work, for example, 95% complete in 1 second or less. The WLM Service Class also indicates the importance of the goals relative to other work on the system. WLM uses information that is provided by the Liberty server during classification to assign a WLM Service Class.

In a high availability environment, you can specify the same value on the WLM Native Enclave Manager Configuration element for all your servers. For example,

```
<zosWorkloadManager collectionName="<value>" />
```

This method allows the servers to use the same collection name in WLM, instead of the default collection name, which is the server name for each server.

Related information

[Enabling workload management for Liberty on z/OS](#)

High availability of Systems of Record (SoRs)

Configure high availability between your z/OS Connect EE servers and the Systems of Record (SoRs).

The following topic describes how to set up a high availability environment for a System of Record (SoR).

CICS IPIC high availability

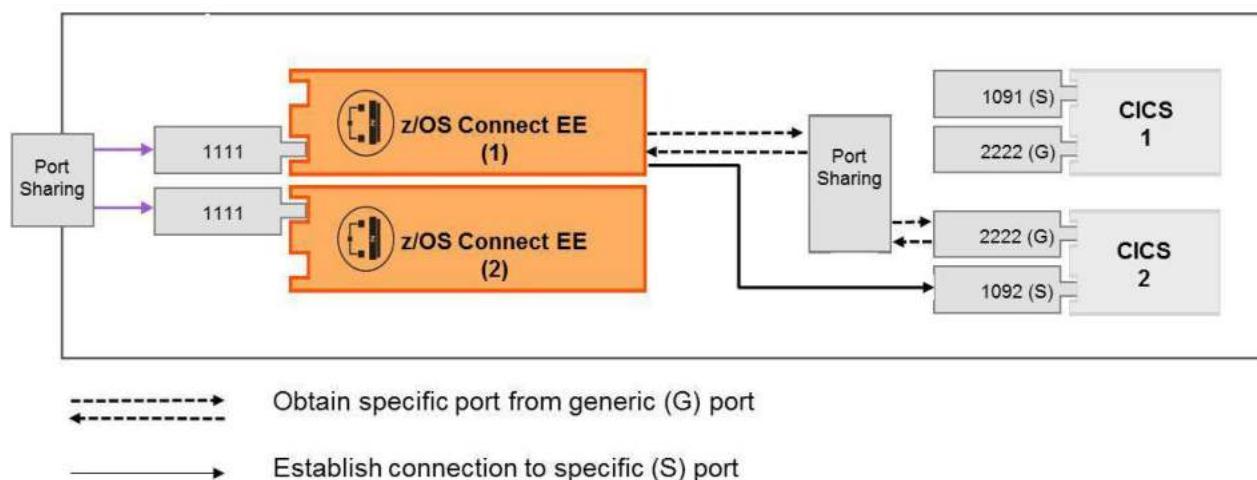
The IPIC high availability capability provides a single point of access from a z/OS Connect EE server to a cluster of CICS TS regions through an IP network.

This capability ensures resilience of access to the cluster as a whole, for both planned and unplanned outages of individual regions within the cluster. The single end point of the cluster is managed on z/OS by a connection balancing mechanism that spreads connectivity across the set of regions within a cluster. z/OS Communications Server offers two balancing mechanisms: TCP/IP port sharing and Sysplex Distributor. You can also configure Sysplex Distributor to prefer local connections. For more information, see “[Sysplex distributor](#)” on page 317.

Note: This capability provides high availability for IPIC connections between z/OS Connect EE servers and CICS TS regions. It applies only to the establishment of the initial connection to a CICS TS region or reestablishing a connection after a failure. It does not provide workload balancing of subsequent requests.

For a step-by-step example of how to implement high availability for connections to CICS using the CICS service provider by adding a second CICS region, see [“Configure HA for connections into CICS” on page 174](#).

How it works



The CICS regions listen on two end points that are defined by TCPIPSERVICE resources. The generic end point is shared by all regions in the cluster and a specific end point that is used exclusively by a specific region. The IPConn can be auto-installed, or pre-defined. When using security, ensure that the generic and specific TCPIP Services specify the same security credentials.

A z/OS Connect EE server connects to a CICS region in the cluster by using the generic end point. The connection request to the generic end point is intercepted by the connection-balancing mechanism, and routed to a generic TCPIPSERVICE that belongs to one of the CICS regions in the cluster. The selected CICS region then returns the IP address and port of its specific end point to the z/OS Connect EE server, and the connection is established to the specific end point.

Related tasks

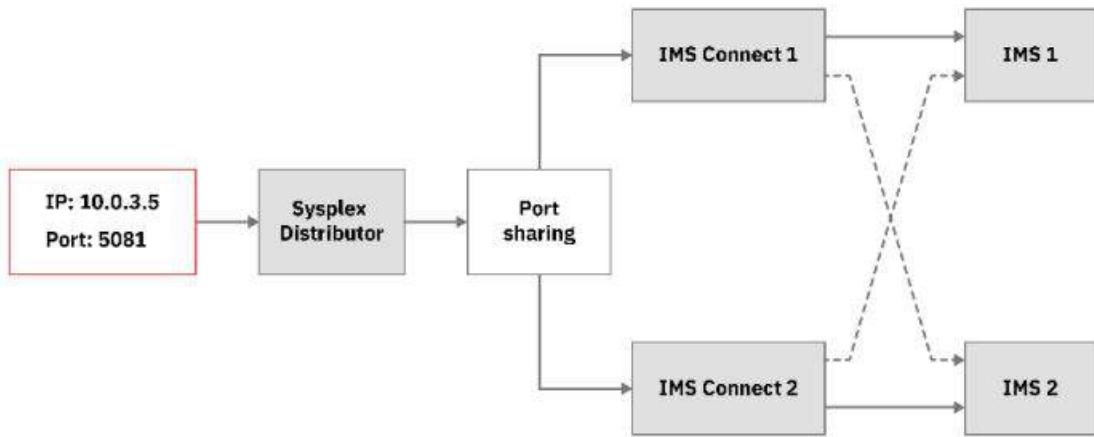
[“Configuring IPIC High Availability” on page 229](#)

CICS service provider IPIC connections can use port sharing or be configured to use sysplex distributor to connect to a cluster of CICS regions. The CICS regions must be configured with the same program, transaction, and security settings. The IPIC connections can be configured to use SSL.

IMS and IMS Connect high availability

When IMS assets serve as REST API endpoints through the IMS or IMS Database service provider, the first place to configure for IMS Connect high availability is in the TCP/IP network that sends and receives messages.

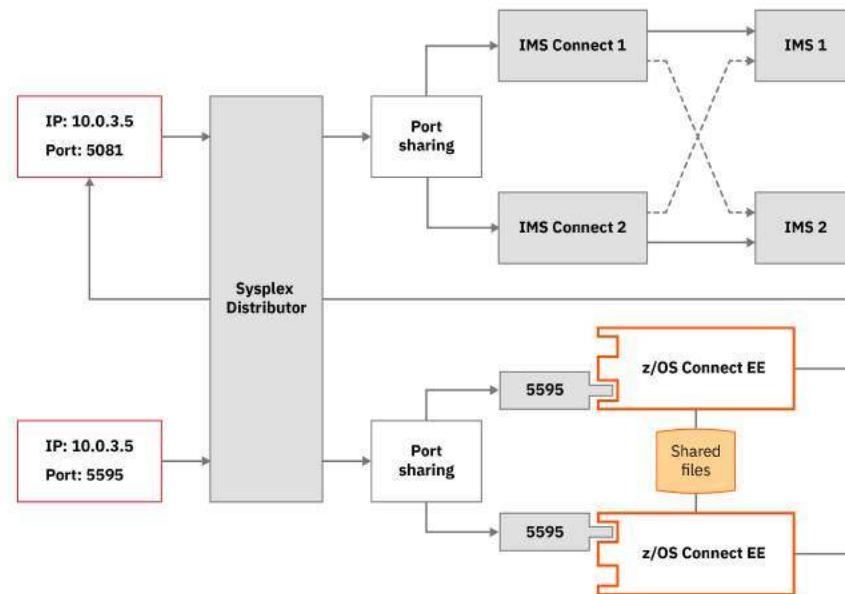
IMS Connect availability is typically achieved with static or dynamic virtual IP addresses (VIPA) or Sysplex Distributor for TCP/IP load balancing and failover. The following diagram shows a sample IMS Connect high availability configuration that might be already in place in your environment. The Sysplex Distributor routes incoming requests for an IP address of 10.0.3.5 to two IMS Connect instances that share the port of 5081. Each IMS Connect is configured to access one or more IMS systems.



You can also configure Sysplex Distributor to prefer local connections. For more information, see [“Sysplex distributor” on page 317](#).

Connections to IMS from the IMS service provider in z/OS Connect EE can be configured to go through a port on the Sysplex Distributor that is shared by multiple z/OS Connect EE servers for high availability.

The following diagram shows how connections between z/OS Connect EE servers and IMS regions are set up based on an existing IMS Connect high availability configuration. The two z/OS Connect EE servers are configured to share port 5595 for incoming API requests. The IMS service providers on these servers share the same IMS connection profile and IMS interaction profile. The connection to IMS is set to a host/IP address of 10.0.3.5 and a port of 5081.



The shared files include:

- Shared server configuration
- Shared archive files for services and APIs
- Shared IMS connection profile and IMS interaction profile

For steps to configure for shared resources, see [“Configure HA for connections to z/OS Connect EE” on page 170](#).

Related information

[IMS Integration and Connectivity Across the Enterprise \(IBM Redbooks publication\)](#)

Sysplex Distributor

VIPA

[IMS in the Parallel Sysplex Volume I \(IBM Redbooks publication\)](#)

[IMS in the Parallel Sysplex Volume II \(IBM Redbooks publication\)](#)

[IMS in the Parallel Sysplex Volume III \(IBM Redbooks publication\)](#)

IBM MQ high availability

A highly available IBM MQ and z/OS Connect EE environment can be achieved by using queue-sharing groups.

A queue-sharing group consists of two or more queue managers, which can access a common set of queues with messages that are stored in list structures in a coupling facility. Queues with messages that are stored in a coupling facility are called shared queues. A queue-sharing group provides high availability of individual messages because any queue manager in a queue-sharing group can send messages to or receive messages from a shared queue.

Queue managers outside a queue-sharing group store messages on VSAM linear data sets called page sets. Queues that have messages stored on page sets are called private queues and are only accessible by the queue manager that owns the page set.

Queue managers in a queue sharing group can have a combination of private and shared queues.

For more information about shared queues and queue sharing groups, see [Queue sharing groups](#) in the *IBM MQ* documentation.

Use z/OS Connect EE with IBM MQ queue-sharing groups

Two scenarios describe how to provide a highly available environment with z/OS Connect EE and IBM MQ queue-sharing groups.

In all cases, the scenarios provide resiliency against the failure of individual components. Your existing IBM MQ and z/OS Connect EE configurations determine which scenario is most applicable in your environment.

Scenario one: z/OS Connect EE server and queue managers on same LPAR

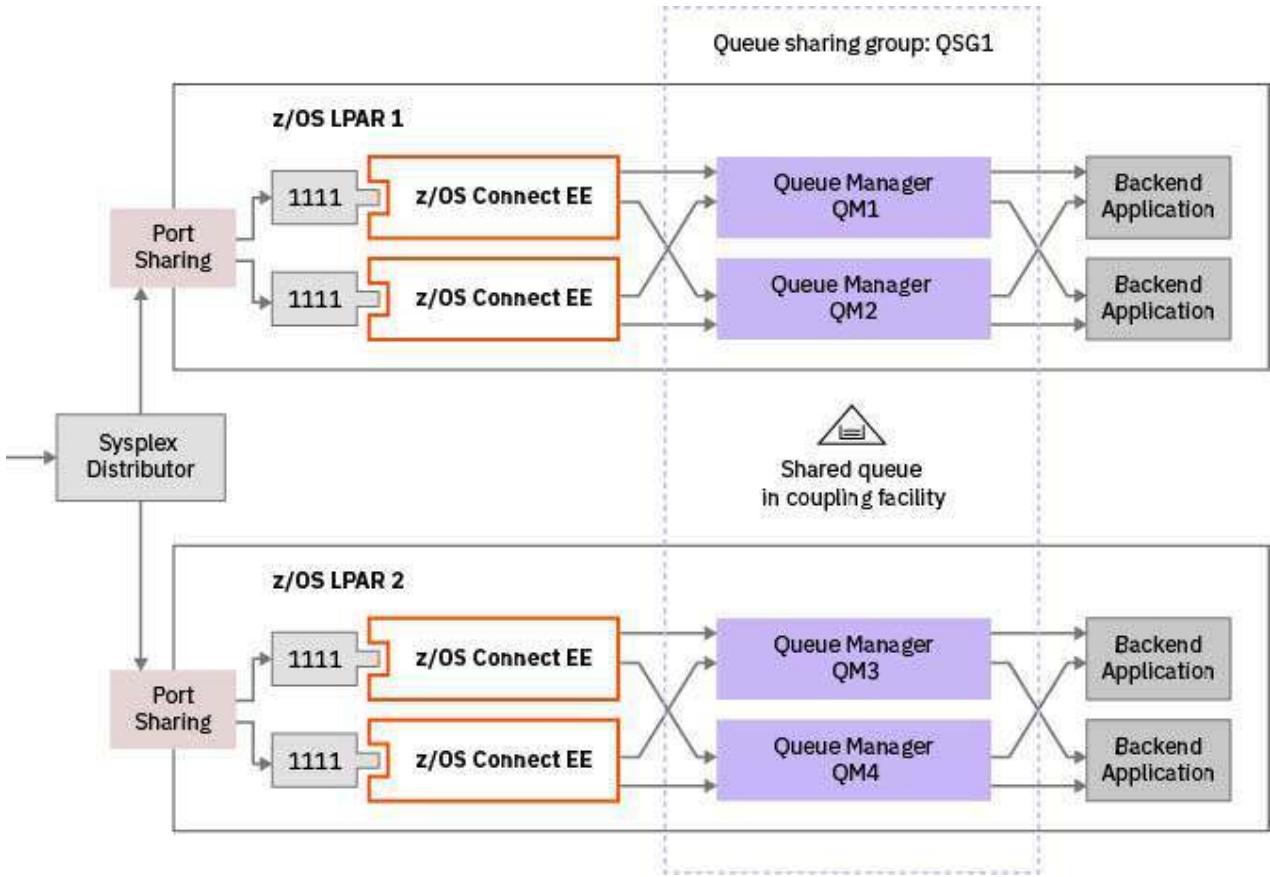


Figure 68. Scenario one: z/OS Connect EE servers and queue managers on the same LPAR

The diagram in Figure 68 on page 324, shows how Sysplex Distributor is used to distribute work across a pair of identically configured LPARs. Each LPAR is configured with a pair of z/OS Connect EE servers that listen on the same shared port. For more information about this configuration, see “[High availability of z/OS Connect EE servers](#)” on page 319.

The queue-sharing group is made up of four queue managers. To spread workload evenly, there are two queue managers on each LPAR.

Several instances of the backend application run across both LPARs. They put and get messages to shared queues that reside in a coupling facility. Each instance of the backend application can connect to any local queue manager and access the same messages because they are all held on shared queues.

The configuration of each z/OS Connect EE server is the same and consists of an IBM MQ service. The IBM MQ connection factory that is used by the service, connects in bindings mode to a local queue manager that is part of a queue-sharing group. The connection factory definition refers to the name of the queue-sharing group, such as QSG1, not to an individual queue manager.

The following connection factory definition is typical for this scenario:

```
<jmsConnectionFactory jndiName="jms/cf1" connectionManagerRef="ConMgr1">
  <properties.wmqJms transportType="BINDINGS" queueManager="QSG1"/>
</jmsConnectionFactory>
```

The IBM MQ service provider uses global transactions to connect to IBM MQ, so the queue managers in the queue-sharing group need to be enabled for group units of recovery. For more information, see [Enabling GROUP units of recovery](#) in the *IBM MQ* documentation.

Scenario two: z/OS Connect EE server and queue managers on separate LPARs

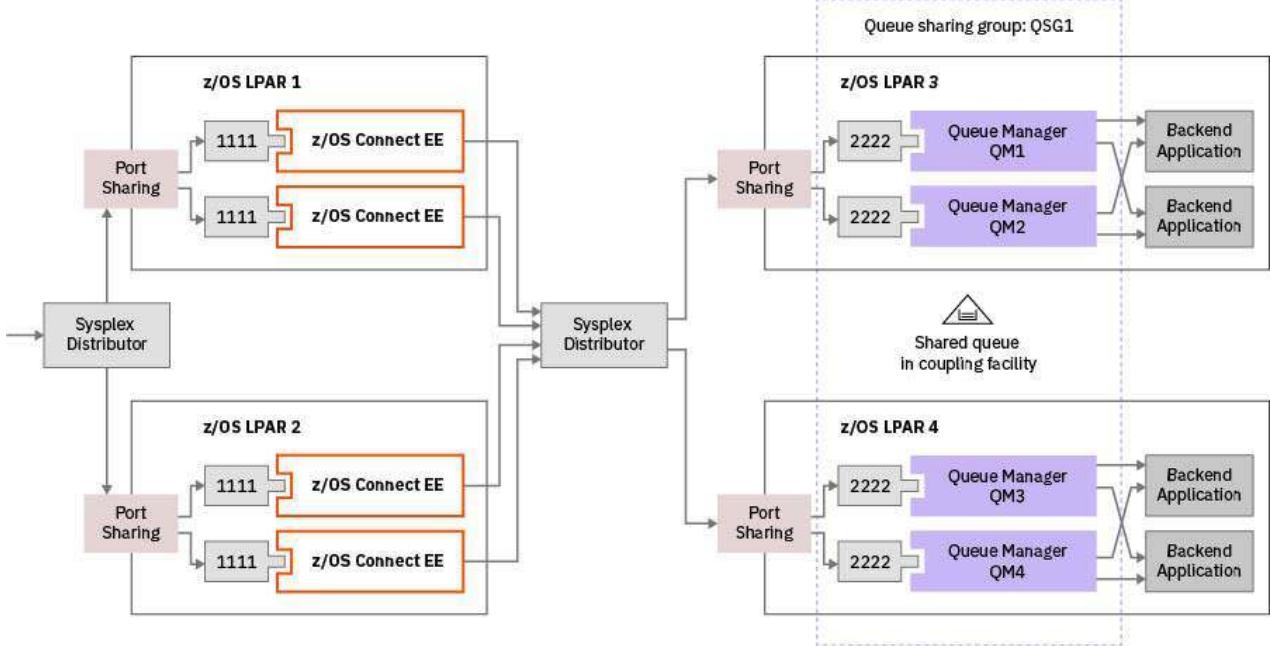


Figure 69. Scenario two: z/OS Connect EE servers and queue managers on separate LPARs

The diagram in Figure 69 on page 325, shows the z/OS Connect EE servers on different LPARs to the queue managers. The z/OS Connect EE servers must connect to the queue managers over the network by using client mode connections. Sysplex Distributor and shared ports are used to spread connections across the queue managers in the queue-sharing group.

As in Scenario one, the configuration of each z/OS Connect EE server is the same and consists of an IBM MQ service. The service uses a connection factory definition which refers to the name of the queue-sharing group, such as QSG1, not to an individual queue manager.

The following connection factory definition is typical for this scenario:

```
<jmsConnectionFactory jndiName="jms/cf1" connectionManagerRef="ConMgr1">
  <properties.wmqJms transportType="CLIENT" queueManager="QSG1"/>
  <properties.wmqJms hostName="sysplex.distributor.address"/>
  <properties.wmqJms port="shared.port"/>
</jmsConnectionFactory>
```

Typically, TLS is also enabled by configuring the `ssl*` properties. For more information, see [WebSphere MQ JMS connection factory](#) configuration element in the *WebSphere Application Server Liberty* documentation.

The IBM MQ service provider uses global transactions to connect to IBM MQ, so the queue managers in the queue-sharing group need to be enabled for group units of recovery. For more information, see [Enabling GROUP units of recovery](#) in the *IBM MQ* documentation.

Sharing server configuration

Use a shared configuration to rapidly deploy and maintain multiple z/OS Connect EE servers.

You can have many z/OS Connect EE servers configured, for example, to manage workload or provide high availability.

To avoid the need to manually edit multiple z/OS Connect EE server configuration files with the same content each time an update is required, you can share a configuration file between the z/OS Connect EE servers. Sharing a configuration file simplifies administration and maintenance because only a single file needs to be updated and ensures that all servers are using the same configuration. To use a shared configuration file between servers, you must consider the following aspects:

- The mechanisms that allow server configuration to be split into multiple files.
- Which configuration elements can be shared and which elements must be server-specific.
- Where to store the shared configuration file.

The following topics describe how to set up a shared configuration.

Splitting server configuration information

Configuration elements that are common to all z/OS Connect EE servers can be included from a shared configuration file.

Each server must have a valid Liberty server configuration file that is called `server.xml` in its server configuration directory `${server.config.dir}` . The following mechanisms can be used to split a server configuration file into server-specific content and content that is common to multiple servers:

- Use include elements in server configuration files to include additional server configuration files, in the format: `<include location=" pathname/filename " />`. For example, each server's configuration file might contain only those entries that are specific to that server and specify an element to include another configuration file whose content is common to all servers.

When you use include elements, be aware of the Liberty profile configuration element merging rules. See [Liberty:Configuration element merging rules](#) in the *WebSphere Application Server for z/OS* documentation. Specify each element's attribute in only one of the configuration files to ensure that only one value takes effect. For more information, see [Using include elements in configuration files](#) in the *WebSphere Application Server for z/OS* documentation.

- Use properties in server configuration files to enable an otherwise server-specific element to be contained in the common server configuration file. Properties for each server are defined in a file that is called `bootstrap.properties` in the server configuration directory `${server.config.dir}` . The properties are defined in the format of name=value pairs that are written one line per property. For example, `propertyName=propertyValue`. Server configuration files can reference these properties by using the syntax: `${propertyName}` .

For example, for two servers to specify unique three part names for WOLA, the configuration file contains the following statements.

- server1 has a `bootstrap.properties` file with the following content:
 - `wola_dgn=ZC1DGN`
 - `wola_ndn=ZC1NDN`
 - `wola_svn=ZC1SVN`
- server2 has a `bootstrap.properties` file with the following content:
 - `wola_dgn=ZC2DGN`
 - `wola_ndn=ZC2NDN`
 - `wola_svn=ZC2SVN`

A shared configuration file might then specify the element by using these properties:

```
<zosLocalAdapters wolaGroup=" ${wola_dgn} " wolaName2=" ${wola_ndn} " wolaName3=" ${wola_svn} " />
```

Note: If you update the `bootstrap.properties` file, you must restart the server for the changes to take effect. For more information, see [Using variables in configuration files](#) in the *WebSphere Application Server for z/OS* documentation.

Server-specific configuration elements

Whether configuration elements are common or server specific depends on your environment, APIs, and service providers.

When you use the WOLA service provider, the three part name is derived from the `wolaGroup`, `wolaName2`, and `wolaName3` attribute values on the local adapter's configuration element `zosLocalAdapters`. This three part name must be unique for each server in the same z/OS LPAR.

Location of shared configuration files

A shared configuration file must be stored in a location that can be read by all servers.

It is good practice to use the Liberty profile's shared directory structure. The `shared.config.dir` property specifies the location `<WLP_USER_DIR>/shared/config`. You can use this property name within your server configuration file. By default the value of `WLP_USER_DIR` is `/var/zosconnect`, so this location might not be suitable if you want to use shared zFS in a Parallel Sysplex, because the `/var` mount point cannot be shared. However, you can customize the value of the `WLP_USER_DIR` environment variable before you install z/OS Connect EE. For more information, see [“Installing z/OS Connect EE” on page 196](#).

In a single LPAR, a UNIX System Services directory that is not in the `${server.config.dir}` directory structure can be used so that it is not nested in the directory structure of a single server.

In a Parallel Sysplex, you can mount a shared zFS to share a single configuration file between multiple servers. For more information about using shared zFS in a Parallel Sysplex, see [z/OS File System overview](#). You can further improve the resilience of these file systems by using RAID arrays and DASD mirroring technologies. Alternatively, you can share configuration files only among the servers in each LPAR and maintain separate physical copies of the file on each LPAR. These files must be kept in sync manually.

Related information

[Customizing the Liberty environment](#)

[Liberty:Directory locations and properties](#)

Maintenance of API, service, and API requester deployments

Workload distribution requires z/OS Connect EE servers to be kept in sync.

To enable workload distribution across a group of servers, the servers must have the same APIs, services, and API requesters deployed. Any changes to those APIs, services, or API requesters must be reflected across all the servers that participate in the shared workload.

APIs are deployed to a location directory that is specified by the **location** attribute of the `zosconnect_zosConnectAPIs` element. By default, this attribute contains the server-specific directory `${server.config.dir}/resources/zosconnect/apis`. Storing the API archive files in a single directory that is shared by all servers ensures that the same versions of all APIs are available in all servers.

Likewise, API requesters are deployed to a directory that is specified in the **location** attribute of the `zosconnect_apirequesters` element. By default, this attribute contains the server-specific directory `${server.config.dir}/resources/zosconnect/apiRequesters`. Storing the API requester archive files in a single directory that is shared by all servers ensures that the same versions of all API requesters are available in all servers.

Files that are associated with services can also be stored in UNIX System Services directories, depending on the service provider they use. For example, the WOLA service provider uses the supplied DataXForm to transform JSON payload to bytes. This service requires WSbind files and JSON schema files whose locations are specified to the `zosconnect_zosConnectDataXform` server configuration element. Choosing to store all the service files in directories that are shared by all the servers ensures that the same versions of all services are available in all servers.

Deciding where to store the API, service, and API requester files

The API location directory, any directories that contain service files, and the API requesters must be stored in a location that all servers have read and write access to.

It is good practice to use the Liberty profile's shared directory structure. The `shared.resource.dir` property specifies the location `$WLP_USER_DIR/shared/resources`. You can use this property name within your server configuration file. You can choose to create nested subdirectories within this directory that are appropriate for your APIs, service artifact files, and API requester artifact files.

For more information about storing shared files, see [“Location of shared configuration files” on page 327](#).

Administration and operation tasks in an HA environment

In an HA environment, administration and operation tasks require further considerations.

Stopping and starting z/OS Connect EE servers in an HA environment

There are some things you must consider when stopping or starting a z/OS Connect EE server.

When a server is shut down, the order of events causes the z/OS Connect EE APIs, services, and API requesters to become unavailable while the HTTP and HTTPS ports are still enabled. This can result in the client application receiving HTTP response code 404 (Not Found).

To avoid this situation, you can pause or disable the HTTP endpoints before shutting down the server. As soon as the endpoint is paused or disabled, TCP/IP port sharing will no longer route work to that server, until the HTTP endpoint is resumed or re-enabled. Use one of the following methods:

1. To pause HTTP endpoints, issue the **MODIFY** command from the z/OS console. For example, issue the command

```
MODIFY [jobname]identifier,PAUSE,TARGET='httpEndpointId1'
```

where `httpEndpointId1` is the **id** attribute value associated with the **httpEndpoint** configuration element that you want to pause. Multiple endpoints can be specified by using commas to separate the ID values.

For more information see [Pausing and resuming a Liberty server from the z/OS console](#).

2. To disable HTTP endpoints, set the `enabled` attribute on the **httpEndpoint** configuration elements to false.

Note: If the **httpEndpoint** element is defined in a shared configuration file, you will need to override the attribute definition to specify the `enabled` attribute uniquely in each server, so that you can control the value per server. For example, add the following entry to the `server.xml` files used by each server.

```
<!-- Override the httpEndpoint enabled attribute on a per server basis -->
<httpEndpoint id="defaultHttpEndpoint" enabled="false" />
```

When a server is started, its HTTP ports might be enabled before all the z/OS Connect EE APIs, services, and API requesters are started. In an HA environment with an active workload, requests might be distributed to the starting server before it is fully initialized. A similar approach to that described above can be used to start the server with the **httpEndpoint** initially disabled and then resume or enable the HTTP endpoint. Use one of the following methods:

1. To resume HTTP endpoints:, issue the **MODIFY** command from the z/OS console. For example: issue the command:

```
MODIFY [jobname]identifier,RESUME,TARGET='httpEndpointId1'
```

where *httpEndpointId1* is the **id** attribute value associated with the **httpEndpoint** configuration element that you want to resume. Multiple endpoints can be specified by using commas to separate the ID values.

For more information see [Pausing and resuming a Liberty server from the z/OS console](#).

2. To enable HTTP endpoints, set the enabled attribute on the **httpEndpoint** configuration elements to true.

Related information

Management of z/OS Connect EE APIs, services, and API requesters in an HA environment

Considerations for querying, starting, stopping, deploying, updating, and undeploying APIs, services, and API requesters in an HA environment.

z/OS Connect EE provides an administration interface for services, and a RESTful administration interface for APIs and API requesters. See “[Retrieving service information](#)” on page 606, “[Using the RESTful administration interface](#)” on page 675, and Chapter 24, “[How to manage API requesters](#),” on page 693. These administration interfaces connect to the HTTP or HTTPS port of a z/OS Connect EE server and provide various functions such as:

- Listing currently deployed APIs and services and retrieving details of an individual API or service.
- Stopping and starting individual APIs or services.
- Deploying, updating, or undeploying an API.

In an HA environment, HTTP, and HTTPS requests are targeted at a single host name and port but are then distributed across multiple servers when those servers use TCP/IP port sharing, Sysplex Distributor or both. Routing of an administration request cannot be guaranteed. Therefore, in an HA environment, administration, or operation actions through HTTP administration interfaces cannot be targeted at either a specific server or to all of the servers.

Querying APIs, services, and API requesters

If all the servers in a high availability group are configured with the same APIs, services, and API requesters, a shared HTTP or HTTPS port can still be used to retrieve details of currently deployed APIs, services, or API requesters, because any server can return the same information. However, it is not possible to determine the API and service status on any particular server. In that case, a second **httpEndpoint** can be configured as described below.

Starting and stopping APIs, services, and API requesters

To start or stop an individual API, service, or API requester, and to retrieve definitive deployment and status information from individual servers in an HA environment, configure a second **httpEndpoint** element in **server.xml** to specify an additional HTTP and HTTPS port pair for each server. These ports must be unique for each server and not shared. They are used only for the RESTful Administration interface administration and operation requests. Requests to invoke APIs and services continue to use the shared ports so that those requests are distributed across the servers.

For example, add the following entry into each server's **server.xml** file:

```
<httpEndpoint id="operationsHttpEndpoint" host="*" httpPort=""  
httpsPort="" />
```

Note:

1. In a production environment, you might restrict this administration endpoint to require SSL. By default, an **httpEndpoint** server configuration element uses the server's default SSL configuration, an **ssl** element with **id=defaultSSLConfig**. However, if you want to use different SSL certificates to

protect requests over this administration endpoint, you can associate this endpoint with a different SSL configuration. For more information, see the [Configuring an httpEndpoint to use an SSL configuration other than the default](#).

2. Requests received on either httpEndpoint are authenticated against the same z/OS Connect EE roles that are defined in your configured security repository. For more information, see [“Overview of z/OS Connect EE security” on page 331](#).

Deploying APIs

If the RESTful Administration interface is used to deploy, update, or undeploy APIs to a location directory that is shared by multiple servers, the time at which the change to the API takes effect in each server is unpredictable. This is because the server that the administration request is distributed to immediately implements the change to the API in its runtime and the API archive file is updated in the location directory. However, the other servers reflect the change to the API only when they refresh their API information, as specified by the **updateTrigger** attribute specified on `zosconnect_zosConnectAPIs` server configuration element. For more information, see [“zosconnect_zosConnectAPIs” on page 780](#).

A better alternative in an HA environment is to deploy or update APIs by copying or FTPing the archive files in binary mode to a single location directory that is shared by all servers. Similarly, an API archive file can be deleted from the location directory so that new, updated, or deleted APIs are reflected on all servers at the same time. The change to the API archive file again takes effect in the running servers based on the **updateTrigger** attribute that is specified on `zosconnect_zosConnectAPIs` server configuration element.

When run in production, an mbean trigger is preferable to a polled trigger for performance reasons.

Deploying services

To deploy, update, or undeploy services in an HA environment, use the standard Liberty functions appropriate for your service provider.

Updates to the server configuration file can be made to the common shared configuration file and take effect dependent on the **updateTrigger** attribute that is specified on the config server configuration element. For more information, see [Server configuration](#) in the *IBM WebSphere Application Server for z/OS Liberty* documentation.

If your z/OS Connect EE services use DataXForms, creation, modification, or deletion of the data transformation files in the UNIX System Services directory can be updated based on the `updateTrigger` attribute that is specified on the `zosconnect_zosConnectDataXform` server configuration element. For more information, see [“zosconnect_zosConnectDataXform” on page 782](#).

Note: If you specify the `updateTrigger` attribute value of mbean on any of the applicable server configuration elements, you must also configure your server to use the Java Management Extensions (JMX) connector, for more information see [“Using an MBean to trigger updates” on page 301](#).

Chapter 9. Securing z/OS Connect EE resources

z/OS Connect EE resources can be secured through user authority roles. Connections can be encrypted using AT-TLS. Access to applications can be configured globally or specifically.

Overview of z/OS Connect EE security

z/OS Connect EE security can operate with traditional z/OS security such as System Authorization Facility (SAF), with open standards such as Transport Layer Security (TLS), and with third party authentication tokens such as OAuth 2.0, OpenID Connect tokens and JSON Web Token (JWT).

The following sections introduce the security principles relevant to z/OS Connect EE.

Security principles

Confidentiality

Confidentiality ensures that an unauthorized party cannot obtain the meaning of the transferred or stored data. Typically confidentiality is achieved by encrypting the data.

Integrity

Integrity ensures that transmitted or stored information was not altered in an unauthorized or accidental manner. Typically it is a mechanism to verify that what is received over a network is the same as what was sent.

Authentication

Authentication is the process of validating the identity that is claimed by the accessing entity. Authentication is performed by verifying authentication information that is provided with the claimed identity. The authentication information is generally referred to as the accessor's credentials. A credential can be the accessor's name and password. It can also be a token that is provided by a trusted party, such as a JSON Web Token (JWT) or an X.509 certificate.

Authentication is usually one of the earliest steps in a request workflow. When authenticated, an identity can be asserted to the downstream process steps, meaning that these steps trust that the identity was successfully authenticated by the upstream steps.

Identification

Identification is the ability to assign an identity to the entity that is attempting to access the system. Typically the identity is used to control access to resources. Depending on the security model in which the identification is performed, the identity might come from the authentication credentials or it might be asserted from another server.

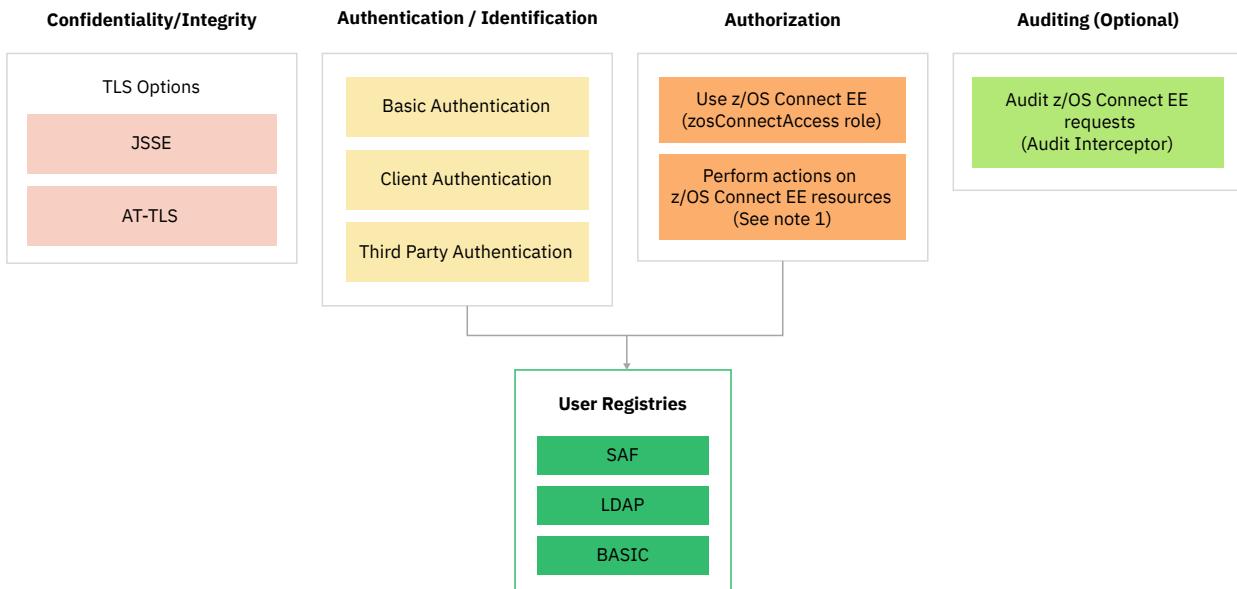
Authorization

Authorization is the process of checking whether an authenticated identity is permitted access to the resource that it is requesting. A typical implementation of authorization is to pass to the access control mechanism a security context that contains the authenticated identity.

Auditing

Auditing provides you with the ability to capture and record events such as an API request so that you can analyze them later, perhaps if your security was breached.

The following diagram shows the high-level security options available in z/OS Connect EE.

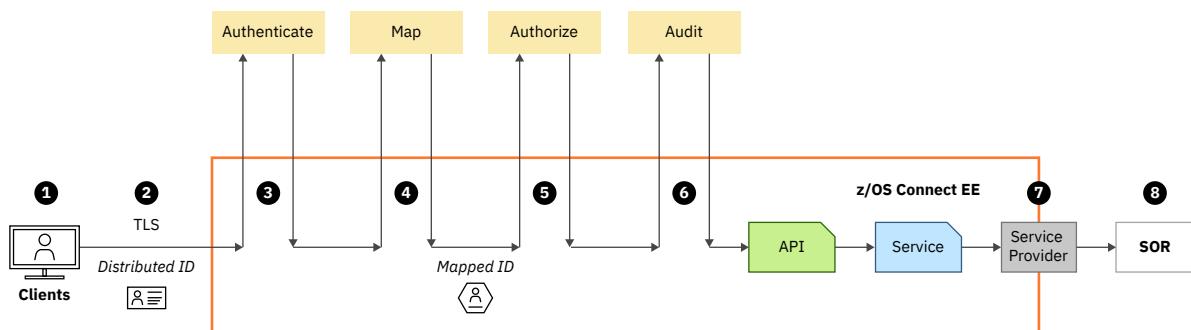


Note 1 The actions that can be controlled by authorization are: deploying, querying, updating, starting, stopping, and deleting of APIs, services and API requesters.

The following sections show how the security principles apply to z/OS Connect EE, when used as an API provider or API requester.

API provider security

The following diagram indicates where the security principles apply in an end to end flow when z/OS Connect EE acts as an API provider.



The flow includes the following security steps that can be performed by z/OS Connect EE.

1. The credentials are provided by the client. These can be a user ID and password, a third-party token, or a TLS certificate.
2. The credentials, including the identity, are passed on the connection between the client and the z/OS Connect EE server. The identity is typically a distributed ID, such as an X.500 distinguished name and associated LDAP realm, that originates from a remote system. Alternatively the identity might be a SAF user ID. The data that is sent on the connection can be encrypted using TLS.
3. The client is authenticated. This can be within the z/OS Connect EE server or by requesting verification from a third-party server.
4. The authenticated identity is mapped to a user ID in the z/OS Connect EE user registry.
5. The authenticated user ID is authorized to connect to z/OS Connect EE and to perform specific actions on z/OS Connect EE APIs or services.

6. The API or service request is audited.
7. Using a secure connection to the System of Record (SoR), the identity is asserted to be used to invoke the program or transaction in the SoR.
8. The program or transaction runs in the SoR using the asserted identity.

Security options for the API provider

Authentication and Identification

z/OS Connect EE supports the following authentication methods:

- **Basic authentication** that uses credentials in the form of a user ID and password that are verified by System Authorization Facility (SAF), Lightweight Directory Access Protocol (LDAP) or a basic user registry.
- **Client authentication** that uses a TLS client certificate to identify the client.
- **Third-party authentication** that uses a token such as a JSON Web Token (JWT), Security Assertion Markup Language (SAML) token or Lightweight Third-Party Authentication (LTPA) token. Alternatively, you can use custom authentication, such as an IBM WebSphere Trust Association Interceptor (TAI).

Optionally, the authenticated identity might be mapped to a different user ID in the user registry, for example, a distributed identity to be mapped to a SAF user ID. z/OS Connect EE can then authorize the mapped user ID using a SAF registry and pass it to a SoR.

For more information, see [“API provider authentication and identification” on page 349](#).

Authorization

z/OS Connect EE supports authorization for an authenticated user to:

- Grant third-party access to an API using the OAuth 2.0 protocol.
- Control access to z/OS Connect EE, using the `zosConnectAccess` role.
- Perform specific actions on a z/OS Connect EE API or service. z/OS Connect EE groups users into one of four authorization levels: Admin, Operations, Invoke, and Reader. This is implemented by the z/OS Connect EE authorization interceptor. You can enable the authorization interceptor globally in the z/OS Connect EE server or for individual APIs or services.

For more information, see [“API provider authorization” on page 373](#).

Confidentiality and Integrity

z/OS Connect EE supports TLS to ensure confidentiality and integrity of the request. z/OS Connect EE supports TLS through use of Java Secure Socket Extension (JSSE), Application Transparent Transport Layer Security (AT-TLS) and hardware cryptography to provide secure TCP/IP connections to and from a z/OS Connect EE server.

For more information, see [“API provider confidentiality and integrity” on page 336](#).

Auditing

z/OS Connect EE provides an audit interceptor. The z/OS Connect EE audit interceptor records information about API and service requests to SMF data sets on the z/OS operating system. SMF type 123, subtype 1 records are generated. These include the user name and, if present, the mapped user name, associated with the request. You can enable the audit interceptor globally in the z/OS Connect EE server or for individual APIs or services.

For more information, see [Chapter 11, “Monitoring,” on page 475](#) and [“Configuring the audit interceptor” on page 291](#)

System of Record security

The security options available when calling a transaction or program in a System of Record (SoR) from a z/OS Connect EE server, vary depending on the service provider that is used and the SoR. These options include:

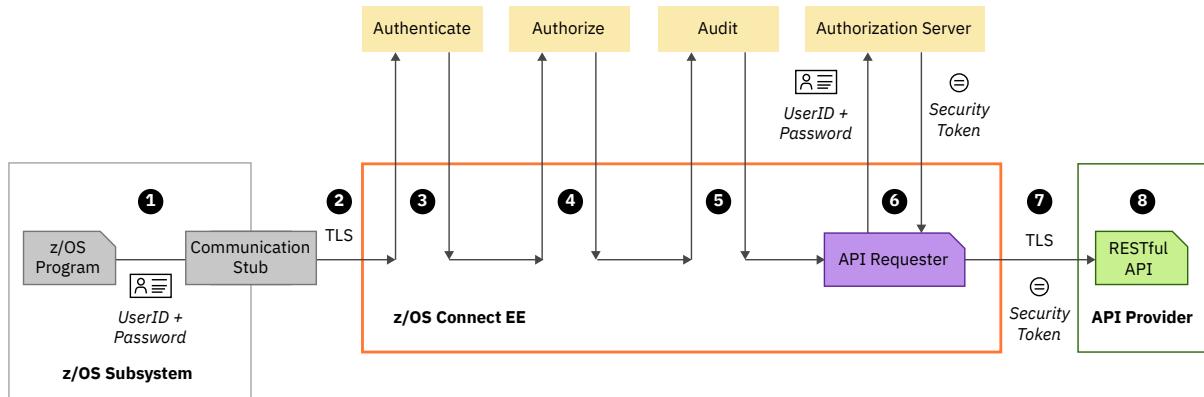
- Whether the connection is secured with TLS.

- Whether any credentials are required to establish the connection to the SoR.
- Whether a user identity is asserted, for example, when using identity propagation or identity assertion, to the SoR to run the program or transaction.

For more information, see [“Configuring security for an IPIC connection” on page 225](#), [“IMS service security process flow” on page 238](#), and [“Configuring security for a REST client connection” on page 287](#).

API requester security

The following diagram indicates where the security principles apply in an end to end flow with z/OS Connect EE acting as an API requester.



The API requester flow includes the following security steps that can be performed by z/OS Connect EE.

1. Two credentials, each consisting of a user ID and password, can be provided by the CICS, IMS or z/OS application and can be used for different purposes.
 - One user ID and password can be used for basic authentication by the z/OS Connect EE server. These credentials are specified in different places by the application. For example, in CEEOPTS environment variables for IMS and z/OS applications or in a CICS user exit for CICS.
 - The other user ID and password can be used to obtain a token from an authorization server to use on the request to the RESTful API. These credentials are specified as parameters in the data structure to pass to the communication stub for communication with the z/OS Connect EE server. Alternatively, a locally generated JWT can be configured.
 - Depending on the security configuration, either both or neither of these credentials might be required.
2. The connection between the CICS, IMS, or z/OS application and the z/OS Connect EE server can be secure with data sent on the connection encrypted using TLS.
3. z/OS Connect EE authenticates the CICS, IMS, or z/OS application.
4. The authenticated user ID is authorized to connect to z/OS Connect EE and to perform specific actions on z/OS Connect EE API requesters.
5. The API requester request is audited.
6. The user ID and password credentials are passed to an authorization server to obtain a security token. Alternatively, a locally generated JWT can be configured.
7. The connection to the external API provider can be secure, with security credentials such as a security token provided to invoke the RESTful API.
8. The RESTful API runs in the external API provider.

Security options for the API requester

Confidentiality and Integrity

z/OS Connect EE supports TLS to ensure confidentiality and integrity of the request.

- To provide secure connections between the CICS, IMS, or z/OS application and the z/OS Connect EE server:
 - For CICS, either CICS native TLS support or Application Transparent Transport Layer Security (AT-TLS) can be used.
 - For IMS or z/OS applications, AT-TLS can be used.
- z/OS Connect EE can use Java Secure Socket Extension (JSSE) or AT-TLS for secure connections between the z/OS Connect EE server and the RESTful API endpoint.

For more information, see [“API requester confidentiality and integrity” on page 394](#).

Authentication and Identification

z/OS Connect EE supports the following authentication methods:

- **Basic authentication** using credentials in the form of a user ID and password, sent from the CICS, IMS, or z/OS application, and verified using System Authorization Facility (SAF), Lightweight Directory Access Protocol (LDAP) or a basic user registry.
- **Client authentication** using a TLS client certificate to identify the CICS, IMS, or z/OS application.
 - For CICS, either CICS native TLS support or Application Transparent Transport Layer Security (AT-TLS) can be used.
 - For IMS or z/OS applications, Application Transparent Transport Layer Security (AT-TLS) can be used.

z/OS Connect EE supports identity assertion, which allows you to invoke an API requester from a z/OS application with an identity that is provided in the application context.

For more information, see [“API requester authentication and identification” on page 407](#).

Authorization

z/OS Connect EE supports authorization for an authenticated user to:

- Control access to z/OS Connect EE using the `zosConnectAccess` role.
- Perform specific actions on a z/OS Connect EE API requester. z/OS Connect EE groups users into one of four authorization levels: Admin, Operations, Invoke, and Reader. This is implemented by the z/OS Connect EE authorization interceptor. You can enable the authorization interceptor globally in the z/OS Connect EE server or for individual API requesters.

For more information, see [“API requester authorization” on page 446](#).

When you develop a CICS, IMS, or z/OS application to call an API that is protected by OAuth 2.0, you can include the information for the OAuth parameters in the request. For more information, see [“OAuth parameters and variables” on page 639](#)

Auditing

z/OS Connect EE provides an audit interceptor that records information about API requester requests to SMF datasets on the z/OS operating system. SMF type 123, subtype 1 records are generated and include the user name associated with the request. You can enable the audit interceptor globally in the z/OS Connect EE server or for individual API requesters.

For more information, see [Chapter 11, “Monitoring,” on page 475](#) and [“Configuring the audit interceptor” on page 291](#).

RESTful API security

Each RESTful API may have different security requirements, as defined in their Swagger documents. z/OS Connect EE supports the following methods for authenticating with a RESTful API:

- **Basic authentication** using credentials in the form of a user ID and password.
- **Client authentication** using a TLS client certificate to identify the z/OS Connect EE server.

- **OAuth 2.0** by configuring the z/OS Connect EE server to send a user ID and password to an authorization server to request an OAuth 2.0 access token. This token is then sent on the request to the RESTful API.
- **API keys** by specifying the API key values in the CICS, IMS, or z/OS application, which can then be sent on the request to the RESTful API.
- **JSON Web Token (JWT)** by configuring the z/OS Connect EE server to send a user ID and password to an authentication server to request a JWT or by configuring the z/OS Connect EE server to generate a JWT locally. This JWT is then sent on the request to the RESTful API.

For more information, see [“API requester authentication and identification” on page 407](#).

Related tasks

[“Configuring client certificates for server connections” on page 604](#)

API provider confidentiality and integrity

Learn how to maintain the confidentiality and integrity of the data that is handled by z/OS Connect EE.

Before you study this topic, you should be familiar with the information in [“Overview of z/OS Connect EE security” on page 331](#).

Confidentiality ensures that an unauthorized party cannot obtain the information in the transferred or stored data. Typically, confidentiality is achieved by encrypting the data.

Integrity ensures that transmitted or stored information is not altered in an unauthorized or accidental manner.

Securing communications to z/OS Connect EE

You can secure communications between a REST client and a z/OS Connect EE server by using the Transport Layer Security (TLS) protocol. TLS provides transport layer security that includes confidentiality, integrity, and authentication to secure the connection between a client and a z/OS Connect EE server. For more information on TLS, see [“Transport Layer Security \(TLS\)” on page 459](#).

z/OS Connect EE uses Java Secure Sockets Extension (JSSE) as the TLS implementation for secure connections. JSSE provides a framework and Java implementation that handles the handshake negotiation and protection capabilities that are provided by TLS. For more information, see [“Java Secure Sockets Extension \(JSSE\)” on page 460](#).

Alternatively you can use Application Transparent Transport Layer Security (AT-TLS), a capability of z/OS Communications Server, for transport layer security with z/OS Connect EE. For more information, see [“Application Transparent Transport Layer Security \(AT-TLS\)” on page 460](#).

Figure 70 on page 337 shows the TLS implementation options available for API provider.

- The HTTPS port 5002 is used for a TLS connection from a REST client. The port is associated with an SSL configuration in the z/OS Connect EE server.
- The HTTPS port 5003 is protected with an AT-TLS outbound policy. The port is associated with an SSL configuration in the z/OS Connect EE server.
- The HTTP port 5004 is protected with both AT-TLS outbound and inbound policies so the TLS connection is managed by AT-TLS. Client certificate authentication cannot be used for this connection.

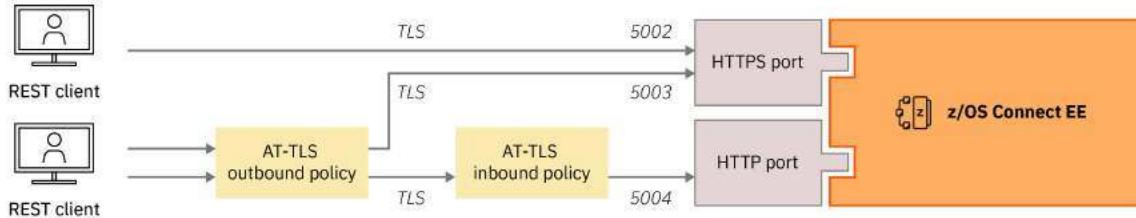


Figure 70. TLS implementation options for API provider.

In most cases, REST clients do not connect directly to a z/OS Connect EE server, but through an intermediate server, as shown in [Figure 71 on page 337](#).

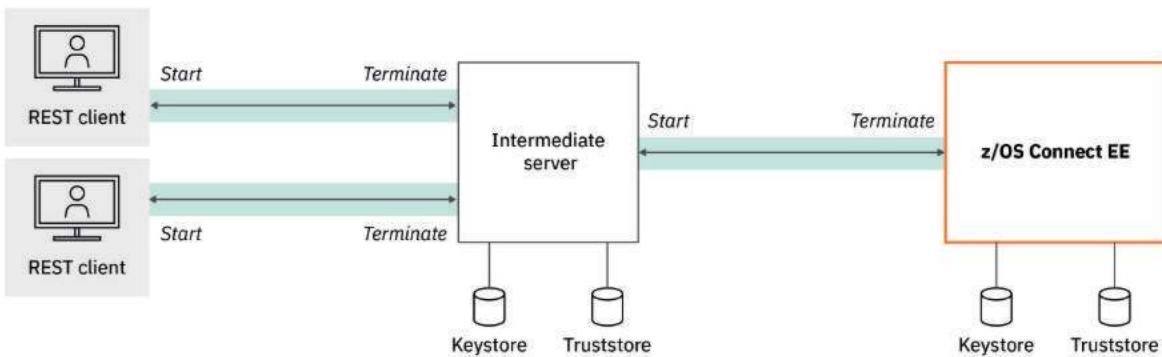


Figure 71. Using an intermediate server for requests to a z/OS Connect EE server.

There might be hundreds or thousands of TLS connections constantly being created and released between the various REST clients and the intermediate server. This activity focuses the CPU cost and management of the TLS certificates on the intermediate server, and allows for comparatively fewer TLS connections between the intermediate server and the z/OS Connect EE server. These TLS connections can be long lived through using persistent connections.

Persistent connections

Persistent connections can reduce CPU usage.

When TLS is used, CPU intensive processing occurs most during the handshake phase. The simplest way to reduce this cost is to enable persistent HTTP connections because a TLS handshake occurs only during the creation of the HTTP connection. By using this mechanism, the cost of the handshake is spread over multiple requests.

In z/OS Connect EE, the default number of persistent requests that can reuse an HTTP connection is 100. After 100 requests, the HTTP connection and the underlying socket are closed. If HTTPS is used, a TLS handshake therefore occurs every 100 requests. To improve performance, you can increase this value by setting the `maxKeepAliveRequests` attribute on an `httpOptions` element that is referenced from the `httpEndpoint` element. You can also increase the amount of time that a connection persists when it is not being used by changing the `persistTimeout` attribute on the `httpOptions` element. The default is 30 seconds.

Even after the persistent maximum number of requests or timeout is reached, the SSL session ID can be used to avoid another costly handshake. During the handshake phase, z/OS Connect EE and the REST client create an SSL session ID, which is used for all the persistent HTTP connections. When another handshake is needed, and the REST client calls z/OS Connect EE with the same SSL session ID as used in the previous handshake, z/OS Connect EE can decide to do a null handshake. The null handshake means

that the connection reuses the same symmetric key as the previous session to avoid the handshake phase. The connection is a prolongation of the persistent HTTP connections. The control of the usage of the SSL session ID is done in the `httpEndpoint`, which points to an `sslOptions` element where the `sslSessionTimeout` attribute establishes the time the encryption symmetric key for HTTP requests is used.

Additional trust can be established between the intermediate server and the z/OS Connect EE server by using mutual TLS authentication, in which the two parties authenticate each other.

Configuring TLS for z/OS Connect EE resources

By default, z/OS Connect EE requires clients to connect to the server into the HTTPS port. The `requireSecure` attribute on the `zosconnect_zosConnectManager`, `zosConnectAPI`, `service`, and `zosconnect_zosConnectService` elements of the `server.xml` configuration file, control whether a specific request must connect into the HTTPS port. The default value of the `requireSecure` attribute is `true`. You can set this value to `false` to remove this requirement at a specific scope or when AT-TLS inbound is used.

TLS server authentication is enabled by default, but you can also configure TLS client authentication, also called mutual TLS authentication. In this configuration, the client authenticates itself with a personal certificate. You can then choose whether to also use the client certificate to establish the authenticated identity for the request.

Alternatively, you can use TLS server authentication, or mutual TLS authentication with other authentication methods. For example,

- Use basic authentication.
- Use an authentication mechanism such as OpenID Connect (that uses a JWT), OAuth, or TAI. These mechanisms all take precedence over the client certificate authentication method.

For more information, see [“API provider authentication and identification” on page 349](#).

Configuring z/OS Connect EE SSL elements

If you set `requireSecure` attribute to `true` then the z/OS Connect EE server must be configured to use TLS.

z/OS Connect EE TLS is configured by using elements in the `server.xml` configuration file. To configure TLS for connections from client applications, an SSL configuration is associated with an HTTP endpoint (using the `httpEndpoint` element).

z/OS Connect EE includes a default SSL configuration. This default configuration is typically customized to add your own keystores and truststores, configure whether client authentication is required or supported, or whether only server authentication is required.

For information, see [“How to configure TLS with RACF key rings” on page 339](#).

In a development environment, you might choose to start testing with the default SSL configuration. You can use this default configuration to get started quickly with TLS because a keystore and certificate are automatically created for you. For production environments, use SAF to create your own key rings and certificates.

To use the default SSL configuration, add the following element to dynamically create a keystore:

```
<keyStore id="defaultKeyStore" password="yourpassword" />
```

For more information about using the default SSL configuration, see [SSL defaults in Liberty](#) in the [WebSphere Application Server for z/OS Liberty](#) documentation.

If you require different TLS implementations, such as using different certificates for different clients, you can configure extra SSL configuration elements and HTTP endpoint elements, and associate the appropriate SSL configuration to the appropriate HTTP endpoint (HTTPS) ports.

Other considerations

When you implement API provider confidentiality and integrity, you might also want to consider the following options:

Cipher suites

You can use cipher suites to control the cryptographic algorithms used on the TLS connection. For more information, see [“Cipher suites” on page 462](#).

Hardware cryptography

You can use hardware cryptography with TLS connections. For more information, see [“Hardware cryptography” on page 463](#).

SP800-131a

You can configure z/OS Connect EE to meet the SP800-131a specification. For more information, see [“SP800-131a” on page 464](#).

How to configure TLS with RACF key rings

Configure a TLS connection between a client or intermediate server such as an API Gateway, and a z/OS Connect EE server.

This task is applicable when z/OS Connect EE is used as an API provider.

- Before you begin this task, ensure you understand the information in [“API provider confidentiality and integrity” on page 336](#).
- You must have authorization to issue the following **RACDCERT** commands: **ADD**, **ADDRING**, **CONNECT**, **EXPORT**, **GENCERT**, **LIST**, and **LSTRING**, to create a RACF key ring and certificates. For more information about the **RACDCERT** commands and the authorizations that are required, see [RACDCERT \(Manage RACF digital certificates\)](#) in the *z/OS Security Server RACF Command Language Reference*.
- You must have access to the **keytool** command for creating Java KeyStores (JKS) and certificates for the client. This command is provided with the IBM SDK for z/OS, Java Technology Edition. For more information about the **keytool** command, see [Keytool](#) in the *IBM SDK, Java Technology Edition Security Guide*.
- You must have write access to the `server.xml` configuration file.

How to configure a TLS connection between z/OS Connect EE and the client

Configure a z/OS Connect EE server so that clients must connect by using an HTTPS connection with TLS server authentication.

About this task

This configuration creates the following artifacts:

- A certificate authority (CA) signed personal RACF certificate for a z/OS Connect EE server to identify itself on TLS connections.
- A RACF key ring to act as the server's keystore.
- A Java KeyStore (JKS) for a client to use as its truststore so that it can trust the certificate that is presented by the z/OS Connect EE server on the TLS connection.

You can also configure TLS client authentication, sometimes referred to as mutual TLS authentication, to require the client to provide its personal certificate on the connection. This configuration creates the following artifacts:

- A CA signed personal certificate for the client to identify itself on the TLS connections.
- A JKS to act as the client's keystore.
- A RACF key ring for the z/OS Connect EE server to use as its truststore so that it can trust the certificate that is presented by the client.

The certificates, key rings, and JKS files that are used in this task are described in [Figure 72 on page 340](#).

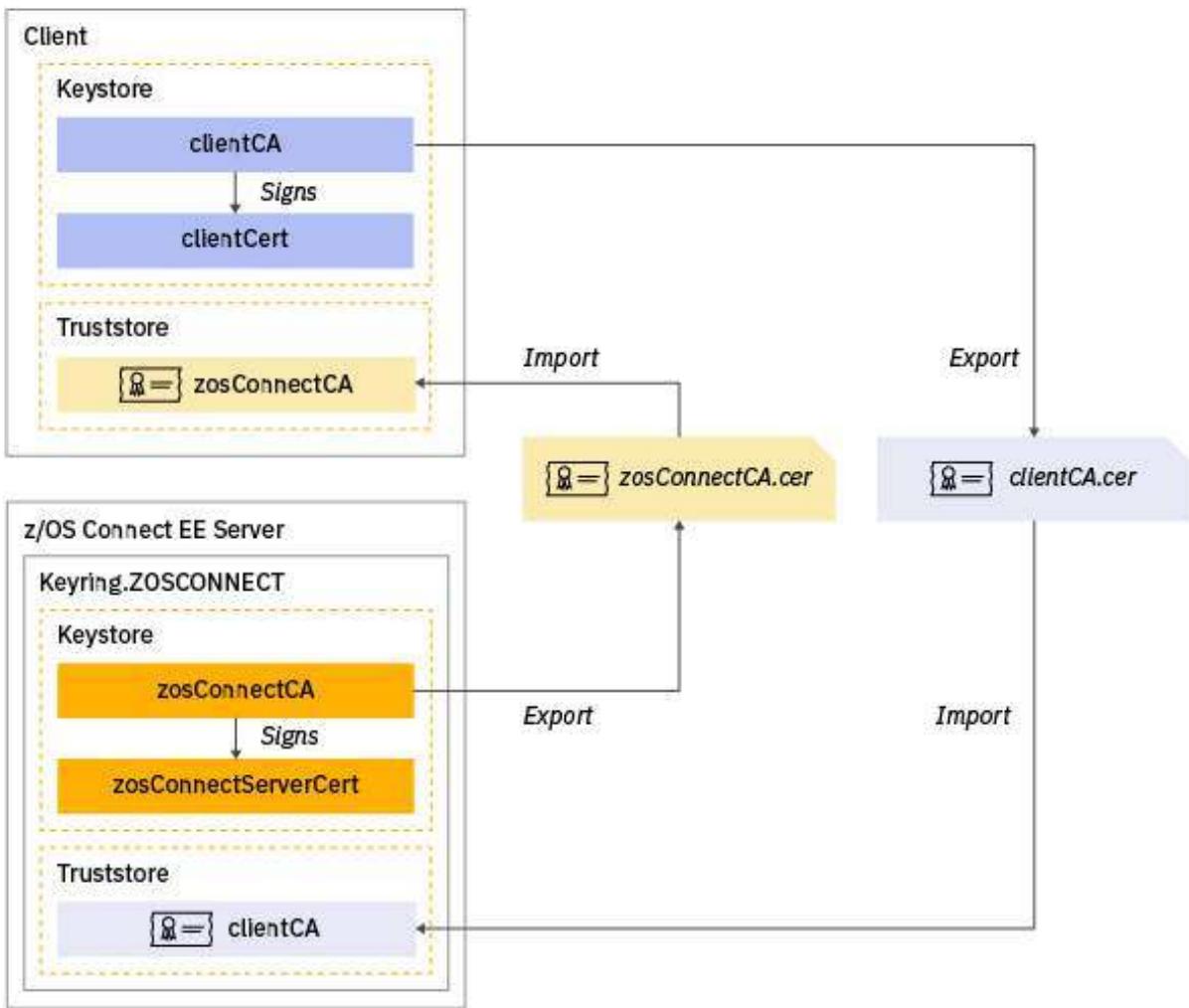


Figure 72. How certificates, key rings, and JKS files are used in this task.

This task makes the following assumptions:

- RACF is the security manager for the z/OS Connect EE server. If you are using an alternative External Security Manager, refer to the appropriate product documentation for the equivalent commands.
- The client, which might be an intermediate server such as an API Gateway, can use JKS files for its keystore and truststore. This task uses a JKS for the client, but other keystore types or SAF key rings might be used depending on the client and the operating system that it runs on.
- The **keytool** command is used to create the certificates and JKS files for the client. The z/OS Connect EE server uses a single RACF key ring for both its keystore and truststore. An alternative method is to use a separate RACF key ring for each role.
- The client uses separate JKS files for its keystore and truststore.
- The default z/OS Connect EE SSL elements in the `server.xml` configuration file are customized. This method is typical for configuring TLS on the z/OS Connect EE server's default HTTPS port. If you are configuring TLS for an additional HTTPS port, follow these instructions, but replace the default `id` attribute values of the `ssl` and `keyStore` elements with customized values. You would also need to associate the customized `ssl` element with the appropriate `httpEndpoint` element by configuring the `sslOptions` subelement. For example,

```
<httpEndpoint id="myHttpEndpoint" host="*"
  httpPort="9081" httpsPort="9444">
```

```
<sslOptions sslRef="mySSLConfig" />  
</httpEndpoint>
```

In this example, `mySSLConfig` is the `id` attribute value of the `ssl` element.

Note: The term *label* is used by RACF and *alias* is used by JKS to reference the same artifact, therefore in this documentation, the phrase *label or alias* is used for clarity.

Procedure

In the following steps, you configure the TLS connection between the z/OS Connect EE server (steps 1 - 5) and the client (steps 6 - 9). Finally, you configure the z/OS Connect EE server to use the RACF key ring (steps 10 - 13).

Enter the RACF commands from the z/OS LPAR where the z/OS Connect EE server is hosted.

1. Create a RACF key ring for the z/OS Connect EE server to use as its keystore.

Enter the following command:

```
RACDCERT ID(ZCSERV1) ADDRING(Keyring.ZOSCONN)
```

The command uses the following values:

- ZCSERV1 is the user ID that owns the key ring. The user ID under which the z/OS Connect EE server runs must have READ access to this key ring. Either specify the server's user ID on the ID property or ensure that the server's user ID has READ access to the key ring owned by an alternative user ID.
- Keyring.ZOSCONN is the name of the key ring to be created.

2. Create a CA certificate for the z/OS Connect EE server.

- a) Create a self-signed RSA key pair to act as a CA certificate. A *key pair* consists of a public and private key.

Enter the following command:

```
RACDCERT GENCERT CERTAUTH SUBJECTSDN(CN('CA for zosConnect') O('IBM')  
OU('zosConnect') C('US')) SIZE(2048) WITHLABEL('zosConnectCA')  
NOTAFTER(DATE(2029-12-31))
```

The command uses the following values:

- CN('CA for zosConnect') O('IBM') OU('zosConnect') C('US') is an example distinguished name (DN) for the certificate.
- zosConnectCA is the label or alias of the certificate.
- 2029-12-31 is the expiry date of the certificate.

- b) Connect (add) the CA certificate to the key ring.

Enter the following command:

```
RACDCERT ID(ZCSERV1) CONNECT(RING(Keyring.ZOSCONN) LABEL('zosConnectCA')  
CERTAUTH)
```

The command uses the following values:

- ZCSERV1 is the user ID that owns the key ring.
- Keyring.ZOSCONN is the name of the key ring.
- zosConnectCA is the label or alias of the certificate to be connected to the key ring.

3. Create a personal certificate, signed by the CA certificate, for the z/OS Connect EE server.

- a) Create an RSA key pair for the z/OS Connect EE server signed by the CA certificate.

Enter the following command:

```
RACDCERT ID(ZCSERV1) GENCERT SUBJECTSDN(CN('myServer.host.com') O('IBM')  
OU('zosConnect') C('US')) SIZE(2048) SIGNWITH(CERTAUTH)
```

```
LABEL('zosConnectCA')) WITHLABEL('zosConnectServerCert')
NOTAFTER(DATE(2029-12-31))
```

The command uses the following values:

- CN('myServer.host.com') O('IBM') OU('zosConnect') C('US') is an example distinguished name (DN) for the certificate. The common name (CN) value is typically the hostname of the z/OS LPAR that hosts the z/OS Connect EE server.
- zosConnectCA is the label or alias for the CA certificate that is used to sign the personal certificate.
- zosConnectServerCert is the label, or alias, for the personal certificate to be created and signed.

b) Connect (add) the personal certificate to the key ring.

Enter the following command:

```
RACDCERT ID(ZCSERV1) CONNECT(RING(Keyring.ZOSCONN)
LABEL('zosConnectServerCert'))
```

The command uses the following values:

- ZCSERV1 is the user ID that owns the key ring.
- Keyring.ZOSCONN is the name of the key ring.
- zosConnectServerCert is the label or alias of the personal certificate to be connected to the key ring.

4. Confirm that the key ring and certificates were created correctly.

a) List the certificates in the key ring.

Enter the following command:

```
RACDCERT ID(ZCSERV1) LISTRING(Keyring.ZOSCONN)
```

The following screen capture shows the expected response:

Ring: >Keyring.ZOSCONN<			
Certificate Label Name	Cert Owner	USAGE	DEFAULT
zosConnectCA	CERTAUTH	CERTAUTH	NO
zosConnectServerCert	ID(ZCSERV1)	PERSONAL	NO

b) List the details of the CA certificate.

Enter the command:

```
RACDCERT CERTAUTH LIST(LABEL('zosConnectCA'))
```

The expected response contains the following information:

```
Issuer's Name:  
>CN=CA for zosConnect.OU=zosConnect.O=IBM.C=US<
Subject's Name:  
>CN=CA for zosConnect.OU=zosConnect.O=IBM.C=US<
```

c) List details of the z/OS Connect EE server's personal certificate.

Enter the command:

```
RACDCERT ID(ZCSERV1) LIST(LABEL('zosConnectServerCert'))
```

The expected response contains the following information:

```
Issuer's Name:  
>CN=CA for zosConnect.OU=zosConnect.O=IBM.C=US<  
Subject's Name:  
>CN=myServer.host.com.OU=zosConnect.O=IBM.C=US<
```

5. Export the z/OS Connect EE server's CA certificate, containing the public key, to a z/OS sequential file. Enter the following command:

```
RACDCERT CERTAUTH EXPORT(LABEL('zosConnectCA')) DSN('ZCSERV1.CERTS.ZCCA')  
FORMAT(CERTDER)
```

The command uses the following values:

- `zosConnectCA` is the label or alias for the CA certificate.
- `'ZCSERV1.CERTS.ZCCA'` is the name of the target z/OS sequential file.
- `CERTDER` specifies a DER encoded X.509 certificate.

In the following steps, you configure the TLS connection on the client.

6. Prepare to use the **keytool** command.

- a) Create a directory to contain the certificates and JKS files to be created by the keytool command.

In a later step, you copy those files to other locations, so if you run the **keytool** command on z/OS, make the location a z/OS UNIX System Services (USS) directory. For example, `/u/myuser/zOSConnectJKS`.

- b) Add the keytool command to the PATH environment variable.

For example,

```
export PATH=$PATH:<javaInstallPath>/bin
```

The value `<javaInstallPath>` is the installation path of the IBM SDK for z/OS that includes the **keytool** command.

7. Import the CA certificate used to sign the z/OS Connect EE server's personal certificate into the client's JKS truststore as a trusted certificate.

- a) Transfer in binary format, the z/OS sequential file `'ZCSERV1.CERTS.ZCCA'` containing the exported CA certificate to the client's workstation.

The method that you use depends on the operating system on which the client is running and the file transfer utilities you have access to. You might be able to FTP to the z/OS LPAR and get the z/OS sequential file directly, or you might first need to copy the z/OS sequential file to a USS file, and then get the USS file. For example,

```
cp "'ZCSERV1.CERTS.ZCCA'" /u/myuser/zOSConnectJKS/zosConnectCA.cer
```

- b) Name the CA certificate file that you received onto the client system as `zosConnectCA.cer`

- c) Enter the following command from the USS directory you created in step [“6” on page 343](#).

```
keytool -importcert  
-file zosConnectCA.cer  
-alias zosConnectCA  
-keystore clientTrust.jks  
-storepass passw0rd  
-storetype jks
```

The command uses the following values:

- `zosConnectCA` is the alias to be given to the CA certificate to be imported.
- `zosConnectCA.cer` is the name of the certificate file that contains the public key of the CA certificate to be imported.
- `clientTrust.jks` is the name of the JKS file into which the certificate is to be imported. This file is used as the client's truststore.

The command creates the JKS file if it does not exist and issues the following prompt:

```
Trust this certificate? [no]:
```

Enter yes in response.

8. Check that the CA certificate was added to the client's truststore correctly.

List the contents of the client's truststore by issuing the following command from the USS directory you created in step “[6](#)” on page 343.

```
keytool -list -v -keystore clientTrust.jks
```

The expected response contains the following information:

```
Your keystore contains 1 entry
Alias name: zosconnectca
Entry type: trustedCertEntry
Owner: CN=CA for zosConnect, OU=zosConnect, O=IBM, C=US
Issuer: CN=CA for zosConnect, OU=zosConnect, O=IBM, C=US
```

9. Make the client's truststore JKS file available to the client.

In this task, the client's truststore is called `clientTrust.jks`. Transfer this file in binary mode to the workstation that hosts the client. The instructions for configuring your client to use these JKS files depend on your client application. For more information, see the documentation for the client application.

In the following steps, you configure the z/OS Connect EE server to use the RACF key ring.

10. Configure the HTTPS port for the z/OS Connect EE server.

Identify an unused TCP/IP port on your z/OS LPAR, which can be used as the HTTPS port for z/OS Connect EE. Specify the port value on the `httpsPort` attribute of the `httpEndpoint` element in the `server.xml` configuration file.

For example, to use 9443 as the default HTTPS port, add the following element to the configuration:

```
<httpEndpoint id="defaultHttpEndpoint" host="*"
    httpPort="9080" httpsPort="9443"/>
```

11. Configure the z/OS Connect EE server to require HTTPS requests.

Set the attribute `requireSecure="true"` in the `server.xml` configuration file to force all requests at that scope to use an HTTPS connection. This attribute can be set at different scopes:

- To require an HTTPS connection globally for the server, set `requireSecure="true"` on the `zosconnect_zosConnectManager` element in the `server.xml` configuration file. For example,

```
<zosconnect_zosConnectManager requireSecure="true" ... />
```

- To require an HTTPS connection for a specific API, set `requireSecure="true"` on the `zosConnectAPI` element in the `server.xml` configuration file. For example,

```
<zosconnect_zosConnectAPIs>
    <zosConnectAPI name="Api1" requireSecure="true"/>
</zosconnect_zosConnectAPIs>
```

This setting takes precedence over the global setting.

- To require an HTTPS connection for a specific service, set `requireSecure="true"` on the `service` element in the `server.xml` configuration file. For example,

```
<zosconnect_services>
    <service name="Service1" requireSecure="true"/>
</zosconnect_services>
```

This setting takes precedence over the global setting.

For more information about these elements, see [“Configuration elements” on page 753](#) in the *Reference* section.

12. Configure the SSL configuration to be used by the z/OS Connect EE server.

Create an `ssl` repertoire element in the `server.xml` configuration file, with default values for the `id` and `keyStoreRef` attributes. For example,

```
<ssl id="defaultSSLConfig"
      keyStoreRef="defaultKeyStore" />
```

For more information about the `ssl` repertoire element, see the [Server configuration](#) section in the *IBM WebSphere Application Server for z/OS Liberty* documentation.

The server now has only a keystore. Because it does not yet have any trusted certificates, it does not require a truststore.

13. Configure the z/OS Connect EE server to reference the RACF key ring that contains the server's personal certificate and CA certificate.

Create a `keyStore` element in the `server.xml` configuration file. For example,

```
<keyStore id="defaultKeyStore"
          fileBased="false"
          location="safkeyring:///Keyring.ZOSCONN"
          password="password"
          readOnly="true"
          type="JCERACFKS" />
```

The element uses the following values:

- `defaultKeyStore` must match the value that is specified on the `keyStoreRef` attribute of the `ssl` element.
- The `location` value must be the RACF key ring that acts as the server's keystore.
- The `password` attribute is mandatory, so a value must be specified. However, the value is not used when `type="JCERACFKS"` because RACF key rings are not secured with passwords.

What to do next

You can optionally configure TLS client authentication for this connection by following the steps in [“How to configure TLS client authentication” on page 345](#)

How to configure TLS client authentication

Before you begin

This task is optional, but before attempting this task, you must complete the prerequisite task [Configure a TLS connection between z/OS Connect EE and the client](#).

About this task

This optional task describes additional configuration for TLS client authentication, also called mutual TLS authentication, to require the client to provide its personal certificate on the connection. It does not include the additional configuration that is required to use the client's personal certificate to authenticate with a z/OS Connect EE server, but is a prerequisite to that task. For more information about configuring a client certificate to authenticate with a z/OS Connect EE server, see [“How to configure client certificate authentication with RACF” on page 371](#).

Procedure

In the following steps, you configure TLS client authentication for the client.

1. Create a self-signed certificate for the client.

Generate a self-signed RSA key pair for the client and add them to a JKS file, which acts as the client's keystore. Enter the following command from the USS directory you created in step “6” on

[page 343 of “How to configure a TLS connection between z/OS Connect EE and the client” on page 339](#)

```
keytool -genkeypair
    -alias clientCert
    -dname "CN=myClient.host.com, O=IBM, C=US"
    -keyalg RSA
    -keypass passw0rd
    -keysize 2048
    -keystore clientKey.jks
    -storepass passw0rd
    -validity 365
```

The command uses the following values:

- clientCert is the alias of the personal certificate to be created.
- CN=myClient.host.com, O=IBM, C=US is an example distinguished name (DN) for the certificate. The CN value is typically the host name of the client that owns the certificate.
- clientKey.jks is the name of the JKS file to be dynamically created to act as the client's keystore.

This command creates the JKS file if it does not exist.

2. Sign the client's personal certificate with a CA certificate.

A certificate signing request (CSR) is created for the self-signed personal certificate and sent to a certificate authority (CA). The CA verifies the request and returns a CA signed version of the personal certificate, a CA root certificate and optionally, an intermediate certificate.

a) Create a certificate signing request (CSR) for the personal certificate.

Enter the following command from the USS directory you created in step [“6” on page 343 of “How to configure a TLS connection between z/OS Connect EE and the client” on page 339](#).

```
keytool -certreq
    -alias clientCert
    -keystore clientKey.jks
    -file clientCert.csr
    -storepass passw0rd
```

The command uses the following values:

- clientCert is the alias of the personal certificate to be signed.
- clientKey.jks is the name of the JKS file that contains the personal certificate to be signed.
- clientCert.csr is the name of the CSR file to be created.

b) Send the CSR (clientCert.csr), to your preferred certificate authority with any additional details they require.

The certificate authority returns a CA root certificate, the signed client's personal certificate, and optionally an intermediate certificate.

c) Import the CA root certificate and if present, the intermediate certificate, into the client's JKS keystore.

Enter the following command from the USS directory that you created in step [“6” on page 343 of “How to configure a TLS connection between z/OS Connect EE and the client” on page 339](#).

```
keytool -importcert
    -file clientCA.cer
    -alias clientCA
    -keypass passw0rd
    -keystore clientKey.jks
    -storepass passw0rd
    -storetype jks
```

This command issues the following prompt:

```
Trust this certificate? [no]:
```

Enter yes in response.

The command uses the following values:

- clientCA is the alias to be given to the CA root certificate to be imported.
- clientCA.cer is the name of the certificate file that is returned by the certificate authority. It contains the CA root certificate to be imported.
- clientKey.jks is the name of the JKS file into which the certificate is to be imported. This file is the client's keystore.

d) Import the signed client's personal certificate.

Enter the following command from the USS directory that you created in step “6” on page 343 of “How to configure a TLS connection between z/OS Connect EE and the client” on page 339

```
keytool -importcert  
    -file clientCertSigned.cer  
    -alias clientCert  
    -keystore clientTrust.jks  
    -storepass passw0rd  
    -storetype jks
```

The command uses the following values:

- clientCert is the alias to be given to the signed personal certificate to be imported. It must match the name of the original self-signed personal certificate, so that one is replaced in the keystore.
- clientCertSigned.cer is the name of the certificate file that is returned by the certificate authority, and contains the signed personal certificate to be imported.
- clientKey.jks is the name of the JKS file into which the certificate is to be imported. This is the client's keystore.

3. Confirm that the client's keystore and personal certificate are created correctly.

Enter the following command from the USS directory that you created in step “6” on page 343 of “How to configure a TLS connection between z/OS Connect EE and the client” on page 339 to list the contents of the client's keystore.

```
keytool -list -v -keystore clientKey.jks
```

The expected response should contain entries for the signed personal certificate with an Entry type of PrivateKeyEntry and, for the CA root certificate (and if applicable, the intermediate certificate), an entry with Entry type of trustedCertEntry.

4. Export the client's CA certificate (public key) to a certificate file.

Enter the following command from the USS directory that you created in step “6” on page 343 of “How to configure a TLS connection between z/OS Connect EE and the client” on page 339.

```
keytool -exportcert  
    -alias clientCA  
    -file clientCA.cer  
    -keystore clientKey.jks  
    -storepass passw0rd  
    -storetype jks
```

The command uses the following values:

- clientCA is the alias of the CA certificate to be exported.

- `clientCA.cer` is the name of the certificate file to be created.
- `clientKey.jks` is the name of the JKS file from which the certificate is to be exported.

5. Make the client's keystore JKS file available to the client.

The client's keystore that is created in this task is called `clientKey.jks`. Transfer this file in binary mode to the workstation that hosts the client. The instructions for configuring your client to use these JKS files depend on your client application. For more information, see the documentation for the client application.

In the following steps, you configure TLS client authentication for the z/OS Connect EE server.

6. Connect the client's CA certificate (public key) to the z/OS Connect EE server's RACF key ring truststore as a trusted certificate.

- Transfer the exported client's CA certificate in binary format to the z/OS LPAR where the z/OS Connect EE server is hosted, as a z/OS sequential file.

Name the created z/OS sequential file as '`ZCSERV1.CERTS.CLIENT`'.

The transfer method that you use depends on the operating system on which you ran the `keytool` command. You might be able to FTP to the z/OS LPAR and put the z/OS sequential file directly, or you may first need to FTP the file to a USS directory and then copy that USS file to be a z/OS sequential file. For example,

```
cp /u/myuser/zOSConnectJKS/clientCA.cer " //'ZCSERV1.CERTS.CLIENTCA'"
```

- Import the client's CA certificate (public key) into RACF.

Enter the following command

```
RACDCERT ID(ZCSERV1) ADD('ZCSERV1.CERTS.CLIENTCA') WITHLABEL('clientCA') TRUST
```

- Connect (add) the client's CA certificate (public key) as a trusted (CERTAUTH) certificate to the RACF key ring used by the z/OS Connect EE server as its truststore.

Enter the following command:

```
RACDCERT ID(ZCSERV1) CONNECT(RING(Keyring.ZOSCONN) LABEL('clientCA') USAGE(CERTAUTH))
```

7. Confirm that the client's CA certificate is connected to the z/OS Connect EE server's key ring correctly.

Enter the following command to list the certificates in the key ring:

```
RACDCERT ID(ZCSERV1) LISTRING(Keyring.ZOSCONN)
```

This screen capture shows the expected response:

Ring:			
>Keyring.ZOSCONN<	Cert Owner	USAGE	DEFAULT
zosConnectCA	CERTAUTH	CERTAUTH	NO
zosConnectServerCert	ID(ZCSERV1)	PERSONAL	NO
clientCA	ID(ZCSERV1)	CERTAUTH	NO

8. Edit the SSL configuration to be used by the z/OS Connect EE server.

Edit the existing `ssl` repertoire element in the `server.xml` configuration file, to add a `trustStoreRef` attribute with the default value. Set client authentication as required by setting `clientAuthentication="true"`. For example,

```
<ssl id="defaultSSLConfig"
    keyStoreRef="defaultKeyStore"
    trustStoreRef="defaultTrustStore"
    clientAuthentication="true" />
```

9. Create a `keyStore` element in the `server.xml` configuration file for the z/OS Connect EE server's truststore.

The keyStore element is also used for truststores. For example,

```
<keyStore id="defaultTrustStore"
  fileBased="false"
  location="safkeyring:///Keyring.ZOSCONN"
  password="password"
  readOnly="true"
  type="JCERACFKS" />
```

The element uses the following values:

- defaultTrustStore must match the value that is specified on the trustStoreRef attribute of the ssl element.
- The location value must be the RACF key ring that acts as the server's truststore.
- The password attribute is mandatory, so a value must be specified. However, the value is not used when type="JCERACFKS" because RACF key rings are not secured with passwords.

10. Start, or restart the server if it was already running, to pick up the changes.

The messages.log file should contain the following message:

```
CWWK00219I: TCP Channel defaultHttpEndpoint-ssl has been started and is now listening for
requests on host * (IPv6) port 9443
```

The message uses the following values:

- defaultHttpEndpoint-ssl is the id attribute value of the httpEndpoint element followed by -ssl.
- * is the value of the httpEndpoint element host attribute.
- 9443 is the value of the httpEndpoint element httpsPort attribute.

API provider authentication and identification

Learn how requests sent to a z/OS Connect EE server are authenticated.

Before you study this topic, you should be familiar with the information in [“Overview of z/OS Connect EE security” on page 331](#).

Consider the following requirements to implement authentication for clients that need to connect to a z/OS Connect EE server:

- Authentication options.
- User registries.
- Caching authentication credentials.

By default, z/OS Connect EE requires that all requests are authenticated. Successful authentication that uses any of the supported authentication methods results in an authenticated user ID being associated with the request. The authenticated user ID is also checked to ensure that it is authorized to access z/OS Connect EE.

Authentication is governed by the requireAuth attribute of the zosconnect_zosConnectManager element in the server.xml configuration file. If this attribute is set to true, all requests to a z/OS Connect EE server are authenticated. You can override this global setting by specifying the requireAuth attribute on individual entries, such as for a particular API or service.

Note: If requireAuth="true" is configured for a service, then authentication is checked only if that service is invoked directly by an HTTP or HTTPS request. It does not take effect if that service is invoked by an API.

Three methods are provided for authentication between clients and a z/OS Connect EE server:

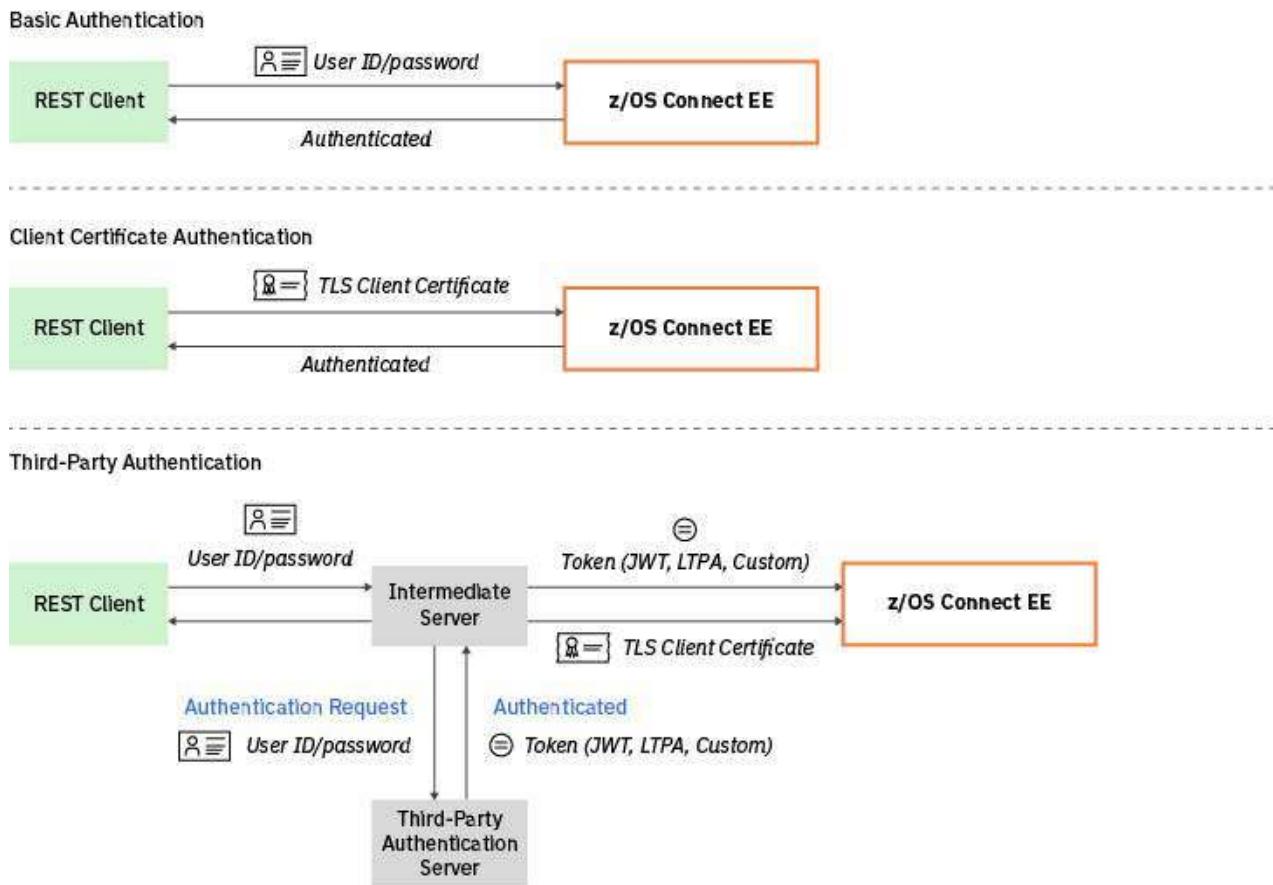


Figure 73. The three methods of authentication.

Basic authentication

When basic authentication is used, the REST client sends, or is prompted for, a user ID and password on the request to the z/OS Connect EE server. The server authenticates the credentials based on information in the user registry.

Client certificate authentication

With client certificate authentication, the REST client is prompted by the z/OS Connect EE server to supply a certificate that the server then validates and associates with a user ID in the user registry. The identity in the certificate must be mapped to a user ID in the user registry.

Third-party authentication

Third-party authentication means that the REST client authenticates with a third-party authentication server. This server generates an authentication token. The REST client might access the third-party authentication server through an intermediate server, typically an API gateway, for example, IBM API Connect®, as shown in [Figure 73 on page 350](#). The authentication token is then sent to the z/OS Connect EE server. z/OS Connect EE validates the token and the identity in the token is optionally mapped to a user ID in the user registry.

User registries

User registries store information about users and groups that can be used for authentication and authorization. z/OS Connect EE supports the following types of user registry.

Basic user registry

A simple file-based registry, where users and groups are defined in the `server.xml` configuration file. It is typically used in development environments.

Lightweight Directory Access Protocol (LDAP) user registry

Typically used in an environment where LDAP already stores the user IDs.

System Authorization Facility (SAF) registry

Typically used to store SAF user IDs on z/OS.

Using multiple user registries in the same z/OS Connect EE server is referred to as *federated user registries*. This configuration is useful when, for example, the user information is in two different LDAP servers, in two subtrees of the same LDAP server, or in both an LDAP server and a SAF registry. For more information, see [Federation of user registries in the WebSphere Application Server for z/OS Liberty documentation](#).

For more information about configuring the user registries, see [“User registries” on page 453](#).

Basic authentication

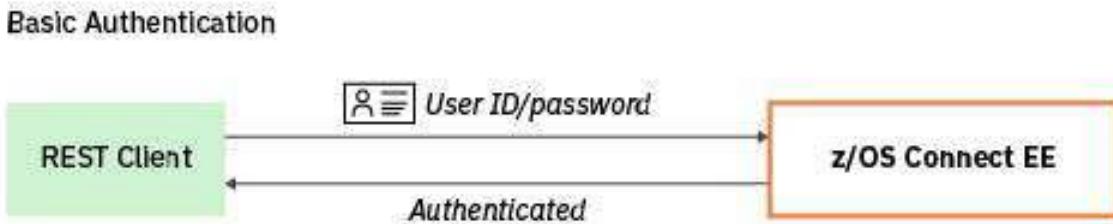


Figure 74. A representation of basic authentication.

Basic authentication is a simple authentication scheme that is built into the HTTP protocol. It requires the client to provide a user ID and password in the request. The user ID and password are encoded in base64 and sent in the HTTP Authorization header of the request. The z/OS Connect EE server validates the user ID and password against a configured user registry. The user ID is set as the authenticated user.

The following types of user ID and password are supported for basic authentication when z/OS Connect EE acts as an API provider:

- A user ID and password defined in a basic user registry.
- An LDAP distinguished name (or uid) and password defined in an LDAP user registry.
- A SAF user ID and password defined in the SAF registry on the same LPAR as the z/OS Connect EE server.

When basic authentication is used, the credentials are encoded, but are not encrypted. Therefore, it is typically used with HTTPS (TLS) to provide confidentiality.

By default, z/OS Connect EE uses client certificate authentication. You can use one of the following methods to implement basic authentication:

- Configure z/OS Connect EE to fail over to use basic authentication when the client certificate authentication does not succeed. For example, when the client does not send a certificate or when the client sends a certificate but the certificate is not mapped to a user ID in the user registry.

To configure fail over to basic authentication, add the following element to the `server.xml` configuration file:

```
<webAppSecurity allowFailOverToBasicAuth="true"/>
```

- Configure z/OS Connect EE to override the client certificate authentication default. However, this configuration applies globally, so this option is not suitable when any requests to the same z/OS Connect EE server require client certificate authentication. This option also provides improved performance. For more information, see [“Performance best practices for a z/OS Connect EE production system” on page 205](#).

To configure z/OS Connect EE to override the client certificate authentication default with basic authentication, add the following element to the `server.xml` configuration file:

```
<webAppSecurity overrideHttpAuthMethod="BASIC"/>
```

For more information about configuring basic authentication, see [“API provider basic authentication” on page 354](#).

Client certificate authentication



Figure 75. A representation of client certificate authentication.

Choose certificate-based client authentication to use information that is provided in the client's TLS certificate to map to an associated user ID. It also provides all of the normal benefits that are associated with a secure TLS connection. For more information about TLS, see [“API provider confidentiality and integrity” on page 336](#).

When TLS client authentication is configured, the client must provide its certificate to z/OS Connect EE for each HTTPS connection. z/OS Connect EE validates the chain of trust by checking that the client certificate issuer is in the truststore. This process is standard TLS behavior and if the client certificate is successfully validated, the connection can be established to the z/OS Connect EE server.

To authenticate to the z/OS Connect EE server, the client certificate must also be mapped to a user ID in the user registry. If the client certificate is successfully mapped to a user ID, then that user ID is set as the authenticated user.

- For the basic registry, the user identity is the common name (CN) from the distinguished name (DN) of the certificate. For more information about using client authentication with a basic registry, see [Basic certificate map mode](#) in the *WebSphere Application Server for z/OS Liberty* documentation.
- For an LDAP registry, the DN from the client certificate must be in the LDAP registry. For more information about using client authentication with LDAP, see [LDAP certificate map mode](#) in the *WebSphere Application Server for z/OS Liberty* documentation.
- For a SAF registry, a DIGTCERT profile is generated from the information in the certificate, such as the certificate's serial number and the issuer's distinguished name. The profile must be associated with a SAF user ID.

Client certificates can be associated with a RACF user ID in two ways:

1. Use the **RACDCERT MAP** command to define a certificate name filter, which is also called a user ID mapping. This command maps the certificate subject's distinguished name (DN) to a RACF user ID. Certificate name filtering supports generic filters that allow multiple certificates to be associated with a single RACF user ID.
2. Use the **RACDCERT ADD** command to add the certificate into RACF and specify the user ID it is to be associated with. This command is typically used only when REST clients connect to a z/OS Connect EE server via an intermediate server, such as an API Gateway, where there are only a few such servers and certificates to store and maintain in RACF.

For more information, see [RACDCERT ADD](#) (add certificate) and [RACDCERT MAP](#) (create mapping) in the *z/OS Security Server RACF Command Language Reference*.

z/OS Connect EE is configured for client certificate authentication with an SSL configuration. The SSL client authentication can be configured as required, or configured as supported.

- If TLS client authentication is required, then the client must provide a certificate that is trusted by the server for the handshake to succeed.
- If TLS client authentication is supported and the client provides a certificate, then that certificate must be trusted by the server. However, if the client does not provide a certificate, the TLS handshake continues by using TLS server authentication only.

For certificate-based client authentication, client authentication should be configured as required, rather than supported. If the client certificate is not mapped to a user ID in the user registry, then you can configure failover to basic authentication.

If required, you can configure multiple ports that have different SSL configurations.

Note: AT-TLS is not supported for client authentication between a client and the z/OS Connect EE server when z/OS Connect EE is acting as an API provider.

Third-party authentication

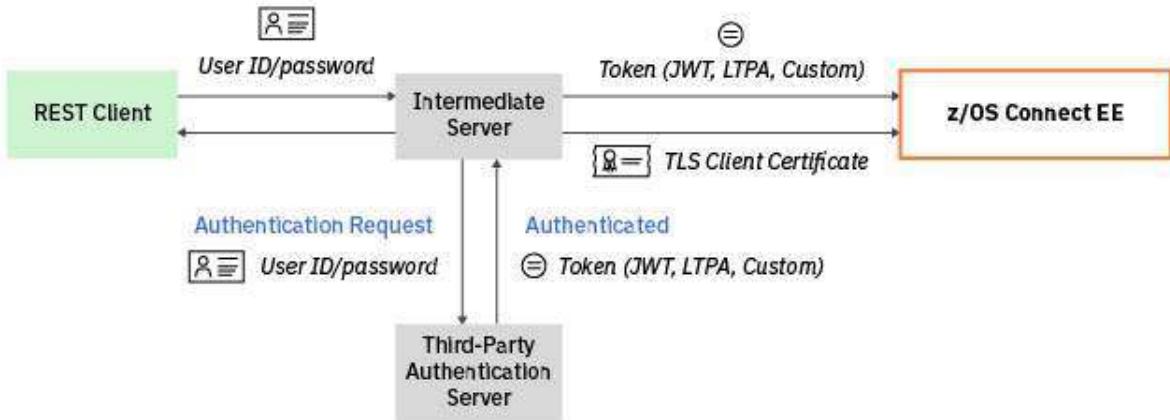


Figure 76. A representation of third-party authentication.

When third-party authentication is used, the REST client authenticates with a third-party authentication server. This server generates an authentication token. The REST client might access the third-party authentication server through an intermediate server, typically an API gateway, for example, IBM API Connect, as shown in [Figure 76 on page 353](#). The authentication token is then sent to the z/OS Connect EE server. z/OS Connect EE validates the token and the identity in the token is optionally mapped to a user ID in the user registry.

Trust between the third party and z/OS Connect EE can be established in different ways, including the use of a digital signature. Third-party authentication tokens can be used as part of a Single Sign-On (SSO) solution.

Client authentication can also be used in this model so that the client certificate of the intermediate server is used to establish the connection to the z/OS Connect EE server. However, the user identity in the token rather than the client certificate distinguished name is used to determine the authenticated identity for the request.

One of the advantages of using third-party authentication is that the z/OS Connect EE server does not see the client's password.

For more information about third-party authentication, see [“API provider third-party authentication” on page 360](#).

Caching authentication credentials

An authentication cache is provided to store a subject after successful authentication of a user to reduce the potential performance impact of creation of a subject. For more information, see [Configuring the authentication cache in Liberty in the WebSphere Application Server for z/OS Liberty](#).

Using a Trust Authentication Interceptor (TAI) to allow selected unauthenticated requests

Where z/OS Connect EE is configured for basic authentication, but selected requests do not present the required credential, a Trust Authentication Interceptor (TAI) can be developed, configured, and deployed with a z/OS Connect EE server to allow selected "unauthenticated" requests to be processed.

For more information about creating a TAI, see [Developing a custom TAI for Liberty and Web Container Application Security \(webAppSecurity\)](#) in the *WebSphere Application Server for z/OS Liberty* documentation.

API provider basic authentication

The steps required to configure basic authentication between a client and a z/OS Connect EE server depend on the type of user registry being used.

This task is applicable when z/OS Connect EE is used as an API provider.

Before you begin these tasks, you should be familiar with the information in [“API provider authentication and identification” on page 349](#).

When you have chosen an authentication method, follow the instructions in the appropriate task:

How to configure basic authentication with a basic user registry

Configure a z/OS Connect EE server to perform basic authentication with a basic user registry.

This task is applicable when z/OS Connect EE is used as an API provider.

Before you begin

- You should be familiar with the information in [“API provider authentication and identification” on page 349](#).
- You must complete the task [“How to configure a basic user registry” on page 453](#).
- You must have write access to the `server.xml` configuration file.

About this task

You configure the z/OS Connect EE server to require authentication, by setting the attribute `requireAuth="true"`. This task then configures the server to use basic authentication.

This task does not include information on how to configure the z/OS Connect EE server to use TLS. If the attribute `requireSecure` is set to true (the default), you must configure a TLS connection between the client and the z/OS Connect EE server, for example, by completing the task [“How to configure TLS with RACF key rings” on page 339](#).

Procedure

For more information about configuration elements, see [“Configuration elements” on page 753](#) in the *Reference* section.

1. Ensure that the server is configured to require authentication for the request.

This configuration can be set at different scopes:

- To require authentication globally for the server, set `requireAuth="true"` on the `zosconnect_zosConnectManager` element in the `server.xml` configuration file. For example,

```
<zosconnect_zosConnectManager requireAuth="true" ... />
```

- To require authentication for a specific API, which takes precedence over the global server setting, set `requireAuth="true"` on the `zosConnectAPI` element in the `server.xml` configuration file. For example,

```
<zosconnect_zosConnectAPIs>
  <zosConnectAPI name="Api1" requireAuth="true"/>
</zosconnect_zosConnectAPIs>
```

- To require authentication for a specific service, which takes precedence over the global server setting, set `requireAuth="true"` on the `service` element in the `server.xml` configuration file. For example,

```
<zosconnect_services>
  <service name="Service1" requireAuth="true"/>
</zosconnect_services>
```

2. Configure the server to use basic authentication.

`z/OS Connect EE` attempts to use a TLS client certificate for authentication, unless an alternative authentication mechanism is configured. Use one of the following methods to configure basic authentication:

- Configure fail-over to basic authentication, by adding the following element to the `server.xml` configuration file:

```
<webAppSecurity allowFailOverToBasicAuth="true"/>
```

- Configure basic authentication to override the client certificate authentication default, by adding the following element to the `server.xml` configuration file:

```
<webAppSecurity overrideHttpAuthMethod="BASIC"/>
```

3. Assign users and groups to the `zosConnectAccess` role.

Follow the instructions in task [“How to configure the `zosConnectAccess` role with a basic user registry” on page 377](#).

Results

The pre-defined set of users and groups that are defined in the basic user registry can be used to authenticate with the `z/OS Connect EE` server. Additionally, the basic user registry users and groups that are assigned to the `zosConnectAccess` role now have authorization to access `z/OS Connect EE`.

How to configure basic authentication with an LDAP user registry

Configure a `z/OS Connect EE` server to perform basic authentication with a Lightweight Directory Access Protocol (LDAP) user registry.

This task is applicable when `z/OS Connect EE` is used as an API provider.

Before you begin

- You should be familiar with the information in [“API provider authentication and identification” on page 349](#).
- You must complete the task [“How to configure an LDAP user registry” on page 455](#).
- You must have write access to the `server.xml` configuration file.

About this task

You configure the z/OS Connect EE server to require authentication, by setting the attribute `requireAuth="true"`. This task then configures the server to use basic authentication.

This task does not include information on how to configure the z/OS Connect EE server to use TLS. If the attribute `requireSecure` is set to `true` (the default), you must configure a TLS connection between the client and the z/OS Connect EE server, for example, by completing the task [“How to configure TLS with RACF key rings” on page 339](#).

Procedure

For more information about configuration elements, see [“Configuration elements” on page 753](#) in the *Reference* section.

1. Ensure that the server is configured to require authentication for the request.

This configuration can be set at different scopes:

- To require authentication globally for the server, set `requireAuth="true"` on the `zosconnect_zosConnectManager` element in the `server.xml` configuration file. For example,

```
<zosconnect_zosConnectManager requireAuth="true" ... />
```

- To require authentication for a specific API, which takes precedence over the global server setting, set `requireAuth="true"` on the `zosConnectAPI` element in the `server.xml` configuration file. For example,

```
<zosconnect_zosConnectAPIs>
  <zosConnectAPI name="Api1" requireAuth="true"/>
</zosconnect_zosConnectAPIs>
```

- To require authentication for a specific service, which takes precedence over the global server setting, set `requireAuth="true"` on the `service` element in the `server.xml` configuration file. For example,

```
<zosconnect_services>
  <service name="Service1" requireAuth="true"/>
</zosconnect_services>
```

2. Configure the server to use basic authentication.

z/OS Connect EE attempts to use a TLS client certificate for authentication, unless an alternative authentication mechanism is configured. Use one of the following methods to configure basic authentication:

- Configure fail-over to basic authentication, by adding the following element to the `server.xml` configuration file:

```
<webAppSecurity allowFailOverToBasicAuth="true"/>
```

- Configure basic authentication to override the client certificate authentication default, by adding the following element to the `server.xml` configuration file:

```
<webAppSecurity overrideHttpAuthMethod="BASIC"/>
```

3. Optional: Map the LDAP user ID received in the basic authentication header to a SAF user ID.

If you require a SAF user ID because the z/OS Connect EE server uses SAF for authentication, or the System of Record requires a SAF user ID, then you can map the authenticated LDAP user ID to a SAF user ID. For more information, see the task [“How to map an LDAP user ID to a RACF user ID” on page 357](#).

4. Assign users and groups to the `zosConnectAccess` role.

If you mapped the LDAP user ID to a SAF user ID, as described in step “3” on page 356, follow the instructions in task [“How to configure the `zosConnectAccess` role with a SAF user registry” on page](#)

380. Otherwise, follow the instructions in task “[How to configure the zosConnectAccess role with an LDAP user registry](#)” on page 378.

Results

Users and groups that match the filters defined in the `ldapRegistry` element can be used to authenticate with the z/OS Connect EE server. Additionally, the LDAP users and groups that are assigned to the `zosConnectAccess` role now have authorization to access z/OS Connect EE.

How to map an LDAP user ID to a RACF user ID

Configure RACF and a z/OS Connect EE server to map an LDAP user ID to a RACF user ID.

Perform this procedure if your z/OS Connect EE server is configured to perform authorization using RACF, or the System of Record (SoR) requires a RACF user ID, but the client authenticates with an LDAP user ID as its identity. For example, the client might use a basic authentication (HTTP Authorization) header.

This task is applicable when z/OS Connect EE is used as an API provider.

Before you begin

- You must have configured the z/OS Connect EE server to require authentication and to use basic authentication, by following the appropriate steps in the task “[How to configure basic authentication with an LDAP user registry](#)” on page 355.
- You must have configured the z/OS Connect EE server to access RACF, by completing the task “[How to activate and configure the SAF user registry](#)” on page 456. However, you do not need to specify the `safRegistry` element in the `server.xml` configuration file.
- If you specify both `safRegistry` and `ldapRegistry` elements in the `server.xml` configuration file, then you should also configure a `federatedRepository` element, so that you can specify the primary realm name. This ensures that the primary realm name used by the z/OS Connect EE server when it attempts to map the LDAP user ID to the RACF user ID, matches the `REGISTRY NAME` value you will specify on the **RACMAP** command. Only one security realm is supported. If you do not specify a primary realm name, the realm name from one of the existing user registries is used, but the same realm name might not be returned. For more information, see [Federation of user registries in the WebSphere Application Server for z/OS Liberty documentation](#).
- You need to know the distinguished name (DN) of the LDAP user ID to be mapped.
- You need to know the RACF user ID to which the LDAP user ID is to be mapped, and that RACF user ID must exist.
- You must have write access to the `server.xml` configuration file.

About this task

This task makes the following assumptions:

- The client is sending an LDAP user ID and password in the HTTP Authorization header of a request to a z/OS Connect EE server, to authenticate using basic authentication.
- RACF is being used as the security manager. If you are using an alternative External Security Manager, refer to the appropriate product documentation for the equivalent commands.

Procedure

1. Activate the RACF IDIDMAP class

Enter the following RACF command:

SETROPTS CLASSACT(IDIDMAP) RACLIST(IDIDMAP)

2. Create a mapping in RACF to associate the LDAP user ID to a RACF user ID.

For example, enter the following RACF command:

```
RACMAP MAP ID(EMPLOY1)
USERIDFILTER(NAME('cn=JeanLeclerc,ou=employees,o=ibm,c=fr'))
REGISTRY(NAME('SampleLdapIDSRealm')) WITHLABEL('LDAP Mapping EMPLOY1')
```

The command uses the following values:

- EMPLOY1 is the RACF user ID to which the LDAP user ID is to be mapped.
- cn=JeanLeclerc,ou=employees,o=ibm,c=fr is the distinguished name (DN) of the LDAP user ID to be mapped.
- SampleLdapIDSRealm is the value specified as the realm attribute of the ldapRegistry element in the server.xml configuration file or, if federated registries are being used, the federatedRegistry element. Alternatively, replace SampleLdapIDSRealm with * (the asterisk character) to match to any registry realm name.

3. Refresh the RACF IDIDMAP class.

For the changes to take effect, issue the following RACF command:

```
SETROPTS RACLIST(IDIDMAP) REFRESH
```

4. Check that the RACF mapping has been created.

Enter the following RACF command:

```
RACMAP QUERY USERIDFILTER(NAME('cn=JeanLeclerc,ou=employees,o=ibm,c=fr'))
REGISTRY(NAME('SampleLdapIDSRealm'))
```

The following screen capture shows the expected response:

```
RACMAP QUERY result. RACF user ID: EMPLOY1
```

5. Configure the server to call RACF to perform the mapping of the received LDAP user ID to the RACF user ID.

Add the following element to the server.xml configuration file:

```
<safCredentials mapDistributedIdentities="true"/>
```

For more information about the safCredentials element, see [Server configuration](#) in the *IBM WebSphere Application Server for z/OS Liberty* documentation.

6. Start, or restart the server if it was already running, to pick up the changes made to the RACF class profiles.

If you have linked to this task from another task, you might wish to complete the steps in that task before restarting the server.

Results

The LDAP user ID is now mapped to a RACF user ID. When this LDAP user ID is sent as the identity in a request, the z/OS Connect EE server will attempt to authenticate the user ID and map it to the RACF user ID, so that the RACF user ID can be used for authorization and when connecting to the System of Record (SoR).

How to configure basic authentication with a SAF user registry

Configure a z/OS Connect EE server to perform basic authentication with a SAF user registry.

This task is applicable when z/OS Connect EE is used as an API provider.

Before you begin

- You should be familiar with the information in “[API provider authentication and identification](#)” on page 349.

- You must complete the task “[How to activate and configure the SAF user registry](#)” on page 456 to configure the z/OS Connect EE server to use z/OS authorized services and a SAF user registry.
- You must have write access to the `server.xml` configuration file.

About this task

You configure the z/OS Connect EE server to require authentication, by setting the attribute `requireAuth="true"`. This task then configures the server to use basic authentication.

This task does not include information on how to configure the z/OS Connect EE server to use TLS. If the attribute `requireSecure` is set to true (the default), you must configure a TLS connection between the client and the z/OS Connect EE server, for example, by completing the task [“How to configure TLS with RACF key rings”](#) on page 339.

Procedure

For more information about configuration elements, see [“Configuration elements”](#) on page 753 in the *Reference* section.

1. Ensure that the server is configured to require authentication for the request.

This configuration can be set at different scopes:

- To require authentication globally for the server, set `requireAuth="true"` on the `zosconnect_zosConnectManager` element in the `server.xml` configuration file. For example,

```
<zosconnect_zosConnectManager requireAuth="true" ... />
```

- To require authentication for a specific API, which takes precedence over the global server setting, set `requireAuth="true"` on the `zosConnectAPI` element in the `server.xml` configuration file. For example,

```
<zosconnect_zosConnectAPIs>
  <zosConnectAPI name="Api1" requireAuth="true"/>
</zosconnect_zosConnectAPIs>
```

- To require authentication for a specific service, which takes precedence over the global server setting, set `requireAuth="true"` on the `service` element in the `server.xml` configuration file. For example,

```
<zosconnect_services>
  <service name="Service1" requireAuth="true"/>
</zosconnect_services>
```

2. Configure the server to use basic authentication.

z/OS Connect EE attempts to use a TLS client certificate for authentication, unless an alternative authentication mechanism is configured. Use one of the following methods to configure basic authentication:

- Configure fail-over to basic authentication, by adding the following element to the `server.xml` configuration file:

```
<webAppSecurity allowFailOverToBasicAuth="true" />
```

- Configure basic authentication to override the client certificate authentication default, by adding the following element to the `server.xml` configuration file:

```
<webAppSecurity overrideHttpAuthMethod="BASIC" />
```

3. Assign users and groups to the `zosConnectAccess` role.

Follow the instructions in task [“How to configure the `zosConnectAccess` role with a SAF user registry”](#) on page 380.

Results

User IDs and groups in the SAF user registry can be used to authenticate with the z/OS Connect EE server. Additionally, the SAF user IDs and groups that are assigned to the `zosConnectAccess` role now have authorization to access z/OS Connect EE.

API provider third-party authentication

Learn how z/OS Connect EE implements third-party authentication for clients that connect to a server.

Before you study this topic, you should be familiar with the information in [“API provider authentication and identification” on page 349](#).

When you plan to implement third-party authentication for clients that connect to a z/OS Connect EE server, consider the following methods:

- Authenticating with a JSON Web Token (JWT).
- Using the OpenID Connect Client feature.
- Authentication filters

When you use third-party authentication, the REST client authenticates with a third-party authentication server. This server generates an authentication token. The REST client might access the third-party authentication server through an intermediate server, typically an API gateway, for example, IBM API Connect. The authentication token is then sent to the z/OS Connect EE server, which validates the token. The identity in the token can also be mapped to a user ID in the user registry.

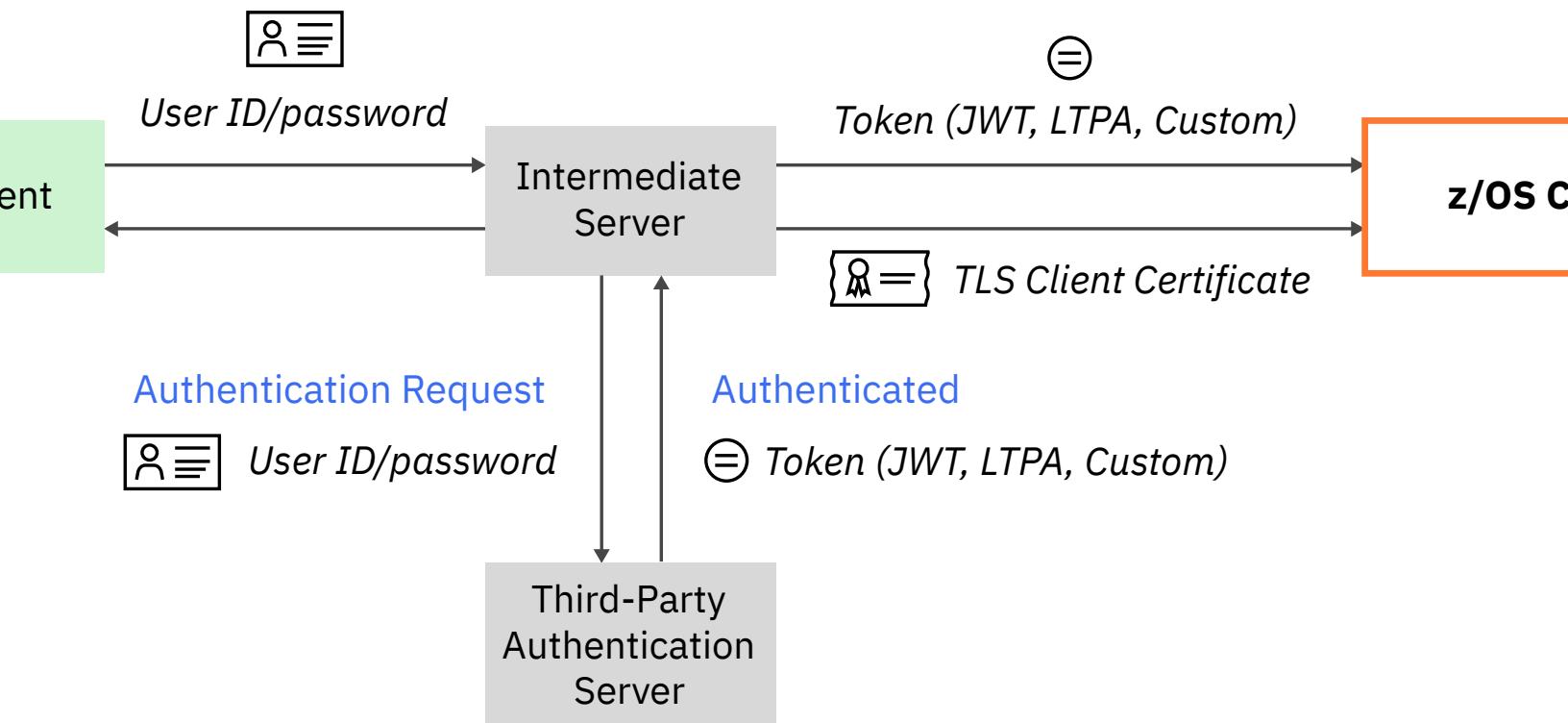


Figure 77. A representation of third-party authentication.

Trust between the third party and a z/OS Connect EE server can be established in different ways, including the use of a digital signature. Third-party authentication tokens can be used as part of a Single Sign-On (SSO) solution.

z/OS Connect EE supports the following types of authentication token:

JSON Web Token (JWT)

A JSON formatted token conveys claims, such as the identity used by a user login. As JSON is less verbose than XML, a JWT is more compact than a Security Assertion Markup Language (SAML) token. This efficiency makes JWT a good choice to be passed in REST API requests to z/OS Connect EE, and is why JWT authentication is becoming a standard for REST APIs.

For more information about using JWTs with z/OS Connect EE, see [“Authenticating with a JSON Web Token”](#) on page 361.

OAuth 2.0 access token

The OAuth 2.0 protocol facilitates the authorization of one site to access and use information that is related to the user's account on another site. You can configure z/OS Connect EE to function as an OAuth 2.0 protected resource server so that it remotely validates the access token to an OAuth authorization server by calling an OAuth 2.0 Token Introspection endpoint.

For more information about configuring Liberty z/OS to function as an OAuth 2.0 protected resource server, see [Configuring OAuth 2.0 protected resources in Liberty](#) in the *WebSphere Application Server for z/OS Liberty* documentation.

Security Assertion Markup Language (SAML) token

A SAML token is an XML-based token that is widely used for securing web services. It is typically used to comply with existing company standards that are based around SAML.

For more information about authenticating with a SAML token in Liberty z/OS, see [Configuring SAML Web Inbound Propagation in Liberty](#) in the *WebSphere Application Server for z/OS Liberty* documentation.

Lightweight Third-Party Authentication (LTPA) token

LTPA is a proprietary authentication technology that is used by Liberty z/OS and other IBM products. When multiple Liberty z/OS servers are configured to use LTPA, it is possible to enable SSO so that clients can reuse their login to access all the servers.

When LTPA is used, a token is created that contains the user information, an expiration time, and the signature of an LTPA key. The LTPA token passes between the client application and the Liberty server as a cookie when SSO is enabled. z/OS Connect EE supports the use of LTPA for authentication and SSO. However, it is more common to use an open authentication token such as a JWT.

For more information, see [Configuring LTPA in Liberty](#) in the *WebSphere Application Server for z/OS Liberty* documentation.



Warning: The default expiration time of an LTPA token is 2 hours. If the user is then re-authenticated, a new LTPA token is created. If you use the same default expiration time for multiple servers that were all started at the same time, the LTPA tokens will all expire at the same time, possibly causing a degradation in performance as the user is re-authenticated and a new token created. Consider setting expiration times so that when a token expires, this re-authentication process occurs at a suitable time for your environment. Also, it is not advisable to set a long expiry time as it could cause a security exposure. Also, it is not advisable to set a long expiry time as it could cause a security exposure.

Authenticating with a JSON Web Token

JSON Web Tokens are an open industry standard for securely representing claims between two parties. Claims are statements about an entity. For example, what identity was used by a user login, or any other type of claims as required by business processes, such as, the user is an administrator.

The claims in a JWT are encoded as a JSON object and are normally digitally signed with a Message Authentication Code (MAC) and if necessary, encrypted. The most common scenario for using a JWT is

authentication. When the user is logged in, each subsequent request includes the JWT, allowing the user to access services that are permitted with that token.

The following list shows some of the advantages of using JWTs with z/OS Connect EE:

- They are lightweight and easy to use by client applications such as mobile apps.
- They are self-contained, which means that a z/OS Connect EE server accepts the token directly and uses a claim from the token as the identity for making the API request.
- They can be symmetrically signed by a shared secret that uses the HMAC algorithm, or asymmetrically by using a private key.
- They have a built-in expiry mechanism.
- They can be extended to contain custom claims.
- They are widely adopted by different Single Sign-On solutions and well-known standards such as “OpenID Connect” on page 363.

Anatomy of a JWT

A JWT consists of three parts: a header, payload, and signature, as shown in [Figure 78 on page 362](#).

Encoded

```
eyJhbGciOiJSUzI1NiJ9.eyJpc3Mi0iJpZGciLCJzdWIiO
iJbjb1KZWtGvjbGVyYxvdT1lbXBsb31ZXMsbz1pYm0
sYz1mcIIsImF1ZCI6InVybjpteUVudGl0eSIsImV4cCI6M
TU0NDE5NjA0NSwiaWF0IjoxNTQ0MTg40DQ1fQ.a6DcawLi
6p87vW1Jr1VN10oAE6gAY10ZSRtL7Z1wCoavZsJCL4ZHjW
DxTfJu0WG0aP3Q4_cgphjLw7GCbc551kQSMe64nJRxl--
7ZIyEkhMpPHA_QHK7Udr9JS-SBJ4e0BCbJwk46d7D-qADd
XSzSDXmp125GwW6HDs4JqIguRDTNpH3Z3R_5HWhitpz2rY
Vn12XQ1VbihuqoeBBtEHKLjma0U0J3sY5Hq2stjLrW5HuL
uTfBbuJsWv1SJMcAie0Ai6d1pEUH2cZ-U6AhHjFi3c_eKV
5kwCX811vEIZC8AGwt0E2VDNmuz_NDOTy9JUYDmL63EgzJ
NaFPV1y29WdA
```

Decoded

Header

```
{  
  "alg" : "RS256"  
}
```

Payload

```
{  
  "iss" : "idg",  
  "sub" : "cn=JeanLeclerc,ou=employees,o=ibm,c=fr",  
  "aud" : "urn:myEntity",  
  "exp" : "1544196045",  
  "iat" : "1544188845",  
}
```

Verify Signature

```
RSASHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  Enter Public key or Certificate,  
  Enter Private key  
)
```

Figure 78. Structure of a JSON Web Token (JWT).

The *header* typically consists of two parts: the type of token, which is JWT, and the hashing algorithm that is used to sign the JWT, such as HMAC SHA256 or RSA SHA256. The header is Base64Url encoded to form the first part of the JWT.

The *payload* contains the claims. A set of predefined claims is provided. For example, *iss* (issuer), *sub* (subject), *aud* (audience), and *exp* (expiration time). These claims are not mandatory but should be

included to provide a set of useful, interoperable claims. The payload can also include other attributes that define custom claims such as employee role. Typically, the subject claim is used to create the OpenID Connect user subject. However, z/OS Connect EE can be configured to use an alternative claim, if required. The payload is Base64Url encoded to form the second part of the JWT.

To create the *signature*, the encoded header, encoded payload, and a secret are signed by using the signature algorithm that is specified in the header `alg` parameter. The signature is used to verify the issuer of the JWT and to ensure the integrity of the message.

z/OS Connect EE does not support JSON Web Encryption (JWE), but you can use HTTPS to encrypt the complete message (including the JWT).

OpenID Connect

You can enable open secure APIs by using standards such as OAuth, an open standard for authorization, and OpenID Connect, an identity protocol that is built on the OAuth 2.0 protocol. OpenID Connect enables client applications to rely on authentication that is performed by an OpenID Connect Provider. The client application retrieves an ID token in the form of a JWT from the OpenID Connect Provider. The token is used to access a resource on behalf of the user. See [Figure 79 on page 364](#)

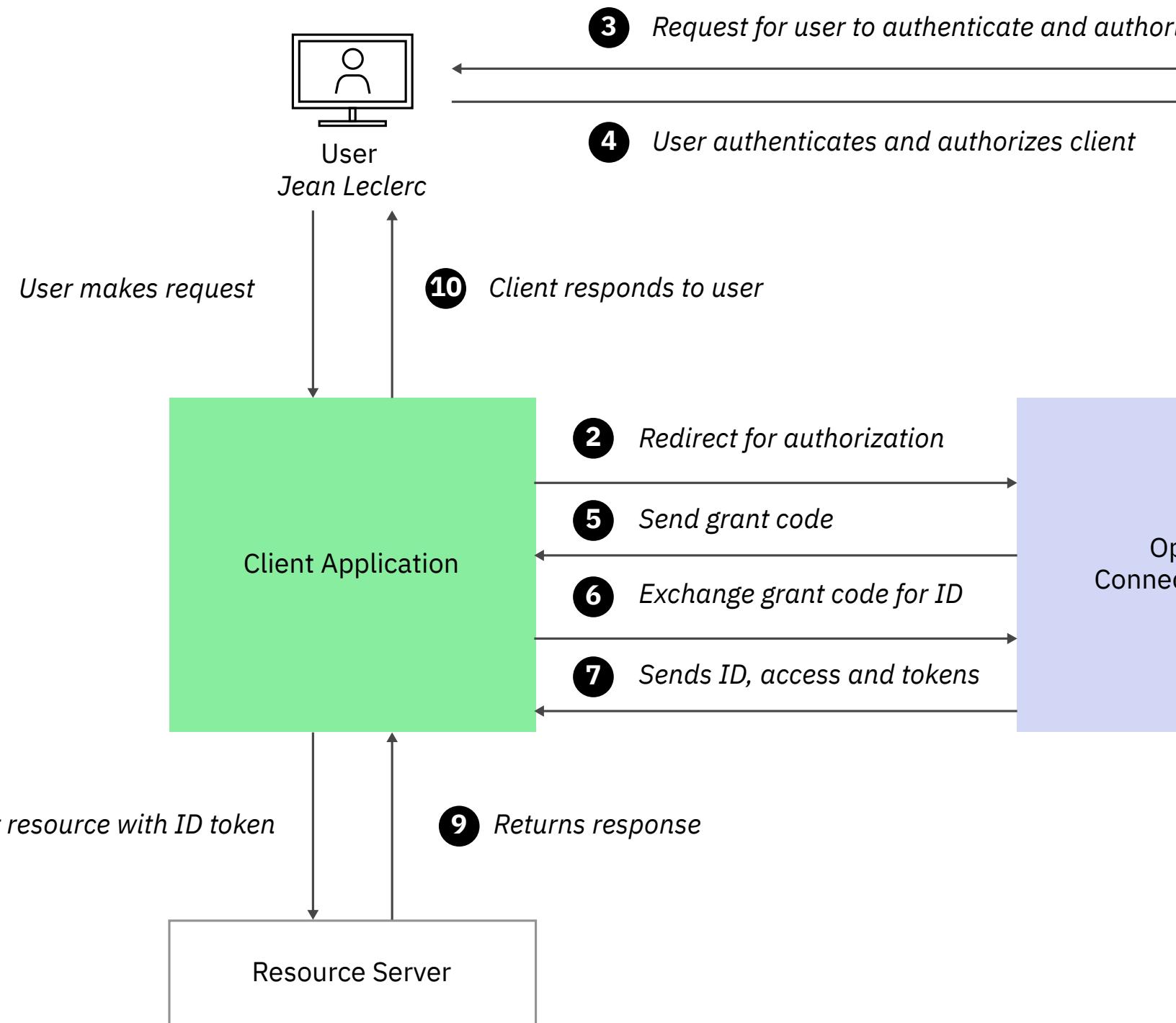


Figure 79. Flow of OpenID Connect tokens

The diagram in Figure 79 on page 364 shows the following steps:

1. The user makes a request to the client application.
2. The client application redirects the request to the OpenID Provider (OP).
3. The OP sends an authentication and authorization request to the user.
4. The user authenticates and authorizes the client application to access the resource.
5. The OP sends a grant code to the client application.

6. The client application sends a request to the OP to exchange the grant code for an ID token in the form of a JWT, access token, and refresh token.
7. The OP sends the ID token (JWT), access token and refresh token to the client application.
8. The client application makes the request to the Resource Server with the JWT, which is used for authenticating the user and authorizing access to the resource.
9. The response is sent from the Resource Server to the client application.
10. The response is sent from the client application to the user.

z/OS Connect EE supports OpenID Connect 1.0 as a Resource server that can accept a JWT as an authentication token.

Liberty z/OS supports OpenID Connect 1.0 and can play a role as an OpenID Connect Client, Provider, or Resource Server. For more information about configuring support for OpenID Connect with Liberty, see [Using OpenID Connect in the WebSphere Application Server for z/OS Liberty documentation](#).

JWT authentication methods

z/OS Connect EE supports the following methods for processing JWTs:

OpenID Connect Client feature

You can configure z/OS Connect EE to use the Liberty `openidConnectClient-1.0` feature to accept a JWT as an authentication token. For more information about configuring the OpenID Connect Client feature with z/OS Connect EE, see [“Using the OpenID Connect Client feature” on page 365](#).

Note: This is the preferred method for configuring JWT authentication with z/OS Connect EE.

Trust Association Interceptor (TAI)

You can configure z/OS Connect EE to integrate with a third-party security service by using a Trust Association Interceptor (TAI). The TAI can inspect the HTTP request from the third-party security server for a specific security token, such as a JWT. For more information on using a TAI with Liberty z/OS, see [Configuring TAI for Liberty in the WebSphere Application Server for z/OS Liberty documentation](#).

Java Authentication Service Provider Interface for Containers (JASPI)

JASPI is a Java EE standard service provider API that enables the implementation of authentication mechanisms into Java EE Web Applications. For more information, see [Configuring a Java Authentication SPI for Containers \(JASPI\) User Feature in the WebSphere Application Server for z/OS Liberty documentation](#).

Using the OpenID Connect Client feature

To configure a z/OS Connect EE server to accept a JWT as an authentication token, enable the `openidConnectClient-1.0` feature and set the `inboundPropagation` attribute of the `openidConnectClient` element to `required`. You can also specify additional JWT configurations, such as authentication filters and claim-to-subject mapping.

Any trusted party in possession of a JWT can use that token to get access to the APIs hosted by z/OS Connect EE. The z/OS Connect EE server validates the JWT and creates the authenticated subject from the token. To be accepted as an authentication token, the JWT must contain `iss`, `sub`, and `exp` claims and be signed with the RS256 or HS256 algorithm.

You can configure z/OS Connect EE to accept a JWT as an authentication token either as part of an OpenID Connect flow, or from any intermediate server, such as an API Gateway.

A JWT might contain privacy-sensitive information, so take precautions to prevent disclosure of this information to unintended parties. For example, use Transport Layer Security (TLS) when transporting the token. Extra trust can be established by accepting a JWT only as an authentication token across a TLS mutually authenticated connection.

For more information, see [“How to configure JWT authentication” on page 366](#).

Authentication filters

If JWT authentication is required only for a subset of requests to the z/OS Connect EE server, then authentication filters can be used to restrict which requests the JWT authentication applies to. Authentication filters specify conditions that are matched against the HTTP request and are configured by using an `authFilter` element in the `server.xml` configuration file. This `authFilter` element can be configured as a subelement of the `openidConnectClient` element, or specified as a separate element referenced by the `authFilterRef` attribute of the `openidConnectClient` element.

By using authentication filters, you can have different security policies for different requests. For example, you can put a filter on the `requestUrl` so that the `openidConnectClient` element applies only to requests that match that URL. Another example is to put a filter on the `remoteAddress` element so that different certificates can validate the signature of the JWT depending on the server that sends the request.

If you use z/OS Connect EE V3.0.17 or later, authentication filters can also match on request headers. For example, to configure an authentication filter to match HTTP requests that send the HTTP Authorization header with a Bearer token, add the following element to the `server.xml` configuration file, either as a subelement, or referenced element, of the `openidConnectClient` element :

```
<authFilter id="JwtAuthFilter">
    <requestHeader id="authHeader" name="Authorization" value="Bearer" matchType="contains"/>
</authFilter>
```

To ensure that any requests containing a Basic token can fail over to use basic authentication, configure the following element:

```
<webAppSecurity allowFailoverToBasicAuth="true"/>
```

For more information, see [Authentication Filter \(authFilter\)](#) and [Authentication Filters](#) in the *WebSphere Application Server for z/OS Liberty* documentation.

How to configure JWT authentication

Configure a z/OS Connect EE server to perform authentication using a JSON Web Token (JWT).

This task is applicable when using z/OS Connect EE as an API provider.

Before you begin

- You should be familiar with the information in [“API provider third-party authentication” on page 360](#).
- You must have an X.509 certificate that contains the public key of the private key that signed the JWT. This key will be used to validate the JWT signature. Consult the security administrator of the JWT issuer for how to obtain a suitable X.509 certificate.
- You need to know the claims that are present in the JWT.
- You must have completed the task [“How to activate and configure the SAF user registry” on page 456](#) to configure the z/OS Connect EE server to use z/OS authorized services and a SAF user registry.
- You must have write access to the `server.xml` configuration file.

About this task

Configure a z/OS Connect EE server to perform JWT authentication and use the identity in the JWT to authorize access to z/OS Connect EE. This configuration uses the OpenID Connect Client feature of WebSphere Application Server Liberty Profile, to accept the JWT as an authentication token.

This task makes the following assumptions:

- The JWT is sent to z/OS Connect EE in an HTTP Authorization request header field as a Bearer token.
- The RS256 algorithm is used to sign the JWT.

- RACF is used for authorizing access to z/OS Connect EE. The identity in the JWT claims may be a RACF user ID, or may be an LDAP user ID that has been mapped to a RACF user ID.

You configure the z/OS Connect EE server to require authentication, by setting the attribute `requireAuth="true"`. This setting also enables the authorization check to ensure that the authenticated user has authority to access z/OS Connect EE, so you assign RACF user registry users to the `zosConnectAccess` role.

An example JWT is shown in [Figure 80 on page 367](#). For a full description of a JWT, see [“Anatomy of a JWT” on page 362](#).

```
{
  "alg": "RS256" },
{
  "iss": "idg",
  "sub": "EMPLOY1",
  "aud": "myEntity",
  "exp": 1496230040,
  "iat": 1496229740 }.
RSASHA256signature....
```

Figure 80. A sample JWT, showing the header, payload and signature

The following values are used:

- The `header` contains an `alg` (algorithm) with the value RS256, which is the hashing algorithm that was used to sign the JWT. RS256 is RSA Signature with SHA-256.
- The `iss` (issuer) claim, `idg` identifies the principal that issued the JWT. In this example, the JWT was issued by IBM DataPower®, which uses `idg` as its default issuer value.
- The `sub` (subject) claim, `EMPLOY1` is an identity. If the identity is a RACF user ID, it can be used directly for authorization control. If the sub claim does not contain a RACF user ID, it is possible to map the identity to a RACF user ID and then use the mapped user ID for authorization control.
- The `aud` (audience) claim, `myEntity` identifies the intended recipient of the JWT. The `aud` claim is optional. It can be used to identify a specific z/OS Connect EE server, a target application, a commercial entity, or any other entity defined by business processes.
- The `exp` (expiration time) claim identifies the expiration time on or after which the JWT must not be accepted for processing. This is expressed as a JSON numeric value representing the number of seconds from 1970-01-01T00:00:00Z UTC until the specified UTC date/time, ignoring leap seconds.
- The `iat` (issued at) claim identifies the time at which the JWT was issued. This is expressed as a JSON numeric value representing the number of seconds from 1970-01-01T00:00:00Z UTC until the specified UTC date/time, ignoring leap seconds.
- The `signature` (not shown) is calculated using the header and the payload. The signature certifies that only the party holding the private key is the one that created and signed the JWT, and it also verifies that the claims have not been tampered with.

Note:

1. If a JWT contains a `jti` (JWT ID) that is identical to a JWT previously used for authentication with a z/OS Connect EE server, the request is considered to be a replay attack. A `jti` is an optional claim.
2. If the JWT has been issued by a JWT provider which supports JWK (JSON Web Key) or has been signed using the HMAC-SHA256 algorithm, then some steps in this procedure must be modified. For more information, see [“Alternative configuration when using JWK or the HS256 algorithm” on page 371](#).

Procedure

1. Add the `openidConnectClient-1.0` Liberty feature to the `featureManager` element in the `server.xml` configuration file.

For example,

```
<featureManager>
  ...
  <feature>openidConnectClient-1.0</feature>
</featureManager>
```

2. Configure the server to use JWT authentication.

Add the `openidConnectClient` element to the `server.xml` configuration file. For example:

```
<openidConnectClient  
    id="RS"  
    clientId="RS-JWT-ZCEE"  
    inboundPropagation="required"  
    signatureAlgorithm="RS256"  
    trustStoreRef="JWTTrustStore"  
    trustAliasName="JWTTrustCert"  
    userIdentifier="sub"  
    mapIdentityToRegistryUser="true"  
    issuerIdentifier="idg"  
    disableLtpaCookie="true"  
    audiences="myEntity"  
/>
```

This element uses the following values:

- `id` and `clientId` are element identifiers.
- `inboundPropagation` is set to `required` to allow z/OS Connect EE to use the received JWT as an authentication token.
- `signatureAlgorithm` specifies the algorithm to be used to verify the JWT signature. z/OS Connect EE supports RS256 (asymmetric algorithm) and HS256 (symmetric algorithm).
- `trustStoreRef` specifies the name in the `id` attribute of the `keyStore` element that defines the location of the validating certificate.
- `trustAliasName` is the label or alias of the certificate to be used for signature validation.
- `userIdentifier` indicates the claim to use to create the user subject. This identity is then used as the authenticated user ID. The default claim to use for the user subject is the `sub` claim.
- `mapIdentityToRegistryUser` indicates whether to map the retrieved identity to a user ID in the user registry. The retrieved identity is the value of the claim that is specified on the `userIdentifier` attribute. This example sets `mapIdentityToRegistryUser="true"` to lookup the RACF user ID, EMPLOY1, from the `sub` claim in the RACF user registry.
- `issuerIdentifier` defines the expected issuer (`iss`). This must match the `iss` claim value in the JWT. If the z/OS Connect EE server receives JWTs with different `iss` claim values, this value can be a comma separated list of expected `iss` values.
- `disableLtpaCookie` indicates whether an LTPA token should be created. We do not want an LTPA token to be created so we set `disableLtpaCookie="true"`.
- `audiences` defines a comma-separated list of target audiences. The `aud` (audience) claim in the JWT must match one of the defined audiences.

For more information about other `openidConnectClient` attributes, see [OpenID Connect Client \(openidConnectClient\)](#) in the *WebSphere Application Server for z/OS Liberty* documentation.

3. Configure a truststore that is used to validate the JWT signature.

In this example, a private key is used to sign the JWT, so the X.509 certificate containing the public key must be stored in the z/OS Connect EE server's truststore.

- a) Add a `keyStore` element to the `server.xml` configuration file for the truststore.

The `id` attribute value of the `keyStore` element must match the value specified on the `trustStoreRef` attribute of the `openidConnectClient` element. The `keyStore` can be a Java KeyStore (JKS) or a RACF key ring.

For example, to reference a JKS truststore:

```
<keyStore id="JWTTrustStore"  
    location="${server.config.dir}/resources/security/myTrustStore.jks"  
    password="myPassword"  
    readOnly="true"  
    type="JKS" />
```

Or, to reference a RACF truststore:

```
<keyStore id="JWTTrustStore"
  fileBased="false"
  location="safkeyring:///myKeyRing"
  password="myPassword"
  readOnly="true"
  type="JCERACFKS" />
```

For more information about the keyStore element see [Server configuration](#) in the *IBM WebSphere Application Server for z/OS Liberty* documentation.

- b) Add the X.509 certificate that contains the public key required to validate the JWT signature into the truststore as a trusted certificate.

If you use a RACF key ring, connect the X.509 certificate that contains the public key to the RACF key ring with a usage of CERTAUTH.

4. Optional: If JWT authentication is required only for a subset of requests to the z/OS Connect EE server, the openidConnectClient element can include an authFilter subelement that represents conditions that are matched against the HTTP request.

If an openidConnectClient element is configured, then by default, all requests will be authenticated using that configuration. If authentication filters are defined, then only requests which meet the condition defined in the authentication filter will be authenticated using that element. For example, if JWT authentication is required for all requests except those using the z/OS Connect EE RESTful administration interface, the following authentication filter can be configured as a subelement of the openidConnectClient element used:

```
<authFilter><requestUrl matchType="notContain" urlPattern="/zosConnect/" /></authFilter>
```

For more information about authentication filters see [Authentication Filter \(authFilter\)](#) in the *WebSphere Application Server for z/OS Liberty* documentation.

For more information about configuring the OpenID Connect Client feature with Liberty z/OS, see [Configuring JSON Web Token authentication for OpenID Connect](#).

5. Optional: If the identity in the JWT is not a RACF user ID, it can be mapped to a different identity in the z/OS Connect EE server's SAF RACF user registry by using a distributed identity filter. Follow the steps in [“Mapping a distributed identity claim to a SAF user ID” on page 370](#) on how to do this.

6. Ensure that the server is configured to require authentication for the request.

This configuration can be set at various scopes:

- To require authentication globally for the server, set `requireAuth="true"` on the `zosconnect_zosConnectManager` element in the `server.xml` configuration file. For example:

```
<zosconnect_zosConnectManager
  requireAuth="true" ... />
```

- To require authentication for a specific API, which takes precedence over the global server setting, set `requireAuth="true"` on the `zosConnectAPI` element in the `server.xml` configuration file. For example:

```
<zosconnect_zosConnectAPIs>
  <zosConnectAPI name="Api1" requireAuth="true" />
</zosconnect_zosConnectAPIs>
```

- To require authentication for a specific service, which takes precedence over the global server setting, set `requireAuth="true"` on the `service` element in the `server.xml` configuration file. For example:

```
<zosconnect_services>
  <service name="Service1" requireAuth="true" />
</zosconnect_services>
```

For more information about these elements, see [“Configuration elements” on page 753](#) in the *Reference* section.

7. Assign the authenticated user ID associated with the JWT to the `zosConnectAccess` role to authorize it to access z/OS Connect EE.
Follow the steps in task “[How to configure the `zosConnectAccess` role with a SAF user registry](#)” on page 380.

Results

A JWT is used to authenticate an API request to a z/OS Connect EE server. The identity in the JWT is used for authorization control.

What to do next

You might be interested in configuring more specific authorization. See the task “[How to configure authorization levels with a SAF user registry](#)” on page 389.

Mapping a distributed identity claim to a SAF user ID

Follow these steps to create a distributed identity filter if the identity in the JWT is not a RACF user ID but is to be mapped to a user ID in the z/OS Connect EE server's SAF RACF user registry.

For more information about the rules for mapping JWT Claims to authentication subjects see [Configuring JSON Web Token authentication for OpenID Connect](#) in the *IBM WebSphere Application Server for z/OS Liberty* documentation.

Procedure

1. Set the attribute `mapIdentityToRegistryUser="false"` on the `openidConnectClient` element in the `server.xml` configuration file.
For more information about the `openidConnectClient` element, see the [Server configuration](#) section in the *IBM WebSphere Application Server for z/OS Liberty* documentation.
2. Optional: Specify a realm name on the `openidConnectClient` element by setting the `realmName` attribute.
3. Set the attribute `mapDistributedIdentities="true"` on the `safCredentials` element in the `server.xml` configuration file.

For example,

```
<safCredentials mapDistributedIdentities="true"/>
```

For more information about the `safCredentials` element, see the [Server configuration](#) section in the *IBM WebSphere Application Server for z/OS Liberty* documentation.

4. Activate the RACF IDIDMAP class.

Enter the following RACF command:

SETROPTS CLASSACT(IDIDMAP) RACLIST(IDIDMAP)

5. Define a distributed identity filter in RACF to map the distributed user ID from the claim to a RACF user ID.

For example, enter the following command:

```
RACMAP ID(EMPLOY1) MAP  
USERDIDFILTER(NAME('cn=JeanLeclerc,ou=employees,o=ibm,c=fr'))  
REGISTRY(NAME('*')) WITHLABEL('Test Mapping EMPLOY1')
```

In this example, the following values are used:

- `EMPLOY1` is the RACF user ID to which the distributed user ID from the JWT identity claim is to be mapped.
- `cn=JeanLeclerc,ou=employees,o=ibm,c=fr` is the distributed user ID from the JWT identity claim to be mapped.

- REGISTRY(NAME('*)) will match any registry realm name. Alternatively, to match a specific realm, replace the * with the value specified on the `realmName` attribute of the `openidConnectClient` element.

For more information about the **RACMAP** command, see [RACMAP \(Create, delete, list, or query a distributed identity filter\)](#) in the *z/OS Security Server RACF Command Language Reference*.

Note: This example shows a one-to-one mapping. z/OS Identity Propagation also supports many-to-one mappings. For example, you could use a many-to-one mapping to map all employees to the same RACF user ID. For more information about configuring distributed identity filters, see [Configuring distributed identity filters in z/OS security](#) in the *WebSphere Application Server for z/OS Liberty* documentation.

6. Refresh the RACF IDIDMAP class.

For the changes to take effect. enter the following RACF command:

SETROPTS RACLIST(IDIDMAP) REFRESH

7. Return to step “6” on page 369 in the original procedure to complete the main task.

Alternative configuration when using JWK or the HS256 algorithm

If the JWT has been issued by a JWT provider which supports JWK (JSON Web Key) or has been signed using the HMAC-SHA256 algorithm, then a truststore is not required. However, you must specify different attributes on the `openidConnectClient` element.

Procedure

1. If the public key is retrieved from a JWK endpoint, you specify the JWK endpoint URL on the `jwkEndpointUrl` attribute.
2. If the JWT is signed by using a shared secret key with the HMAC-SHA256 algorithm, you define the shared secret key on the `sharedKey` or `clientSecret` attributes.

API provider client certificate authentication

Configure a z/OS Connect EE server to perform authentication using the identity from a TLS client certificate.

This task is applicable when z/OS Connect EE is used as an API provider.

Before you begin this task, you should be familiar with the information in [“API provider authentication and identification” on page 349](#).

Follow the instructions in the task to configure client certificate authentication.

How to configure client certificate authentication with RACF

Configure a z/OS Connect EE server to perform authentication of the identity in a TLS client certificate, mapping the certificate to a RACF user ID, and then granting that user ID authority to access z/OS Connect EE resources.

This task is applicable when z/OS Connect EE is used as an API provider.

Before you begin

- You should be familiar with the information in [“API provider authentication and identification” on page 349](#).
- You must have completed the task [“How to activate and configure the SAF user registry” on page 456](#) to configure the z/OS Connect EE server to use z/OS authorized services and a SAF user registry.
- You must have configured a TLS connection between the REST client and the z/OS Connect EE server with TLS client authentication enabled. For example, by completing the steps in the task [“How to configure TLS with RACF key rings” on page 339](#).

- You need to know the subject value of the client certificate to be mapped.
- You need to know the user ID to which the TLS client certificate will be mapped, and this user ID must exist and have an OMVS segment.
- You must have authorization to issue the RACDCERT MAP command. For more information about the RACDCERT commands and the authorizations that are required, see [RACDCERT \(Manage RACF digital certificates\)](#) in the *z/OS Security Server RACF Command Language Reference*.
- You must have write access to the `server.xml` configuration file.

About this task

This task assumes that RACF is used as security manager. If you are using an alternative External Security Manager, refer to the appropriate product documentation for the equivalent commands.

You use RACF certificate name filtering, also called user ID mapping, to map the TLS client certificate to a RACF user ID. You then configure the z/OS Connect EE server to require authentication, by setting the attribute `requireAuth="true"`.

During authentication, the z/OS Connect EE server will call RACF to perform the mapping resulting in the mapped RACF user ID being the authenticated user ID.

Procedure

1. Activate the RACF DIGTNMAP class to allow certificate name filters to be created or changed.

Enter the following RACF command:

```
SETROPTS CLASSACT(DIGTNMAP) RACLST(DIGTNMAP)
```

2. Map the TLS client certificate to a RACF user ID.

Enter the following command to use RACF certificate name filtering to map the client certificate to a RACF user ID.

```
RACDCERT MAP ID(EMPLOY1) SDNFILTER('CN=myClient.host.com.O=IBM.C=US')  
WITHLABEL('ClientCertEMPLOY1')
```

The command uses the following values:

- EMPLOY1 is the RACF user ID to which the client certificate is to be mapped.
- CN=myClient.host.com.O=IBM.C=US is the subject distinguished name filter which corresponds to the client certificate subject's distinguished name value of CN=myClient.host.com, O=IBM, C=US. The syntax of the SDNFILTER is significant, use periods to separate the components of the distinguished name and remove any spaces between DN components.
- ClientCertEMPLOY1 is a label for the mapping.

For the full syntax of the RACDCERT MAP command, see [RACDCERT MAP \(Create mapping\)](#) in the *z/OS Security Server RACF Command Language Reference*.

3. Refresh the DIGTNMAP RACF class.

For the changes to take effect, enter the following RACF command:

```
SETROPTS RACLST(DIGTNMAP) REFRESH
```

4. Ensure that the server is configured to require authentication for the request.

This can be set at various scopes:

- To require authentication globally for the server, set `requireAuth="true"` on the `zosconnect_zosConnectManager` element in the `server.xml` configuration file. For example,

```
<zosconnect_zosConnectManager requireAuth="true" ... />
```

- To require authentication for a specific API, which takes precedence over the global server setting, set `requireAuth="true"` on the `zosConnectAPI` element in the `server.xml` configuration file. For example,

```
<zosconnect_zosConnectAPIs>
  <zosConnectAPI name="Api1"
    requireAuth="true"/>
</zosconnect_zosConnectAPIs>
```

- To require authentication for a specific service, which takes precedence over the global server setting, set `requireAuth="true"` on the `service` element in the `server.xml` configuration file. For example,

```
<zosconnect_services>
  <service name="Service1"
    requireAuth="true"/>
</zosconnect_services>
```

For more information about these elements see [“Configuration elements” on page 753](#) in the *Reference* section.

5. Assign the mapped RACF user ID to the `zosConnectAccess` role.

Follow the instructions in the task [“How to configure the `zosConnectAccess` role with a SAF user registry” on page 380](#).

6. Ensure the Liberty profile angel process is running.

To use z/OS authorized services for SAF authentication and authorization, the Liberty profile angel process must be running for the server to connect to. In one of the prerequisite tasks listed in the “Before you begin” section, you created a started task to run the Liberty angel process and granted permission for the z/OS Connect EE server to access it.

To start the angel process, start the associated started task. Enter the SDSF command,

/S BAQZANL

For more information about starting the angel process and checking that it started successfully, see [Start the Angel process as a started task](#) in *Configuring the Liberty Angel process and z/OS authorized services*.

7. Start, or restart the server if it was already running, to pick up the changes made to the RACF class profiles.

Results

The TLS client certificate is mapped to a RACF user ID, and is authorized to access z/OS Connect EE.

What to do next

You might now be interested in configuring more specific authorization. See the task [“How to configure authorization levels with a SAF user registry” on page 389](#).

API provider authorization

Learn how z/OS Connect EE authorizes both access to z/OS Connect EE and the actions that can be performed on APIs or services.

You should be familiar with the information in [“Overview of z/OS Connect EE security” on page 331](#) and [“API provider authentication and identification” on page 349](#).

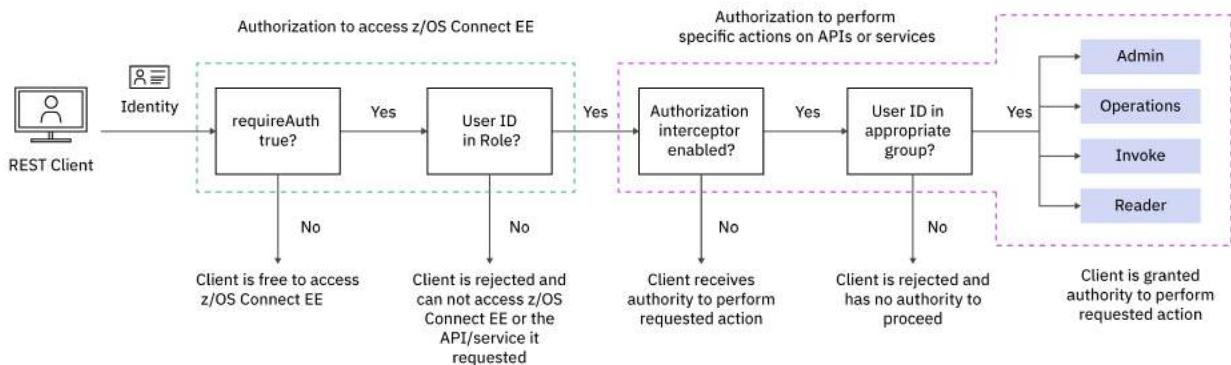
z/OS Connect EE authorization occurs after authentication completes successfully. The following checks are performed:

- Authorization to access z/OS Connect EE. The server checks the `requireAuth` attribute value set in the `server.xml` configuration file. By default, this value is set to `true`. When `true`, an authorization check

is performed to ensure that the authenticated user ID is assigned to the `zosConnectAccess` role. This behavior is described in “[Authorization to access z/OS Connect EE](#)” on page 374.

- Authorization to perform specific actions on APIs or services. z/OS Connect EE checks whether the authorization interceptor is enabled. If so, then it checks whether the authenticated user ID is a member of a user registry group, and whether that group is assigned to the appropriate authorization level. This behavior is described in “[Authorization to perform specific actions on APIs or services](#)” on page 374.

[Figure 81 on page 374](#) shows the z/OS Connect EE authorization flow.



[Figure 81. Authorization flow](#)

Authorization to access z/OS Connect EE

The first authorization check that is performed by the z/OS Connect EE server is to control whether the authenticated user ID is authorized to access z/OS Connect EE. This check occurs when the `server.xml` configuration file attribute `requireAuth` is set to `true`. The attribute `requireAuth` can be set globally for all APIs and services in the z/OS Connect EE server, or for individual APIs or services. The `requireAuth` attribute is primarily used to control authentication, see “[API provider authentication and identification](#)” on page 349, but is also used for this authorization check.

This authorization is controlled by the `zosConnectAccess` role, and access is granted by assigning user IDs associated with requests to that role. The request user ID must be defined in the z/OS Connect EE server's user registry. If a distributed ID is mapped to a user registry user ID, then the authorization check is performed against the mapped user ID.

There are two methods of assigning user IDs to the `zosConnectAccess` role:

- The `authorization-roles` element defined in the `server.xml` configuration file. You can either assign individual users and groups to the role, or you can authorize all authenticated users to the role by using the `<special-subject type="ALL_AUTHENTICATED_USERS"/>`. For more information, see “[How to configure the zosConnectAccess role with a basic user registry](#)” on page 377 and “[How to configure the zosConnectAccess role with an LDAP user registry](#)” on page 378.
- The SAF EJBROLE class profile `profilePrefix.zos.connect.access.roles.zosConnectAccess`. This can be defined with users and groups given READ access to it. For more information, see “[How to configure the zosConnectAccess role with a SAF user registry](#)” on page 380.

Authorization to perform specific actions on APIs or services

You can use a more granular level of authorization to control which users can perform specific actions on APIs or services. This authorization is implemented by the z/OS Connect EE *authorization interceptor*.

The authorization interceptor defines four authorization levels. A comma-separated list of user registry groups can be assigned to each level. Both the authorization interceptor and the user registry groups can

be defined at the global, API or service scope. A specific authorization level is granted by ensuring that the authenticated user ID is a member of the user registry group that is assigned to that authorization level at the appropriate scope. If a distributed identity is mapped to a SAF user ID, then the authorization check is performed against the SAF mapped user ID and its group in the SAF registry. When a SAF registry is used by the z/OS Connect EE server, then OMVS segments must be defined for both the SAF user ID and group.

You can use existing user registry groups, or create new groups. If you create new groups, consider whether you want the group names to indicate the authorization level they control, and also the associated scope. For example, you can create separate groups to control APIs and services that belong to different lines of business.

The authorization levels are as follows,

Admin

All z/OS Connect EE actions are allowed, including all *Operations*, *Invoke*, and *Reader* actions.

Operations

All z/OS Connect EE operations and actions are allowed except for *Invoke*.

Invoke

Ability to invoke APIs and services. *Invoke* authority does not provide access to z/OS Connect EE *Operations* actions.

Reader

Ability to obtain information about APIs and services. *Reader* authority does not provide access to z/OS Connect EE *Operations* actions nor does it provide access to invoke APIs or services.

See [Table 31 on page 375](#) for the authorization levels required to perform specific requests.

The z/OS Connect EE authorization interceptor can be configured at the following scopes:

- Globally for the z/OS Connect EE server. By default, this configuration enables the authorization interceptor for all APIs and services. An individual API or service can opt out of running the global interceptors.
- For specific APIs or services. If required for only a few APIs or services, this configuration is an alternative to enabling the authorization interceptor globally.

The user registry groups assigned to each of the authorization levels can also be configured at the following scopes:

- Globally for the z/OS Connect EE server. The user registry groups that are assigned at this scope become the default groups for actions that are performed on all APIs and services.
- For specific APIs or services. The user registry groups that are assigned at this scope take precedence over the global groups.

If you require all or most API or service requests to be authorized, you can configure the authorization interceptor globally for the z/OS Connect EE server and assign the default user registry groups. If you need to restrict specific APIs or services to different users, you can then assign specific user registry groups to those individual APIs or services, to override the global groups. See [Table 31 on page 375](#) for details of the authorization levels required to perform different requests.

Table 31. Authorization levels required to perform requests

Request type	Global authorization level configuration	Specific API or service level configuration (overrides global level except where specified)
Invoke an API. (See note 1).	Admin or Invoke.	Admin or Invoke (See note 2).
Invoke a service.	Admin or Invoke.	Admin or Invoke.
Get a list of APIs.	Admin, Operations, or Reader.	Only global authorization applies.

Table 31. Authorization levels required to perform requests (continued)

Request type	Global authorization level configuration	Specific API or service level configuration (overrides global level except where specified)
Get a list of services.	Admin, Operations, or Reader.	Admin, Operations, or Reader required in addition to global authorization. (See note 3).
Get details of an API or service.	Admin, Operations, or Reader.	Admin, Operations, or Reader.
Get Swagger document for an API.	Admin, Operations, or Reader.	Admin, Operations, or Reader.
Get request and response schema for a service.	Admin, Operations, or Reader.	Admin, Operations, or Reader.
Deploy an API or service.	Admin or Operations.	Only global authorization applies.
Update an API or service.	Admin or Operations.	Admin or Operations.
Delete an API or service.	Admin or Operations.	Admin or Operations.
Change status of an API or service (start or stop).	Admin or Operations.	Admin or Operations.
Get statistics for all services.	Admin or Operations.	Admin or Operations required in addition to global authorization. (See note 3).
Get statistics for a service.	Admin or Operations.	Admin or Operations.
Get z/OS Connect EE admin Swagger doc.	Admin, Operations, or Reader.	Only global authorization applies.

Note:

1. When a user is authorized to invoke an API, the user is allowed to invoke all operations of the API as described in its Swagger document.
2. If an interceptor is driven for an invoke API request, then any interceptor that is configured for the service it calls is not driven. If an interceptor is configured for a service, then it is driven only if that service is invoked directly by an HTTP or HTTPS request.
3. To authorize the RESTful administration action to get a list of services, the authorization interceptor must be configured globally and the authenticated user ID must be a member of an appropriate global group. This configuration returns a list of all deployed services. However, this action can be further refined by additional authorization checks to restrict the list of services that are returned to only those services for which the authenticated user ID is authorized. This is achieved by further configuring the authorization interceptor at the service scope, and assigning the appropriate service-specific groups. If a service is configured in this way, then it is only included in the list of services that are returned if the authenticated user ID is a member of one of these global groups and one of these service-specific groups. The same behavior applies to using the RESTful administration action to get statistics for all services.

For more information about the authorization levels that are required to perform specific RESTful administration actions for APIs, see “[Using the RESTful administration interface](#)” on page 675.

For more information about the authorization levels that are required to perform specific RESTful administration actions for services, see “[Administering services with the administration interface](#)” on page 655.

How to configure the zosConnectAccess role

Configure authorization to control which users can access z/OS Connect EE.

How to configure the zosConnectAccess role with a basic user registry

Assign basic user registry users and groups to the zosConnectAccess role to control which users can access z/OS Connect EE.

This task is applicable when z/OS Connect EE is used as an API provider.

Before you begin

- You should be familiar with the information in [“API provider authorization” on page 373](#).
- You need to know the users and groups that are to be granted access to z/OS Connect EE.
- You must have completed the task [“How to configure basic authentication with a basic user registry” on page 354](#).
- You must have write access to the `server.xml` configuration file.

About this task

Now you have configured the z/OS Connect EE server to require authentication by setting the attribute `requireAuth="true"`, you assign basic registry users and groups to the zosConnectAccess role.

Procedure

1. Assign users and groups to the zosConnectAccess role.

Choose whether you want to assign only specific users or groups to the role, or allow all authenticated users to be assigned to the role.

Follow the instructions in step [“1.a” on page 377](#) to assign specific users and groups to the role.

Follow the instructions in step [“1.b” on page 377](#) to allow all authenticated users to be assigned to the role.

- a) Configure authorization to access z/OS Connect EE, by assigning users and groups to the zosConnectAccess role.

The users and groups must be defined in the basic user registry. To assign the zosConnectAccess role to the "administrators" and "students" groups, for example, add the following `authorization-roles` element to the `server.xml` configuration file:

```
<authorization-roles id="zos.connect.access.roles">
    <security-role name="zosConnectAccess">
        <group name="administrators"/>
        <group name="students"/>
    </security-role>
</authorization-roles>
```

This element uses the following values:

- `id="zos.connect.access.roles"` and `name="zosConnectAccess"`, must be specified exactly as shown.
- "administrators" and "students" are basic user registry groups to be assigned to the role.

- b) Configure authorization to access z/OS Connect EE, by assigning the special subject type "ALL_AUTHENTICATED_USERS" to the zosConnectAccess role.

Use this method if all authenticated basic user registry users and groups are to be authorized to access z/OS Connect EE. Add the following to the `server.xml` configuration file:

```
<authorization-roles id="zos.connect.access.roles">
    <security-role name="zosConnectAccess">
        <special-subject type="ALL_AUTHENTICATED_USERS"/>
    </security-role>
</authorization-roles>
```

```
</security-role>  
</authorization-roles>
```

This element uses the following values:

- id="zos.connect.access.roles" and name="zosConnectAccess", must be specified exactly as shown.
- "ALL_AUTHENTICATED_USERS" is case sensitive.

For more information about the authorization-roles element, see the [Server configuration](#) section in the *IBM WebSphere Application Server for z/OS Liberty* documentation.

2. Update the server configuration or restart the server.

The following message appears in the messages.log file:

```
CWWKS9120I: Authorization roles with id="zos.connect.access.roles" have been successfully processed.
```

Results

The basic user registry users and groups assigned to the zosConnectAccess role now have authorization to access z/OS Connect EE.

What to do next

You might be interested in configuring more specific authorization. See the task [“How to configure authorization levels with a basic user registry” on page 381](#)

How to configure the zosConnectAccess role with an LDAP user registry

Assign LDAP users and groups to the zosConnectAccess role to control which users can access z/OS Connect EE.

This task is applicable when z/OS Connect EE is used as an API provider.

Before you begin

- You should be familiar with the information in [“API provider authorization” on page 373](#).
- You need to know the users and groups that are to be granted access to z/OS Connect EE.
- You must have completed the task [“How to configure basic authentication with an LDAP user registry” on page 355](#), unless you have authenticated using an alternative method which resulted in the authenticated user ID being an LDAP user ID. For example, using a JWT where the identity in the claim is an LDAP user ID, or client certificate authentication where the client certificate subject's distinguished name has been mapped to an LDAP user ID.
- You must have write access to the server.xml configuration file.

About this task

Now you have configured the z/OS Connect EE server to require authentication by setting the attribute requireAuth="true", you assign LDAP registry users and groups to the zosConnectAccess role.

Procedure

1. Assign users and groups to the zosConnectAccess role.

Choose whether you want to assign only specific users or groups to the role, or allow all authenticated users to be assigned to the role.

Follow the instructions in step [“1.a” on page 379](#) to assign specific users and groups to the role.

Follow the instructions in step [“1.b” on page 379](#) to allow all authenticated users to be assigned to the role.

- a) Configure authorization to access z/OS Connect EE, by assigning users and groups to the `zosConnectAccess` role.

The users and groups must be defined in the LDAP server referenced by the `ldapRegistry` element. To assign the `zosConnectAccess` role to a group and a user, add the `authorization-roles` element to the `server.xml` configuration file:

```
<authorization-roles id="zos.connect.access.roles">
    <security-role name="zosConnectAccess">
        <group name="employees"
            access-id="group:SampleLdapIDSRealm/cn=employees,ou=groups,o=mop,c=fr"/>
        <user name="PierreDuclos"
            access-id="user:SampleLdapIDSRealm/
uid=PierreDuclos,ou=customers,o=mop,c=fr"/>
    </security-role>
</authorization-roles>
```

This element uses the following values:

- The values, `id="zos.connect.access.roles"` and `name="zosConnectAccess"`, must be specified exactly as shown.
- "employees" is an LDAP group to be assigned to the role. Its distinguished name (DN) is "`cn=employees,ou=groups,o=mop,c=fr`".
- "PierreDuclos" is an LDAP user to be assigned to the role. His distinguished name (DN) is "`uid=PierreDuclos,ou=customers,o=mop,c=fr`".
- The values for `access-id` are optional. If not specified, a value will be generated by looking up the user or group in the LDAP server. If specified, the `access-id` has the following format:
 - For a group: `realmName/groupUniqueId`.
 - For a user: `realmName/userUniqueId`.

The values are made up of the following components:

- `realmName` is the realm value specified on the `ldapRegistry` element.
- `groupUniqueId` is the distinguished name (DN) value of the group. The value is not case sensitive.
- `userUniqueId` is the distinguished name (DN) value of the user. The value is not case sensitive.

- b) Configure authorization to access z/OS Connect EE, by assigning the special subject type "ALL_AUTHENTICATED_USERS" to the `zosConnectAccess` role.

Use this method if all authenticated LDAP users and groups are to be authorized to access z/OS Connect EE. Add the following to the `server.xml` configuration file:

```
<authorization-roles id="zos.connect.access.roles">
    <security-role name="zosConnectAccess">
        <special-subject type="ALL_AUTHENTICATED_USERS"/>
    </security-role>
</authorization-roles>
```

This element uses the following values:

- `id="zos.connect.access.roles"` and `name="zosConnectAccess"`, must be specified exactly as shown.
- "ALL_AUTHENTICATED_USERS" is case sensitive.

For more information about the `authorization-roles` element, see the [Server configuration](#) section in the *IBM WebSphere Application Server for z/OS Liberty* documentation.

2. Update the server configuration or restart the server.

The following message appears in the `messages.log` file:

`CWWK9120I: Authorization roles with id="zos.connect.access.roles" have been successfully processed.`

Results

The LDAP users and groups assigned to the `zosConnectAccess` role now have authorization to access z/OS Connect EE.

What to do next

You might be interested in configuring more specific authorization. See the task “[How to configure authorization levels with an LDAP user registry](#)” on page 385.

How to configure the `zosConnectAccess` role with a SAF user registry

Assign SAF users and groups to the `zosConnectAccess` role to control which users can access z/OS Connect EE.

This task is applicable when z/OS Connect EE is used as an API provider.

Before you begin

- You should be familiar with the information in “[API provider authorization](#)” on page 373.
- You need to know the SAF user IDs and groups that are to be granted access to z/OS Connect EE.
- You must have completed the task “[How to configure basic authentication with a SAF user registry](#)” on page 358, or have authenticated using an alternative method and mapped the authenticated user ID to a SAF user ID.
- You must have write access to the `server.xml` configuration file.

About this task

Now you have configured the z/OS Connect EE server to require authentication by setting the attribute `requireAuth="true"`, you assign SAF users and groups to the `zosConnectAccess` role.

This task assumes that RACF is used as security manager. If you are using an alternative External Security Manager, refer to the appropriate product documentation for the equivalent commands.

Procedure

1. Configure the server to use SAF for authorization.

This configures the z/OS Connect EE server to perform authorization checks against the SAF registry. The `zosConnectAccess` role check is performed against the authenticated SAF user ID. Add the following element to the `server.xml` configuration file:

```
<safAuthorization id="saf-authorization"/>
```

To display SAF authorization messages such as RACF ICH408I, when unauthorized users attempt to access z/OS Connect EE, specify the attribute `racRouteLog="ASIS"`.

For more information about the `safAuthorization` element, see the [Server configuration](#) section in the *IBM WebSphere Application Server for z/OS Liberty* documentation.

2. Assign users and groups to the `zosConnectAccess` role.

For SAF authorization, access is controlled using the SAF EJBROLE profile `profilePrefix.zos.connect.access.roles.zosConnectAccess`.

- a) Define the SAF EJBROLE profile.

Enter the following command:

```
RDEFINE EJBROLE profilePrefix.zos.connect.access.roles.zosConnectAccess UACC(NONE)
```

In this command, `profilePrefix` is the value used for this server as specified on the `profilePrefix` attribute of the `safCredentials` element in `server.xml`. The default profile prefix value is `BBGZDFLT`.

b) Activate the EJBROLE class

Enter the following command

SETROPTS CLASSACT(EJBROLE)

c) Assign the user IDs and groups who require authority to access.z/OS Connect EE READ access to this profile.

For example, to assign group "STAFF" and user "EMPLOY1" to the zosConnectAccess role, enter the following commands:

**PERMIT *profilePrefix*.zos.connect.access.roles.zosConnectAccess CLASS(EJBROLE) ID(STAFF)
ACCESS(READ)**

**PERMIT *profilePrefix*.zos.connect.access.roles.zosConnectAccess CLASS(EJBROLE)
ID(EMPLOY1) ACCESS(READ)**

d) Refresh the EJBROLE class.

Enter the following command:

SETROPTS RACLIST(EJBROLE) REFRESH

e) Ensure the user IDs and groups have also been granted READ access to the *profilePrefix* APPL profile.

This is described in the task [“How to activate and configure the SAF user registry” on page 456](#). Enter the following command:

PERMIT *profilePrefix* CLASS(APPL) ACCESS(READ) ID(EMPLOY1)

3. Start, or restart the server if it was already running, to pick up the changes made to the RACF class profiles.

Results

The SAF users and groups assigned to the zosConnectAccess role now have authorization to access z/OS Connect EE.

What to do next

You might be interested in configuring more specific authorization. See the task [“How to configure authorization levels with a SAF user registry” on page 389](#).

How to configure the authorization levels

Configure authorization to control which users can perform specific actions on z/OS Connect EE APIs or services

How to configure authorization levels with a basic user registry

Configure authorization to control which users can perform specific actions on z/OS Connect EE APIs or services using a basic user registry.

This task is applicable when z/OS Connect EE is used as an API provider.

Before you begin

- You should be familiar with the information in [“API provider authorization” on page 373](#).
- You need to know which users and groups are to be granted various authorization levels to which APIs or services.
- You must have write access to the `server.xml` configuration file.
- You must have configured the z/OS Connect EE server to authenticate the users, and ensure each of those users is associated with a group in the basic user registry. These tasks are described in [“How to configure basic authentication with a basic user registry” on page 354](#).

About this task

Configure a z/OS Connect EE server to perform authorization checks by using the z/OS Connect EE authorization interceptor. You assign basic registry groups to each of the authorization levels: *admin*, *operations*, *invoke* and *reader* globally or for a specific API or service.

You configure the authorization interceptor at the global scope with basic user registry groups that are assigned to each of the global authorization levels. The configuration examples demonstrate the following options:

- How specific APIs or services can opt out of using the globally configured authorization interceptor. See examples for "Api1" and "Service1" in step ["5" on page 383](#).
- How the authorization interceptor can be configured only for specific APIs or services instead of at the global scope. See examples for "Api2" and "Service2" in step ["6" on page 383](#).
- How different basic user registry groups can be assigned to each of the authorization levels for specific APIs or services. See examples for "Api3" and "Service3" in step ["7" on page 384](#).

Note:

- If the interceptor is configured for both an API and the service it calls, then an HTTP or HTTPS request to invoke the API drives only the interceptor for the API. If the interceptor is configured for a service, then it is only driven if that service is invoked directly by an HTTP or HTTPS request.
- If you want to configure authorization for the RESTful administration actions such as deploy an API, deploy a service, get a list of APIs, get a list of services, or get statistics for multiple services, additional configuration is needed. For more information, see the ["API provider authorization" on page 373](#).

Procedure

1. Configure the z/OS Connect EE authorization interceptor.

Add the following element to the `server.xml` configuration file:

```
<zosconnect_authorizationInterceptor id="zosConnectAuthorizationInterceptor" />
```

For more information about the `zosconnect_authorizationInterceptor` element see ["zosconnect_authorizationInterceptor" on page 758](#) in the *Reference* section.

2. Configure which interceptors are to be called.

The z/OS Connect EE authorization interceptor is called by referencing it in the following element of the `server.xml` configuration file:

```
<zosconnect_zosConnectInterceptors id="interceptorList1"
    interceptorRef="zosConnectAuthorizationInterceptor" />
```

In this example, `zosConnectAuthorizationInterceptor` is the `id` attribute value of the referenced `zosconnect_authorizationInterceptor` element.

For more information about the `zosconnect_zosConnectInterceptors` element, see ["zosconnect_zosConnectInterceptors" on page 784](#) in the *Reference* section.

3. Optional: Configure the authorization interceptor at the global scope.

Perform this step if you want to control authorization to the RESTful administration actions to deploy an API, deploy a service, list all APIs, list all services, and get statistics for all services. It also sets the default authorization for all APIs and services.

To configure the authorization interceptor globally, add the `globalInterceptorsRef` attribute to the `zosconnect_zosConnectManager` element. For example:

```
<zosconnect_zosConnectManager
    globalInterceptorsRef="interceptorList1"
    ... />
```

In this example, `interceptorList1` is the `id` attribute value of the referenced `zosconnect_zosConnectInterceptors` element.

Perform step “4” on page 383 only if you have completed step “3” on page 382.

4. Optional: Assign the basic user registry groups to each authorization level at the global scope.

The basic user registry groups are authorized by default to perform actions on all APIs and services. These global groups are also used to authorize access to RESTful administration operations to deploy an API, deploy a service, list all APIs, and list all services.

To configure the authorization level groups globally, add the attributes to define the groups to the `zosconnect_zosConnectManager` element. For example:

```
<zosconnect_zosConnectManager  
    globalInterceptorsRef="interceptorList1"  
    globalAdminGroup="administrators"  
    globalInvokeGroup="students"  
    globalOperationsGroup="opsStaff"  
    globalReaderGroup="librarians, guests"  
    ... />
```

The values of the group attributes must be one or more basic user registry group names. The values are case-insensitive. To specify multiple groups for any role, use a comma-separated list, as shown for the `globalReaderGroup`.

For more information about the `zosconnect_zosConnectManager` element, see “[zosconnect_zosConnectManager](#)” on page 784 in the *Reference* section.

5. Optional: Opt out of using the authorization interceptor’s global scope for an individual API or service.

If the authorization interceptor is configured at the global scope, authorization checks are applied to all requests for APIs and services because, by default, APIs and services run all the global interceptors, in addition to any interceptors configured specifically at the API or service scope.

To opt out of running the authorization interceptor for API `Api1`, ensure that a `zosConnectAPI` element for `Api1` is configured and add the `runGlobalInterceptors="false"` attribute to it. For example:

```
<zosconnect_zosConnectAPIs>  
    <zosConnectAPI name="Api1"  
        runGlobalInterceptors="false"  
        ... />  
</zosconnect_zosConnectAPIs>
```

To opt out of running the authorization interceptor for service `Service1`, ensure that a service element for `Service1` is configured and add the `runGlobalInterceptors="false"` attribute to it. For example:

```
<zosconnect_services>  
    <service name="Service1"  
        runGlobalInterceptors="false"  
        ... />  
</zosconnect_services>
```

6. Optional: Configure the authorization interceptor for specific APIs or services.

Perform this step only if you do not require the authorization interceptor to be configured globally. Instead, use this configuration to control authorization only for specific APIs or services, or because you wish to restrict the services returned by the RESTful administration actions to list all services or get statistics for all services.

a) Remove `globalInterceptorsRef="interceptorList1"` from the `zosconnect_zosConnectManager` element if specified.

b) Optional: Configure the authorization interceptor for the specific API.

For example, for the API `Api2`, ensure that a `zosConnectAPI` element for `Api2` is configured and add the `interceptorsRef` attribute to it. For example:

```
<zosconnect_zosConnectAPIs>  
    <zosConnectAPI name="Api2" interceptorsRef="interceptorList1"  
        ... />  
</zosconnect_zosConnectAPIs>
```

In this example, `interceptorList1` is the `id` attribute value of the referenced `zosconnect_zosConnectInterceptors` element.

c) Optional: Configure the authorization interceptor for a specific service.

For example, for the service Service2, ensure that a `service` element for Service2 is configured and add the `interceptorsRef` attribute to it. For example:

```
<zosconnect_services>
  <service name="Service2"
    interceptorsRef="interceptorList1"
    ... />
</zosconnect_services>
```

In this example, `interceptorList1` is the `id` attribute value of the referenced `zosconnect_zosConnectInterceptors` element.

For more information about the API and service elements, see [zosConnectAPI](#) and [service](#) in the *Reference* section.

7. Optional: Assign the basic user registry groups to authorization levels at the API or service scope.

These groups take precedence over the global groups except when controlling authorization to the RESTful administration actions to deploy an API, deploy a service, list all APIs, or list all services.

- Define either `globalInterceptorsRef="interceptorList1"` on the `zosconnect_zosConnectManager` element, or `interceptorsRef="interceptorList1"` on the `zosConnectAPI` or `service` element.
- Optional: To configure the authorization level groups for the API Api3, ensure that a `zosConnectAPI` element for Api3 is configured and add the attributes to define the groups to it.

For example:

```
<zosconnect_zosConnectAPIs>
  <zosConnectAPI name="Api3"
    adminGroup="a3Admin"
    invokeGroup="a3Invoke"
    operationsGroup="a3Operations"
    readerGroup="a3Reader"
    ... />
</zosconnect_zosConnectAPIs>
```

In this example, the values of the group attributes must be the basic user registry group names. The values are case-insensitive.

c) Optional: To configure the authorization level groups for Service Service3, ensure that a `service` element for Service3 is configured and add the attributes to define the groups to it.

For example:

```
<zosconnect_services>
  <service name="Service3"
    adminGroup="s3Admin"
    invokeGroup="s3Invoke"
    operationsGroup="s3Operations"
    readerGroup="s3Reader"
    ... />
</zosconnect_services>
```

In this example, the values of the group attributes must be the basic user registry group names. The values are case-insensitive

Note: If a group is to be assigned to the same authorization level at all scopes, then it can be specified at the global scope and inherited by all APIs and services. For example, to assign the same group to the reader authorization level at all scopes, set the `globalReaderGroup` attribute of the `zosconnect_zosConnectManager` element, and omit the `readerGroup` attribute of each individual API or service element that should inherit it.

8. Update the server configuration or restart the server.

Results

Your z/OS Connect EE server is configured to perform authorization checks by using the z/OS Connect EE authorization interceptor, and basic user registry groups are assigned to each of the authorization levels.

If a request fails an authorization check, messages are written to the messages.log file. For example, if a user attempts to invoke an API, but has reader access only, the following messages are written:

BAQR0409W: User RoseMoubinou is not authorized to perform the request.

BAQR0428W: The zosConnectAuthorization interceptor encountered an error while processing a request for API myAPI under request URL http://myhost.company.com:9080/myAPI/mydata.

The following messages are returned in the HTTP response body:

BAQR0409W: User RoseMoubinou is not authorized to perform the request.

BAQR0436W: The zosConnectAuthorization interceptor encountered an error while processing a request for API myAPI.

How to configure authorization levels with an LDAP user registry

Configure authorization to control which users can perform specific actions on z/OS Connect EE APIs or services by using an LDAP user registry.

This task is applicable when z/OS Connect EE is used as an API provider.

Before you begin

- You should be familiar with the information in [“API provider authorization” on page 373](#).
- You need to know which LDAP users and groups are to be granted various authorization levels to which APIs or services.
- You must have write access to the `server.xml` configuration file.
- You must have configured the z/OS Connect EE server to use an LDAP user registry to authenticate and authorize the users. For example, you can use one of the following methods:
 - Use basic authentication as described in the task [“How to configure basic authentication with an LDAP user registry” on page 355](#).
 - Authenticate with a JWT where the identity in the claim is an LDAP user ID.
 - Use client certificate authentication where the distinguished name of the client certificate subject has been mapped to an LDAP user ID.

About this task

Configure a z/OS Connect EE server to perform authorization checks by using the z/OS Connect EE authorization interceptor. You assign LDAP registry groups to each of the authorization levels: *admin*, *operations*, *invoke* and *reader* globally or for a specific API or service.

You configure the authorization interceptor at the global scope with LDAP groups that are assigned to each of the global authorization levels. The configuration examples demonstrate the following options:

- How specific APIs or services can opt out of using the globally configured authorization interceptor. See examples for `Api1` and `Service1` in step [“5” on page 387](#).
- How the authorization interceptor can be configured only for specific APIs or services instead of at the global scope. See examples for `Api2` and `Service2` in step [“6” on page 387](#).
- How different LDAP user registry groups can be assigned to each of the authorization levels for specific APIs or services. See examples for `Api3` and `Service3` in step [“7” on page 387](#).

Note:

- If the interceptor is configured for both an API and the service it calls, then an HTTP or HTTPS request to invoke the API drives only the interceptor for the API. If the interceptor is configured for a service, then it is only driven if that service is invoked directly by an HTTP or HTTPS request.

- If you want to configure authorization for the RESTful administration actions such as deploy an API, deploy a service, get a list of APIs, get a list of services, or get statistics for multiple services, additional configuration is needed. For more information, see the “[API provider authorization](#)” on page 373.

Procedure

1. Configure the z/OS Connect EE authorization interceptor.

Add the following element to the `server.xml` configuration file:

```
<zosconnect_authorizationInterceptor id="zosConnectAuthorizationInterceptor"/>
```

For more information about the `zosconnect_authorizationInterceptor` element see “[zosconnect_authorizationInterceptor](#)” on page 758 in the *Reference* section.

2. Configure which interceptors are to be called.

The z/OS Connect EE authorization interceptor is called by referencing it in the following element of the `server.xml` configuration file:

```
<zosconnect_zosConnectInterceptors id="interceptorList1"
    interceptorRef="zosConnectAuthorizationInterceptor"/>
```

In this example, `zosConnectAuthorizationInterceptor` is the `id` attribute value of the referenced `zosconnect_authorizationInterceptor` element.

For more information about the `zosconnect_zosConnectInterceptors` element, see “[zosconnect_zosConnectInterceptors](#)” on page 784 in the *Reference* section.

3. Optional: Configure the authorization interceptor at the global scope.

Perform this step if you want to control authorization to the RESTful administration actions to deploy an API, deploy a service, list all APIs, list all services, and get statistics for all services. It also sets the default authorization for all APIs and services.

To configure the authorization interceptor globally, add the `globalInterceptorsRef` attribute to the `zosconnect_zosConnectManager` element. For example:

```
<zosconnect_zosConnectManager
    globalInterceptorsRef="interceptorList1"
    ... />
```

In this example, `interceptorList1` is the `id` attribute value of the referenced `zosconnect_zosConnectInterceptors` element.

Perform step “[4](#)” on page 386 only if you have completed step “[3](#)” on page 386.

4. Optional: Configure the LDAP groups to be assigned to each authorization level at the global scope.

The LDAP groups are authorized by default to perform actions on all APIs and services. These global groups are also used to authorize access to RESTful administration operations to deploy an API, deploy a service, list all APIs, and list all services.

To configure the authorization level groups globally, add the attributes to define the groups to the `zosconnect_zosConnectManager` element. For example:

```
<zosconnect_zosConnectManager>
    globalInterceptorsRef="interceptorList1"
    globalAdminGroup="cn=admins\,ou=groups\,o=mop\,c=fr"
    globalInvokeGroup="cn=customers\,ou=groups\,o=mop\,c=fr"
    globalOperationsGroup="cn=employees\,ou=groups\,o=mop\,c=fr"
    globalReaderGroup="cn=partner1\,ou=groups\,o=mop\,c=fr, cn=guests\,ou=groups\,o=mop
    \,c=fr"
    ... />
```

The values of the group attributes must be one or more LDAP group distinguished names (DNs), with the commas within the names escaped with a backslash. The values are case-insensitive. To specify multiple groups for any role, use a comma-separated list, as shown for the `globalReaderGroup`, with the commas separating the groups not escaped.

For more information about the `zosconnect_zosConnectManager` element, see “[zosconnect_zosConnectManager](#)” on page 784 in the *Reference* section.

5. Optional: Opt out of using the authorization interceptor's global scope for an individual API or service.

If the authorization interceptor is configured at the global scope, authorization checks are applied to all requests for APIs and services because, by default, APIs and services run all the global interceptors, in addition to any interceptors configured specifically at the API or service scope.

To opt out of running the authorization interceptor for API `Api1`, ensure that a `zosConnectAPI` element for `Api1` is configured and add the `runGlobalInterceptors="false"` attribute to it. For example:

```
<zosconnect_zosConnectAPIs>
  <zosConnectAPI name="Api1"
    runGlobalInterceptors="false"
  ... />
</zosconnect_zosConnectAPIs>
```

To opt out of running the authorization interceptor for service `Service1`, ensure that a `service` element for `Service1` is configured and add the `runGlobalInterceptors="false"` attribute to it. For example:

```
<zosconnect_services>
  <service name="Service1"
    runGlobalInterceptors="false"
  ... />
</zosconnect_services>
```

6. Optional: Configure the authorization interceptor for specific APIs or services.

Do not perform this step if you require the authorization interceptor to be configured globally. Use this configuration to control authorization for specific APIs or services only, or to restrict the services that are returned by the RESTful administration actions to “list all services” or “get statistics for all services”.

a) Remove `globalInterceptorsRef="interceptorList1"` from the `zosconnect_zosConnectManager` element if specified.

b) Optional: Configure the authorization interceptor for the specific API.

For example, for the API `Api2`, ensure that a `zosConnectAPI` element for `Api2` is configured and add the `interceptorsRef` attribute to it. For example:

```
<zosconnect_zosConnectAPIs>
  <zosConnectAPI name="Api2" interceptorsRef="interceptorList1"
  ... />
</zosconnect_zosConnectAPIs>
```

In this example, `interceptorList1` is the `id` attribute value of the referenced `zosconnect_zosConnectInterceptors` element.

c) Optional: Configure the authorization interceptor for a specific service.

For example, for the service `Service2`, ensure that a `service` element for `Service2` is configured and add the `interceptorsRef` attribute to it. For example:

```
<zosconnect_services>
  <service name="Service2"
    interceptorsRef="interceptorList1"
  ... />
</zosconnect_services>
```

In this example, `interceptorList1` is the `id` attribute value of the referenced `zosconnect_zosConnectInterceptors` element.

For more information about the API and service elements, see [zosConnectAPI](#) and [service](#) in the *Reference* section.

7. Optional: Assign the LDAP groups to authorization levels at the API or service scope.

These groups take precedence over the global groups. However, there are additional considerations for the RESTful administration actions to deploy an API, deploy a service, list all APIs, or list all services. For more information, see “[API provider authorization](#)” on page 373.

- a) Define either `globalInterceptorsRef="interceptorList1"` on the `zosconnect_zosConnectManager` element, or `interceptorsRef="interceptorList1"` on the `zosConnectAPI` or `service` element.
- b) Optional: To configure the authorization level groups for the API `Api3`, ensure that a `zosConnectAPI` element for `Api3` is configured and add the attributes to define the groups to it.

For example:

```
<zosconnect_zosConnectAPIs>
  <zosConnectAPI name="Api3"
    adminGroup="cn=admins\,ou=groups\,o=bank\,c=fr"
    invokeGroup="cn=customers\,ou=groups\,o=bank\,c=fr"
    operationsGroup="cn=ops\,ou=groups\,o=bank\,c=fr"
    readerGroup="cn=appdev\,ou=groups\,o=bank\,c=fr, cn=auditors\,ou=groups\,o=bank
  \,c=fr"
    ...
  />
</zosconnect_zosConnectAPIs>
```

The values of the group attributes must be one or more LDAP group distinguished names (DNs), with the commas within the names escaped with a backslash. The values are case-insensitive. To specify multiple groups for any role, use a comma-separated list, as shown for the `globalReaderGroup`, with the commas separating the groups not escaped.

- c) Optional: To configure the authorization level groups for Service `Service3`, ensure that a `service` element for `Service3` is configured and add the attributes to define the groups to it.

For example:

```
<zosconnect_services>
  <service name="Service3"
    adminGroup="cn=admins\,ou=groups\,o=sales\,c=fr"
    invokeGroup="cn=customers\,ou=groups\,o=sales\,c=fr"
    operationsGroup="cn=ops\,ou=groups\,o=sales\,c=fr"
    readerGroup="cn=managers\,ou=groups\,o=sales\,c=fr"
    ...
  />
</zosconnect_services>
```

Note: If a group is to be assigned to the same authorization level at all scopes, then it can be specified at the global scope and inherited by all APIs and services. For example, to assign the same group to the reader authorization level at all scopes, set the `globalReaderGroup` attribute of the `zosconnect_zosConnectManager` element, and omit the `readerGroup` attribute of each individual API or service element that inherits it.

8. Update the server configuration or restart the server.

Results

Your z/OS Connect EE server is configured to perform authorization checks by using the z/OS Connect EE authorization interceptor, and LDAP groups are assigned to each of the authorization levels.

If a request fails an authorization check, messages are written to the `messages.log` file. For example, if a user attempts to invoke an API, but has reader access only, the following messages are written:

BAQR0409W: User RoseMoubinou is not authorized to perform the request.
BAQR0428W: The `zosConnectAuthorization` interceptor encountered an error while processing a request for API `myAPI` under request URL `http://myhost.company.com:9080/myAPI/mydata`.

The following messages are returned in the HTTP response body:

BAQR0409W: User RoseMoubinou is not authorized to perform the request.
BAQR0436W: The `zosConnectAuthorization` interceptor encountered an error while processing a request for API `myAPI`.

How to configure authorization levels with a SAF user registry

Configure authorization to control which users can perform specific actions on z/OS Connect EE APIs or services using a SAF user registry.

This task is applicable when z/OS Connect EE is used as an API provider.

Before you begin

- You should be familiar with the information in “[API provider authorization](#)” on page 373.
- You need to know which user IDs and groups are to be granted various authorization levels to which APIs or services.
- Both the SAF user ID associated with any request, and the SAF groups assigned to the authorization levels must have OMVS segments. If you use RACF, specify the following OMVS segment parameters:
 - AUTOGID or GID(group-identifier) on the ADDGROUP or ALTGROUP command.
 - AUTOUID or UID(user-identifier), HOME and PROGRAM on the ADDUSER or ALTUSER command. For example: OMVS(AUTOUID HOME(/u/user) PROGRAM(/bin/sh)).
- You must have write access to the `server.xml` configuration file.
- You must have configured the z/OS Connect EE server to authenticate the SAF user IDs, and ensure that each of those user IDs is associated with a group in the SAF registry. These tasks are described in “[How to configure basic authentication with a SAF user registry](#)” on page 358 or, if authenticating with a JWT whose subject claim is, or is mapped to, a SAF user ID, “[How to configure JWT authentication](#)” on page 366.

About this task

Configure a z/OS Connect EE server to perform authorization checks by using the z/OS Connect EE authorization interceptor. You assign SAF groups to each of the authorization levels: *admin*, *operations*, *invoke* and *reader* globally or for a specific API or service.

You configure the authorization interceptor at the global scope with SAF groups that are assigned to each of the global authorization levels. The configuration examples demonstrate the following options:

- How specific APIs or services can opt out of using the globally configured authorization interceptor. See examples for “*Api1*” and “*Service1*” in step [“5” on page 390](#).
- How the authorization interceptor can be configured only for specific APIs or services instead of at the global scope. See examples for “*Api2*” and “*Service2*” in step [“6” on page 391](#).
- How different SAF groups can be assigned to each of the authorization levels for specific APIs or services. See examples for “*Api3*” and “*Service3*” in step [“7” on page 391](#).

Note:

- If the interceptor is configured for both an API and the service it calls, then an HTTP or HTTPS request to invoke the API drives only the interceptor for the API. If the interceptor is configured for a service, then it is only driven if that service is invoked directly by an HTTP or HTTPS request.
- If you want to configure authorization for the RESTful administration actions such as deploy an API, deploy a service, get a list of APIs, get a list of services, or get statistics for multiple services, additional configuration is needed. For more information, see the “[API provider authorization](#)” on page 373.

Procedure

1. Configure the z/OS Connect EE authorization interceptor.

Add the following element to the `server.xml` configuration file:

```
<zosconnect_authorizationInterceptor id="zosConnectAuthorizationInterceptor"/>
```

For more information about the `zosconnect_authorizationInterceptor` element see “[zosconnect_authorizationInterceptor](#)” on page 758 in the *Reference* section.

2. Configure which interceptors are to be called.

The z/OS Connect EE authorization interceptor is called by referencing it in the following element of the `server.xml` configuration file:

```
<zosconnect_zosConnectInterceptors id="interceptorList1"
    interceptorRef="zosConnectAuthorizationInterceptor"/>
```

In this example, `zosConnectAuthorizationInterceptor` is the `id` attribute value of the referenced `zosconnect_authorizationInterceptor` element.

For more information about the `zosconnect_zosConnectInterceptors` element, see [“zosconnect_zosConnectInterceptors” on page 784](#) in the *Reference* section.

3. Optional: Configure the authorization interceptor at the global scope.

Perform this step to assign the SAF groups that are authorized by default to perform actions on all APIs and services. These global groups are also used to authorize access to RESTful administration actions to deploy an API, deploy a service, list all APIs, list all services, and get statistics for all services.

To configure the authorization interceptor globally, add the `globalInterceptorsRef` attribute to the `zosconnect_zosConnectManager` element. For example:

```
<zosconnect_zosConnectManager
    globalInterceptorsRef="interceptorList1"
    ... />
```

In this example, `interceptorList1` is the `id` attribute value of the referenced `zosconnect_zosConnectInterceptors` element.

Perform step “4” on page 390 only if you have completed step “3” on page 390.

4. Optional: Configure the SAF groups and default groups to be assigned to each authorization level at the global scope.

The SAF groups are authorized by default to perform actions on all APIs and services. These global groups are also used to authorize access to RESTful administration operations to deploy an API, deploy a service, list all APIs, and list all services.

To configure the authorization level groups globally, add the attributes to define the groups to the `zosconnect_zosConnectManager` element. For example:

```
<zosconnect_zosConnectManager
    globalInterceptorsRef="interceptorList1"
    globalAdminGroup="GMADMIN"
    globalInvokeGroup="GMINVOKE"
    globalOperationsGroup="GMOPER"
    globalReaderGroup="GMREADER, GMGUEST"
    ... />
```

The values of the group attributes must be one or more SAF group names. The values are case-insensitive. To specify multiple groups for any role, use a comma-separated list, as shown for the `globalReaderGroup`.

For more information about the `zosconnect_zosConnectManager` element, see [“zosconnect_zosConnectManager” on page 784](#) in the *Reference* section.

5. Optional: Opt out of using the authorization interceptor's global scope for an individual API or service.

If the authorization interceptor is configured at the global scope, authorization checks are applied to all requests for APIs and services because, by default, APIs and services run all the global interceptors, in addition to any interceptors configured specifically at the API or service scope.

To opt out of running the authorization interceptor for API `Api1`, ensure that a `zosConnectAPI` element for `Api1` is configured and add the `runGlobalInterceptors="false"` attribute to it. For example:

```
<zosconnect_zosConnectAPIs>
    <zosConnectAPI name="Api1"
        runGlobalInterceptors="false">
```

```
... />
</zosconnect_zosConnectAPIs>
```

To opt out of running the authorization interceptor for service Service1, ensure there is a service element for Service1 and add the `runGlobalInterceptors="false"` attribute to it. For example:

```
<zosconnect_services>
  <service name="Service1"
    runGlobalInterceptors="false"
  .../>
</zosconnect_services>
```

6. Optional: Configure the authorization interceptor for specific APIs or services.

Perform this step only if you do not require the authorization interceptor to be configured globally and only need to control authorization for specific APIs or services; or because you wish to restrict the services returned by the RESTful administration actions to list all services or get statistics for all services.

a) Remove `globalInterceptorsRef="interceptorList1"` from the `zosconnect_zosConnectManager` element, if specified.

b) Optional: Configure the authorization interceptor for the specific API.

For example, for the API Api2, ensure that a `zosConnectAPI` element for Api2 is configured and add the `interceptorsRef` attribute to it. For example:

```
<zosconnect_zosConnectAPIs>
  <zosConnectAPI name="Api2" interceptorsRef="interceptorList1"
  ... />
</zosconnect_zosConnectAPIs>
```

In this example, `interceptorList1` is the `id` attribute value of the referenced `zosconnect_zosConnectInterceptors` element.

c) Optional: Configure the authorization interceptor for a specific service.

For example, for the service Service2, ensure that a `service` element for Service2 is configured and add the `interceptorsRef` attribute to it. For example:

```
<zosconnect_services>
  <service name="Service2"
    interceptorsRef="interceptorList1"
  ... />
</zosconnect_services>
```

In this example, `interceptorList1` is the `id` attribute value of the referenced `zosconnect_zosConnectInterceptors` element.

For more information about the API and service elements, see [zosConnectAPI](#) and [service](#) in the *Reference* section.

7. Optional: Assign the SAF groups to authorization levels at the API or service scope.

These groups take precedence over the global groups except when controlling authorization to the RESTful administration actions to deploy an API, deploy a service, list all APIs, or list all services.

a) Define either `globalInterceptorsRef="interceptorList1"` on the `zosconnect_zosConnectManager` element, or `interceptorsRef="interceptorList1"` on the `zosConnectAPI` or `service` element.

b) Optional: To configure the authorization level groups for the API Api3, ensure that a `zosConnectAPI` element for Api3 is configured and add the attributes to define the groups to it.

For example:

```
<zosconnect_zosConnectAPIs>
  <zosConnectAPI name="Api3"
    adminGroup="A3ADMIN"
    invokeGroup="A3INVOKE"
    operationsGroup="A3OPER"
    readerGroup="A3READER"
```

```

... />
</zosconnect_zosConnectAPIs>
```

In this example, the values of the group attributes must be the SAF group names. The values are case-insensitive.

- c) Optional: To configure the authorization level groups for service Service3, ensure that a service element for Service3 is configured and add the attributes to define the groups to it.

For example:

```

<zosconnect_services>
  <service name="Service3"
    adminGroup="S3ADMIN"
    invokeGroup="S3INVOKE"
    operationsGroup="S3OPER"
    readerGroup="S3READER"
    ... />
</zosconnect_services>
```

In this example, the values of the group attributes must be the SAF group names. The values are case-insensitive

Note: If a group is to be assigned to the same authorization level at all scopes, then it can be specified at the global scope and inherited by all APIs and services. For example, to assign the same group to the reader authorization level at all scopes, set the `globalReaderGroup` attribute of the `zosconnect_zosConnectManager` element, and omit the `readerGroup` attribute of each individual API or service element that inherits it.

8. Update the server configuration or restart the server.

Results

Your z/OS Connect EE server is configured to perform authorization checks by using the z/OS Connect EE authorization interceptor, and SAF groups are assigned to each of the authorization levels.

API provider audit

Learn how z/OS Connect EE audits access to z/OS Connect EE APIs and services.

Before you study this topic, you should be familiar with the information in “[Overview of z/OS Connect EE security](#)” on page 331 and “[API provider authentication and identification](#)” on page 349.

Auditing provides accountability allowing you to capture and record which users have called what APIs and services.

Auditing occurs after authentication has taken place. One of the ways you can audit requests is by using the audit interceptor that is supplied with z/OS Connect EE.

The audit interceptor can be configured globally for all APIs and services, or for specific APIs or services. The output from the audit interceptor includes information about each request, as shown in [Table 32 on page 392](#) and [Table 33 on page 393](#).

Table 32. Extract from audit interceptor output for an API request

System	Jobname	...	Arrival Time	Service or API	Target URI	Method	User name	Mapped user name
ZT01	ZOSCON N	...	12:44:31.69	catalog	/catalog/orders	POST	JeanLeclerc	EMPLOY1

[Table 32 on page 392](#) shows that on system ZT01 at 12:44:31, API catalog_v1 was requested by JeanLeclerc whose id was mapped to SAF user ID EMPLOY1.

Table 33. Extract from audit interceptor output for an administration request to discover services

System	Jobname	...	Arrival Time	Service or API	Target URI	Method	User name	Mapped user name
ZT02	ZOSCON N	...	13:22:30.78	catalog	/zosConnect/services	GET	NicholasBoss	ZCOBOSS

Table 33 on page 393 shows that on system ZT02 at 13:22:30 user NicolasBoss sent an administration request to get a list of services. His user ID has been mapped to a SAF user ID (ZCOBOSS) using a distributed identity filter.

For more information about how to configure the audit interceptor, see [“Configuring the audit interceptor” on page 291](#)

API provider security for service providers

Learn how to configure security between your z/OS Connect EE server and the System of Record (SoR), for example, CICS, IMS, IBM, Db2.

Before you study this topic, you should be familiar with the information in [“Overview of z/OS Connect EE security” on page 331](#).

To secure the connection between the z/OS Connect EE server and the SoR, you must configure security in both the z/OS Connect EE server and the SoR. The security options and configuration required is different for each service provider.



CICS service provider

- Connections can use TLS. AT-TLS can not be used for this connection.
- Bind, link and user security can be used to secure your IPIC connection with CICS.
- ID propagation is supported.

For further information, see [“Configuring security for an IPIC connection” on page 225](#).

IMS service provider

- Secure connections between the IMS service provider and IMS Connect must use AT-TLS.
- User authentication is supported.
- ID propagation is supported.

For further information, see [“Security configuration for the IMS service provider” on page 250](#).

IMS database service provider

- Secure connections between the IMS database service provider and IMS Connect must use AT-TLS.
- User authentication is supported via basic authentication and with RACF PassTickets.
- SAF distributed ID propagation is supported.

For further information, see [“IMS database service security process flow” on page 258](#).

IBM MQ service provider

- Client mode connections to a queue manager support TLS, including mutual authentication.
- User ID, or user ID and password authentication is supported for both client and bindings mode connections to a queue manager.
- ID propagation is supported for both client and bindings mode connections to a queue manager.

For further information, see [“Security requirements for the IBM MQ service provider” on page 273](#).

REST client service provider

- You can configure HTTPS on a REST client connection to secure your connection.
- Basic authentication for a REST client connection is supported.
- TLS client authentication for a REST client connection is supported.
- SAF ID propagation is supported through the use of PassTicket support when invoking Db2 RESTful services.

For further information, see [“Configuring security for a REST client connection” on page 287](#).

WOLA service provider

- You can use a user ID and password for authentication.
- If using WOLA with CICS, you can control whether the program link invocation transaction (BBO#) runs under the user ID of the link server transaction (BBO\$) or the propagated user ID from the z/OS Connect EE server.

For further information, see [Connection factory properties for optimized local adapters on Liberty and Liberty server transactions for CICS: BBOC, BBO\\$, and BBO#](#).

Other service providers

For other z/OS Connect EE service providers, for example, those from third party vendors, see the appropriate documentation for information on the security configurations supported.

API requester confidentiality and integrity

Learn how to maintain the confidentiality and integrity of the data that is handled by z/OS Connect EE.

Before you study this topic, you should be familiar with the information in [“Overview of z/OS Connect EE security” on page 331](#).

Confidentiality ensures that an unauthorized party cannot obtain the information in the transferred or stored data. Typically, confidentiality is achieved by encrypting the data.

Integrity ensures that transmitted or stored information is not altered in an unauthorized or accidental manner.

Securing communications with z/OS Connect EE

You can secure communications using the Transport Layer Security (TLS) protocol between

- CICS, IMS or a z/OS application, and a z/OS Connect EE server.
- The z/OS Connect EE server and the RESTful API endpoint.

[Figure 82 on page 395](#) shows the TLS connections that can be used in an API requester scenario.

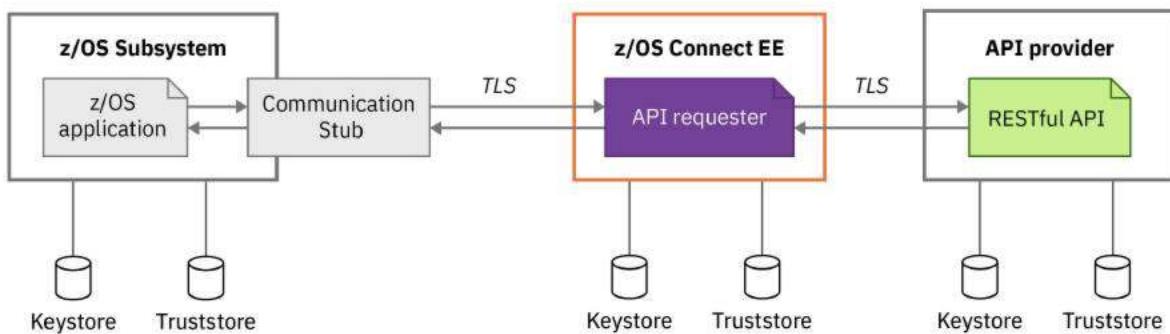


Figure 82. TLS connections in an API requester scenario.

TLS provides transport layer security that includes confidentiality, integrity, and authentication to secure the connection between a client and a z/OS Connect EE server.

z/OS Connect EE uses Java Secure Sockets Extension (JSSE) as the TLS implementation for secure connections. JSSE provides a framework and Java implementation that handles the handshake negotiation and protection capabilities provided by TLS. For more information on JSSE see [“Java Secure Sockets Extension \(JSSE\)” on page 460](#).

Alternatively you can use Application Transparent Transport Layer Security (AT-TLS), a capability of z/OS Communications Server, which can create a secure session on behalf of z/OS Connect EE (or other z/OS applications). Instead of implementing TLS in z/OS Connect EE, AT-TLS provides encryption and decryption of data based on policy statements that are coded in the Policy Agent. z/OS Connect EE sends and receives cleartext (unencrypted data) as usual while AT-TLS encrypts and decrypts data at the TCP transport layer. Note that if you configure an AT-TLS inbound policy to secure the connection into a z/OS Connect EE server, then client certificate authentication cannot be used.

For more information on AT-TLS, see [“Application Transparent Transport Layer Security \(AT-TLS\)” on page 460](#).

Securing communications to z/OS Connect EE

[Figure 83 on page 396](#) shows the TLS implementation options available for securing communications to a z/OS Connect EE server when using API requester.

- The HTTPS port 5002 is used for a TLS connection from CICS. The port is associated with an SSL configuration in the z/OS Connect EE server.
- The HTTPS port 5003 is protected with an AT-TLS outbound policy. The port is associated with an SSL configuration in the z/OS Connect EE server.
- The HTTP port 5004 is protected with both AT-TLS outbound and inbound policies and therefore the TLS connection is completely managed by AT-TLS. Note that client certificate authentication cannot be used for this connection.

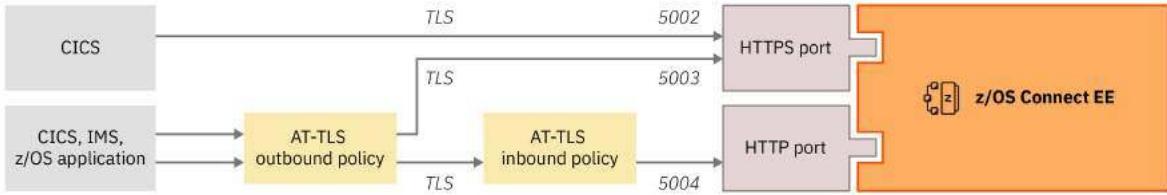


Figure 83. TLS implementation options available for API requester.

Securing communications to RESTful API endpoints

TLS can also be used to secure the connection between the z/OS Connect EE server and the RESTful API endpoint. The RESTful API endpoint determines if TLS is required, and if it requires the z/OS Connect EE server to authenticate itself with a personal certificate.

AT-TLS outbound policies can be configured to secure the TLS connection from the z/OS Connect EE server. If the RESTful API endpoint supports AT-TLS, then inbound policies can also be configured to secure the TLS connection into the RESTful API endpoint.

Configuring TLS for z/OS Connect EE resources

TLS configuration is required in the z/OS Connect EE server when requests come in on the HTTPS port.

The `requireSecure` attribute on the `zosconnect_zosConnectManager` and `apiRequester` elements of the `server.xml` configuration file, control whether a TLS connection is required for a specific request. The default value of the `requireSecure` attribute is `true`. You can set this value to `false` to remove the requirement for a TLS connection at a specific scope. When using AT-TLS inbound set the `requireSecure` attribute in your `server.xml` configuration file to `false`.

TLS server authentication is enabled by default, but you can also configure TLS client authentication, also called mutual TLS authentication. In this configuration, the client (for example, CICS) authenticates itself with a personal certificate.

In addition to the standard TLS behavior you can choose whether to also use client certificate authentication, by using the client certificate to establish the authenticated identity for the request. Alternatively, you can use TLS server authentication or mutual TLS authentication with basic authentication. For more information, see [“API requester authentication and identification” on page 407](#).

Configuring z/OS Connect EE SSL elements

If you set `requireSecure` attribute to `true` then the z/OS Connect EE server must be configured to use TLS.

z/OS Connect EE TLS is configured by using elements in the `server.xml` configuration file.

To configure TLS for connections from your CICS, IMS or z/OS application, an SSL configuration is associated with an HTTP endpoint (using the `httpEndpoint` element).

z/OS Connect EE includes a default SSL configuration (defined by the `ssl` element with `id` value of `defaultSSLConfig`). This default configuration is typically customized to add your own keystores and truststores, configure whether client authentication is required or supported, or whether only server authentication is required.

For TLS connections from your z/OS Connect EE server to the RESTful API endpoint, the default SSL configuration is used. You can override this default SSL configuration by specifying the `sslCertsRef` attribute on the `zosconnect_endpointConnection` element. For more information, see [“API requester TLS client authentication to a RESTful API endpoint” on page 418](#).

Further configuration information

For more information about how to configure CICS, IMS or a z/OS application, and the z/OS Connect EE server to use TLS or AT-TLS, follow the links below.

How to configure TLS from CICS

Configure a TLS connection from CICS to a z/OS Connect EE server.

This task is applicable when z/OS Connect EE is used as an API requester.

About this task

You can use TLS to secure your connection between CICS and your z/OS Connect EE server.

To configure a TLS connection from CICS to your z/OS Connect EE server you must customize the supplied URIMAP BAQURIMP specifying

- USAGE(CLIENT) because CICS is the HTTP client.
- SCHEME(HTTPS) to enable the HTTPS connection.
- PORT to specify the HTTPS port of the z/OS Connect EE server. See note.
- Optionally, CIPHERS(value), to specify cipher suites.

Note: If an AT-TLS policy is in place, for example, an outbound AT-TLS policy, then the schema in the URIMAP must be HTTP but still use the name of the HTTPS port of the z/OS Connect EE server. Otherwise, both CICS system SSL and AT-TLS attempt to perform a TLS handshake and the following error messages are displayed:

```
DFHS00123 05/11/2020 11:59:41 CICSZA51 Return code 410 received from function  
gsk_secure_socket_init of System SSL.  
Reason: Handshake abandoned by peer. Peer: <redacted ip address>, TCPIPSERVICE: *NONE*  
BAQT0008E 2020/05/11 11:59:41 Socket error.
```

If the z/OS Connect EE server requires TLS client authentication you must also customize the supplied URIMAP BAQURIMP to add

- CERTIFICATE(label), where label specifies the label of the X.509 certificate that is to be used as the client certificate during the TLS handshake.

For more information, see [CICS as an HTTP client: authentication and identification](#) and [URIMAP attributes](#) in the CICS documentation.

How to configure AT-TLS from IMS or a z/OS application

Configure an IBM z/OS Communications Server Application Transparent Transport Layer Security (AT-TLS) for secure connections from your IMS or z/OS application to z/OS Connect EE.

This task is applicable when z/OS Connect EE is used as an API requester.

About this task

To configure secure connections between the IMS or z/OS application and the z/OS Connect EE server:

- On the IMS or z/OS application side:
 1. Specify the hostname and the port number for z/OS Connect EE server in the BAQURI and BAQPORT parameters in the CEEOPTS DD statement. Ensure that the HTTPS port number of the z/OS Connect EE server is specified in BAQPORT. For more information, see [“Configuring IMS to access z/OS Connect EE for API calls” on page 309](#) or [“Configuring other z/OS applications to access z/OS Connect EE for API calls” on page 312](#).
 2. Set up AT-TLS, configure a key ring and policy files, as shown below.
- On the z/OS Connect EE server side, see [“How to configure TLS with RACF key rings” on page 400](#).

You must create a key ring to store the Certificate Authority (CA) root certificate and the server certificate that is signed by the (CA) private key. Store the CA certificate in the z/OS Connect EE truststore and the certificate in the server keystore. The following diagram shows the key ring, the server keystore, and the server truststore that are required for secure connections between the z/OS Connect EE server and the communication stub.

The following diagram shows the required Certificate Authority (CA) root certificate, the key ring, the server certificate, and server keystore on the z/OS Connect EE server.

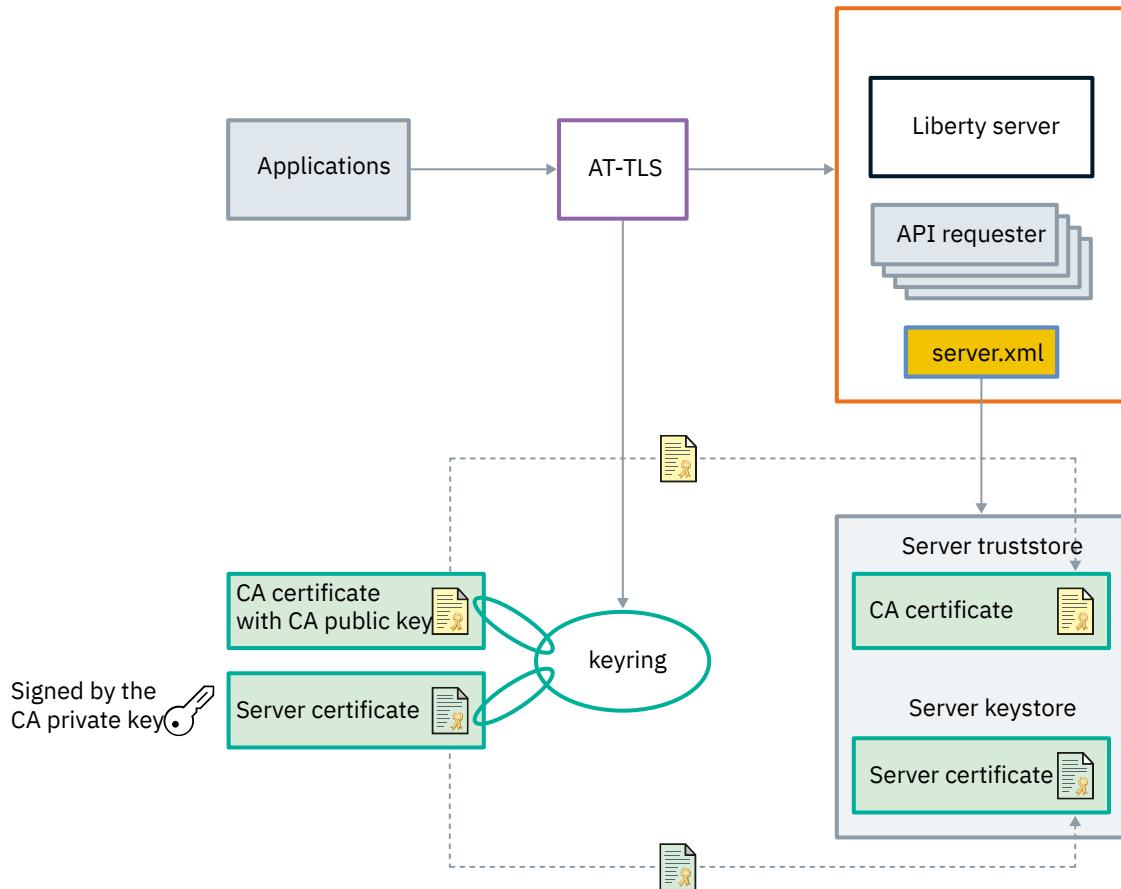


Figure 84. Secure connection configuration for IMS and z/OS applications to call API requesters

Procedure

The following steps demonstrate how to generate the Certificate Authority (CA) root certificate, the server certificate, and the key ring.

1. On the z/OS subsystem, use the RACDCERT command to generate the CA root certificate, the certificate, and the key ring.

- a) Create a key ring.

The following example creates a key ring, ZOSCONN-APIR-KEYRING.

```
/*
 * Create a key ring (ZOSCONN-APIR-KEYRING in this example).*/
 */
RACDCERT ADDRING(ZOSCONN-APIR-KEYRING) ID(OMVSADM)
```

- b) Create the CA root certificate for the server.

The following example generates a CA certificate and connect it to the key ring ZOSCONN-APIR-KEYRING.

```

/*
 *-----*/
/* Create a certificate to use as the Certificate Authority */
/* (CA), export to dataset, and connect to the key ring, */
/* ZOSCONN-APIR-KEYRING. */
/*-----*/
RACDCERT CERTAUTH GENCERT -
SUBJECTSDN(CN('IMSCA') OU('ZCEE') O('US IBM SVL') C('US')) -
KEYUSAGE(CERTSIGN) WITHLABEL('IMSCA')
RACDCERT CERTAUTH EXPORT(LABEL('IMSCA')) DSN(CERTAUTH.CERT)
RACDCERT CONNECT(CERTAUTH LABEL('IMSCA') -
RING(ZOSCONN-APIR-KEYRING)) ID(OMVSADM)

```

2. Create a server certificate, signed by the CA, for API requester clients to authenticate with the server.

In the following example, ZOSCONNCCERT is the label for the z/OS Connect EE server certificate, signed by the IMSCA certificate authority, in PKCS12 format.

```

/*
 *-----*/
/* Create z/OS Connect EE server certificate, export to */
/* dataset, and connect to ZOSCONN-APIR-KEYRING. */
/*-----*/
RACDCERT GENCERT -
SUBJECTSDN(CN('ZCEESERVER.MYCOM.COM') T('MY ZCEE SERVER') -
OU('ZCEE') O('US MYCOM COM') L('SANJOSE') SP('SANJOSE') C('US'))-
SIZE(2048) WITHLABEL('ZOSCONNCCERT') -
SIGNWITH(CERTAUTH LABEL('IMSCA')) ID(OMVSADM)
RACDCERT EXPORT(LABEL('ZOSCONNCCERT')) DSN(ZCONN.CERT.P12DER) -
FORMAT(PKCS12DER) PASSWORD('imsmobile') ID(OMVSADM)
RACDCERT CONNECT(LABEL('ZOSCONNCCERT') -
DEFAULT RING(ZOSCONN-APIR-KEYRING)) -
ID(OMVSADM)

/*
 *-----*/
/* Permit user IDs associated with server and client to */
/* access the certificates. */
/*-----*/
SETROPTS RACLIST(DIGTCERT) REFRESH
RDELETE FACILITY (IRR.DIGTCERT.LIST IRR.DIGTCERT.LISTRING)
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LIST UACC(NONE)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(OMVSADM) ACCESS(ALTER)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(OMVSADM) ACCESS(ALTER)
SETROPTS CLASSACT(FACILITY)
SETROPTS RACLIST(FACILITY) REFRESH
//
```

3. Exchange CA certificates with the z/OS Connect EE server.

Work with your z/OS Connect EE security administrator to

- Export the CA root certificate created for the IMS or z/OS application and add it to the z/OS Connect EE server's truststore. This will allow the z/OS Connect EE server to trust the certificate sent from the IMS or z/OS application.
- Export the z/OS Connect EE server's CA certificate and add it to the IMS or z/OS application RACF key ring. This will allow the IMS or z/OS application to trust the certificate sent from the z/OS Connect EE server.

For more information, see [“How to configure TLS with RACF key rings” on page 400](#).

The following steps demonstrate high-level AT-TLS configuration steps on the client application side:

4. Create Policy Agent files and store the files in z/OS UNIX System Services (USS) file system directory that can be accessed when you start the policy agent.
 - a) Create a Policy Agent main configuration file containing a TcpImage statement for the z/OS Connect EE server stack. Set the TcpImage statement to point to the image configuration file.
 - b) Create a Policy Agent image configuration file containing a TTLSConfig statement for the server stack. Set the TTLSConfig statement to point to the TTLSConfig policy file.
 - c) Create and configure a Policy Agent TTLSConfig policy file, and add the AT-TLS policy statements to this file. Most of the SSL settings are in this file. Set the security suite you want to support, and specify which ports to use SSL.
5. Activate the RACF SERVAUTH class using the TSO command: **SETROPTS CLASSACT(SERVAUTH)**

6. Set up InitStack access control:

- Define the EZB.INITSTACK.SYSNAME.TCPNAME profile for each AT-TLS stack.
- Permit administrative applications to use the stack before AT-TLS is initialized.

The following sample JCL sets up InitStack access control (based on the member EZARACF in sample data set SEZAINST):

```
//TLSRACF JOB MSGLEVEL=(1,1),USER=OMVSADM,PASSWORD=ALL1SDUN,  
// CLASS=A,MSGCLASS=A  
///*  
/*ROUTE PRINT THISCPU/IMSTST43  
// EXEC PGM=IKJEFT01  
//SYSTSPRT DD SYSOUT=*  
//SYSABEND DD SYSOUT=*  
//SYSTSIN DD *  
SETROPTS RACLIST (SERVAUTH)  
SETROPTS CLASSACT(SERVAUTH)  
SETROPTS GENERIC (SERVAUTH)  
RDEFINE SERVAUTH EZB.INITSTACK.SYSNAME.TCPNAME UACC(NONE)  
PERMIT EZB.INITSTACK.SYSNAME.TCPNAME CLASS(SERVAUTH) ID(*) ACCESS(READ) -  
WHEN(PROGRAM(PAGENT,EZAPAGEN))  
SETROPTS GENERIC(SERVAUTH) REFRESH  
SETROPTS RACLIST(SERVAUTH) REFRESH  
SETROPTS WHEN(PROGRAM) REFRESH  
//
```

7. Enable AT-TLS by adding the following lines to the SYS1.TCPPARMS(PROFILE) member.

```
;Enable AT-TLS Support  
TCPCONFIG TTLS
```

8. Restart TCP/IP, if it is currently running, by issuing **P TCPIP** and then **S TCPIP**.

9. Configure and start the syslog daemon (syslogd):

- Review your syslogd configuration (`/etc/syslog.conf`) to verify that messages written by the Policy Agent and TCP/IP stacks are saved in the intended files.

AT-TLS syslogd messages are written to the daemon facility by default. A sample configuration file is provided by z/OS in `/usr/lpp/tcpip/samples/syslog.conf`

- Start syslogd with the following command. Modify the path to `syslog.conf` based on your setup.

```
/usr/sbin/syslogd -f /etc/syslog.conf &
```

10. Start the policy agent with the following command. Modify the path to your Policy Agent file based on your setup.

```
/usr/sbin/pagent -c /etc/sysname_pagent.conf -l SYSLOGD &
```

Verify that the TCP/IP stack has received the AT-TLS policy and has issued console message EZZ4248E.

How to configure TLS with RACF key rings

Configure a TLS connection between CICS, IMS or a z/OS application and a z/OS Connect EE server.

This task is applicable when z/OS Connect EE is used as an API requester.

Before you begin

- Before you begin this task, you should be familiar with the information in [“API requester confidentiality and integrity” on page 394](#).
- You must have authorization to issue the following **RACDCERT** commands: **ADD**, **ADDRING**, **CONNECT**, **EXPORT**, **GENCERT**, **LIST**, and **LISTRING**, to create a RACF key ring and certificates. For more information about the **RACDCERT** commands and the authorizations that are required, see [RACDCERT \(Manage RACF digital certificates\) in the z/OS Security Server RACF Command Language Reference](#).
- You must have write access to the `server.xml` configuration file.

Note: In this topic, the term *client* refers to *CICS, IMS or a z/OS application*.

About this task

Configure a z/OS Connect EE server so that CICS, IMS, or z/OS applications must connect by using an HTTPS connection with TLS server authentication. This configuration creates the following artifacts:

- A certificate authority (CA) signed personal certificate for a z/OS Connect EE server to identify itself on TLS connections.
- A RACF key ring to act as the z/OS Connect EE server's keystore.
- A RACF key ring to act as the keystore for the CICS, IMS, or the z/OS application, so that it can trust the certificate presented by the z/OS Connect EE server on the TLS connection.

You can also configure TLS client authentication, sometimes referred to as mutual TLS authentication, to require the CICS, IMS, or z/OS application to provide its personal certificate on the connection. This configuration creates the following artifacts:

- A CA signed personal certificate for the CICS, IMS, or z/OS application to identify itself on TLS connections.
- A RACF key ring for the CICS, IMS, or z/OS application.
- A RACF key ring for the z/OS Connect EE server to use as its truststore, so that it can trust the certificate that is presented by the CICS, IMS, or z/OS application.

The certificates and key rings used in this task are described in [Figure 85 on page 402](#).

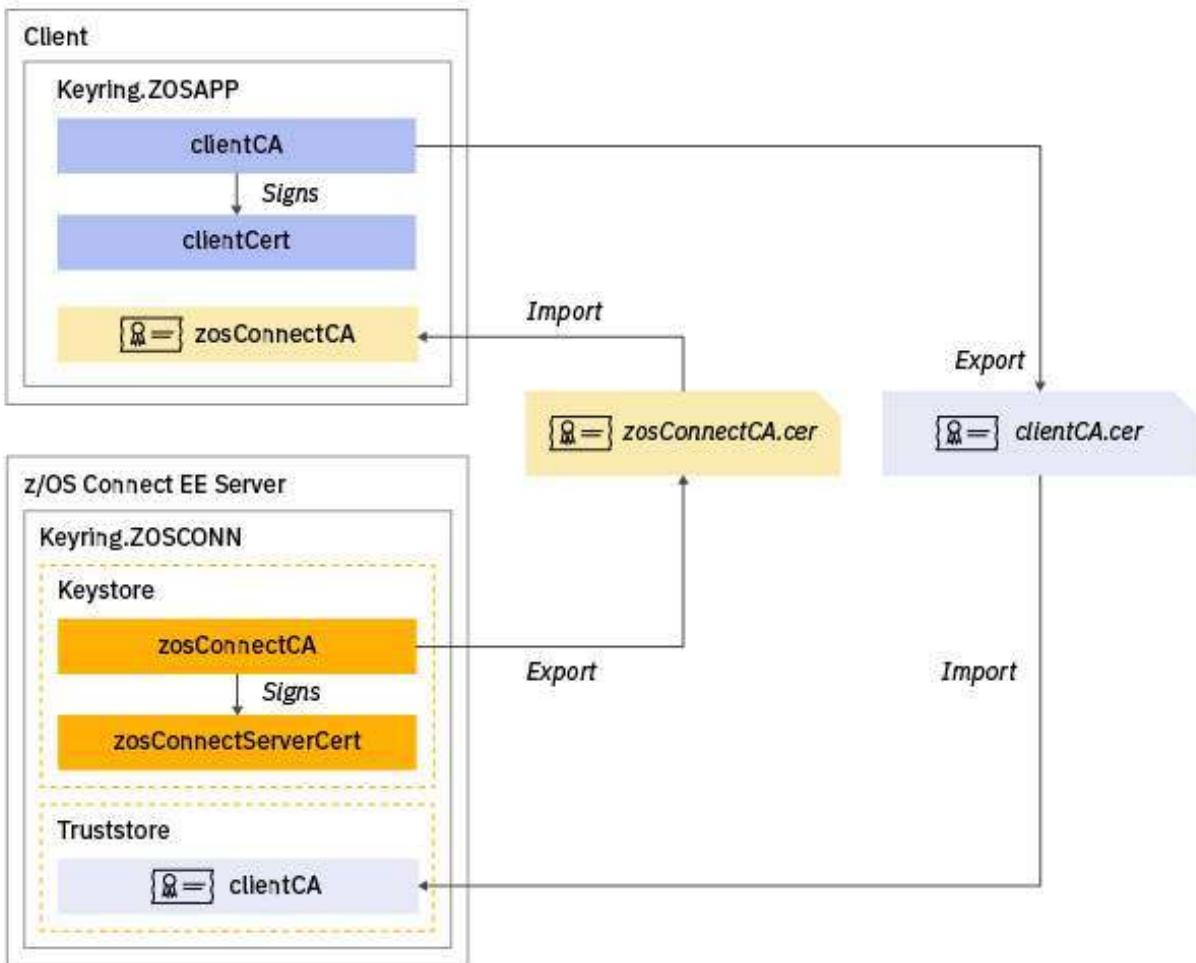


Figure 85. How certificates and key rings are used in this task

This task makes the following assumptions:

- RACF is the security manager. If you are using an alternative External Security Manager, refer to the appropriate product documentation for the equivalent commands.
- The CICS, IMS, or z/OS application, and the z/OS Connect EE server each uses a single RACF key ring for both their own keystores and truststores.
- The default z/OS Connect EE SSL elements in the `server.xml` configuration file are customized. This method is typical for configuring TLS on the z/OS Connect EE server's default HTTPS port. If you are configuring TLS for an additional HTTPS port, follow these instructions, but replace the default `id` attribute values of the `ssl` and `keyStore` elements with customized values. You would also need to associate the customized `ssl` element with the appropriate `httpEndpoint` element by configuring the `sslOptions` subelement. For example,

```
<httpEndpoint id="myHttpEndpoint" host="*"
  httpPort="9081" httpsPort="9444">
  <sslOptions sslRef="mySSLConfig" />
</httpEndpoint>
```

In this example, `mySSLConfig` is the `id` attribute value of the `ssl` element.

Note:

- This task covers TLS server authentication, and TLS client authentication (optional). It does not include the additional configuration to map a CICS, IMS, or z/OS application's personal certificate to a RACF

user ID to authenticate with a z/OS Connect EE server, but is a prerequisite to that task. For more information about configuring a CICS, IMS, or z/OS application certificate to authenticate with a z/OS Connect EE server, see [“How to configure client certificate authentication with RACF” on page 415](#).

- The term *label* is used by RACF, however the term *alias* is often used to reference the same artifact. Therefore in this documentation, the phrase *label or alias* is used for clarity.

Procedure

1. Create a RACF key ring for the z/OS Connect EE server to use as its keystore.

Enter the following command:

```
RACDCERT ID(ZCSERV1) ADDRING(Keyring.ZOSCONN)
```

The command uses the following values:

- ZCSERV1 is the user ID that owns the key ring. The user ID under which the z/OS Connect EE server runs must have READ access to this key ring. Either specify the server's user ID on the ID property or ensure that the server's user ID has READ access to the key ring owned by an alternative user ID.
- Keyring.ZOSCONN is the name of the key ring to be created.

2. Create a CA certificate for the z/OS Connect EE server.

- a) Create a self-signed RSA key pair to act as a CA certificate. A *key pair* consists of a public and private key.

Enter the following command:

```
RACDCERT GENCERT CERTAUTH SUBJECTSDN(CN('CA for zosConnect') O('IBM')  
OU('zosConnect') C('US')) SIZE(2048) WITHLABEL('zosConnectCA')  
NOTAFTER(DATE(2029-12-31))
```

The command uses the following values:

- CN('CA for zosConnect') O('IBM') OU('zosConnect') C('US') is an example distinguished name (DN) for the certificate.
- zosConnectCA is the label or alias of the certificate.
- 2029-12-31 is the expiry date of the certificate.

- b) Connect (add) the CA certificate to the key ring.

Enter the following command:

```
RACDCERT ID(ZCSERV1) CONNECT(RING(Keyring.ZOSCONN) LABEL('zosConnectCA')  
CERTAUTH)
```

The command uses the following values:

- ZCSERV1 is the user ID that owns the key ring.
- Keyring.ZOSCONN is the name of the key ring.
- zosConnectCA is the label or alias of the certificate to be connected to the key ring.

3. Create a personal certificate, signed by the CA certificate, for the z/OS Connect EE server.

- a) Create an RSA key pair for the z/OS Connect EE server signed by the CA certificate.

Enter the following command:

```
RACDCERT ID(ZCSERV1) GENCERT SUBJECTSDN(CN('myServer.host.com') O('IBM')  
OU('zosConnect') C('US')) SIZE(2048) SIGNWITH(CERTAUTH  
LABEL('zosConnectCA')) WITHLABEL('zosConnectServerCert')  
NOTAFTER(DATE(2029-12-31))
```

The command uses the following values:

- CN('myServer.host.com') O('IBM') OU('zosConnect') C('US') is an example distinguished name (DN) for the certificate. The common name (CN) value is typically the hostname of the z/OS LPAR that hosts the z/OS Connect EE server.
- zosConnectCA is the label or alias for the CA certificate that is used to sign the personal certificate.
- zosConnectServerCert is the label, or alias, for the personal certificate to be created and signed.

b) Connect (add) the personal certificate to the key ring.

Enter the following command:

```
RACDCERT ID(ZCSERV1) CONNECT(RING(Keyring.ZOSCONN)
LABEL('zosConnectServerCert'))
```

The command uses the following values:

- ZCSERV1 is the user ID that owns the key ring.
- Keyring.ZOSCONN is the name of the key ring.
- zosConnectServerCert is the label or alias of the personal certificate to be connected to the key ring.

4. Confirm that the key ring and certificates were created correctly.

a) List the certificates in the key ring.

Enter the following command:

```
RACDCERT ID(ZCSERV1) LISTRING(Keyring.ZOSCONN)
```

The following screen capture shows the expected response:

Ring: >Keyring.ZOSCONN<	Certificate Label Name	Cert Owner	USAGE	DEFAULT
	zosConnectCA	CERTAUTH	CERTAUTH	NO
	zosConnectServerCert	ID(ZCSERV1)	PERSONAL	NO

b) List the details of the CA certificate.

Enter the command:

```
RACDCERT CERTAUTH LIST(LABEL('zosConnectCA'))
```

The expected response contains the following information:

```
Issuer's Name:  
>CN=CA for zosConnect.OU=zosConnect.O=IBM.C=US<  
Subject's Name:  
>CN=CA for zosConnect.OU=zosConnect.O=IBM.C=US<
```

c) List details of the z/OS Connect EE server's personal certificate.

Enter the command:

```
RACDCERT ID(ZCSERV1) LIST(LABEL('zosConnectServerCert'))
```

The expected response contains the following information:

```
Issuer's Name:  
>CN=CA for zosConnect.OU=zosConnect.O=IBM.C=US<  
Subject's Name:  
>CN=myServer.host.com.OU=zosConnect.O=IBM.C=US<
```

5. Configure the HTTPS port for the z/OS Connect EE server.

Identify an unused TCP/IP port on your z/OS LPAR, which can be used as the HTTPS port for z/OS Connect EE. Specify the port value on the httpsPort attribute of the httpEndpoint element in the server.xml configuration file.

For example, to use 9443 as the default HTTPS port, add the following element to the configuration:

```
<httpEndpoint id="defaultHttpEndpoint" host="*"  
    httpPort="9080" httpsPort="9443"/>
```

6. Configure the z/OS Connect EE server to require HTTPS requests.

Set the attribute requireSecure="true" in the server.xml configuration file to force all requests at that scope to use an HTTPS connection. This attribute can be set at different scopes:

- To require an HTTPS connection globally for the server, set requireSecure="true" on the zosconnect_zosConnectManager element in the server.xml configuration file. For example,

```
<zosconnect_zosConnectManager requireSecure="true" ... />
```

- To require an HTTPS connection for a specific API requester, set requireSecure="true" on the apiRequester element in the server.xml configuration file. For example,

```
<zosconnect_apiRequesters>  
    <apiRequester name="Stock_Control" requireSecure="true"/>  
</zosconnect_apiRequesters>
```

This setting takes precedence over the global setting.

For more information about these elements, see [“Configuration elements” on page 753](#) in the *Reference* section.

7. Configure the SSL configuration to be used by the z/OS Connect EE server.

Create an ssl repertoire element in the server.xml configuration file, with default values for the id and keyStoreRef attributes. For example,

```
<ssl id="defaultSSLConfig"  
    keyStoreRef="defaultKeyStore" />
```

For more information about the ssl repertoire element, see the [Server configuration](#) section in the *IBM WebSphere Application Server for z/OS Liberty* documentation.

The server now has only a keystore. Because it does not yet have any trusted certificates, it does not require a truststore.

8. Configure the z/OS Connect EE server to reference the RACF key ring that contains the server's personal certificate and CA certificate.

Create a keyStore element in the server.xml configuration file. For example,

```
<keyStore id="defaultKeyStore"  
    fileBased="false"  
    location="safkeyring:///Keyring.ZOSCONN"  
    password="password"  
    readOnly="true"  
    type="JCERACFKS" />
```

The element uses the following values:

- defaultKeyStore must match the value that is specified on the keyStoreRef attribute of the ssl element.
- The location value must be the RACF key ring that acts as the server's keystore.
- The password attribute is mandatory, so a value must be specified. However, the value is not used when type="JCERACFKS" because RACF key rings are not secured with passwords.

9. Exchange the z/OS Connect EE server's CA certificate with the CICS, IMS or z/OS application.

Work with your security administrator of the CICS, IMS or z/OS application to export the z/OS Connect EE server's CA certificate and add it to the CICS, IMS or z/OS application RACF key ring. This

will allow the CICS, IMS or z/OS application to trust the certificate sent from the z/OS Connect EE server.

For CICS, see [“How to configure TLS from CICS” on page 397](#).

For IMS or a z/OS application, see [“How to configure AT-TLS from IMS or a z/OS application” on page 397](#).

The following steps are optional, and are only required to configure TLS client authentication. This is also called mutual TLS authentication.

10. Exchange the CA certificate of the CICS, IMS or z/OS application with the z/OS Connect EE server.

The CICS, IMS or z/OS application must create a personal certificate, signed by a CA certificate, and stored in a RACF key ring.

Work with your security administrator of the CICS, IMS or z/OS application to export the CICS, IMS or z/OS application's CA certificate and add it to the z/OS Connect EE server's truststore. This will allow the z/OS Connect EE server to trust the certificate sent from the CICS, IMS or z/OS application.

For CICS, see [“How to configure TLS from CICS” on page 397](#).

For IMS or a z/OS application, see [“How to configure AT-TLS from IMS or a z/OS application” on page 397](#).

11. Edit the SSL configuration to be used by the z/OS Connect EE server.

Edit the existing `ssl` repertoire element in the `server.xml` configuration file, to add a `trustStoreRef` attribute with the default value. Set client authentication as required by setting `clientAuthentication="true"`. For example,

```
<ssl id="defaultSSLConfig"
    keyStoreRef="defaultKeyStore"
    trustStoreRef="defaultTrustStore"
    clientAuthentication="true" />
```

12. Create a `keyStore` element in the `server.xml` configuration file for the z/OS Connect EE server's truststore.

The `keyStore` element is also used for truststores. For example,

```
<keyStore id="defaultTrustStore"
    fileBased="false"
    location="safkeyring:///Keyring.ZOSCONN"
    password="password"
    readOnly="true"
    type="JCERACFKS" />
```

The element uses the following values:

- `defaultTrustStore` must match the value that is specified on the `trustStoreRef` attribute of the `ssl` element.
- The `location` value must be the RACF key ring that acts as the server's truststore.
- The `password` attribute is mandatory, so a value must be specified. However, the value is not used when `type="JCERACFKS"` because RACF key rings are not secured with passwords.

13. Start, or restart the server if it was already running, to pick up the changes.

The `messages.log` file should contain the following message:

```
CWWK00219I: TCP Channel defaultHttpEndpoint-ssl has been started and is now listening for requests on host * (IPv6) port 9443
```

The message uses the following values:

- `defaultHttpEndpoint-ssl` is the `id` attribute value of the `httpEndpoint` element followed by `-ssl`.
- `*` is the value of the `httpEndpoint` element `host` attribute.
- `9443` is the value of the `httpEndpoint` element `httpsPort` attribute.

Results

The z/OS Connect EE server is now configured to require HTTPS requests on its default HTTPS port. Optionally, TLS client authentication is also enabled. The CICS, IMS or z/OS application can connect to the z/OS Connect EE server using a TLS connection.

API requester authentication and identification

Learn how requests in API requester scenarios are authenticated and identified.

Before you study this topic, you should be familiar with the information in [“Overview of z/OS Connect EE security” on page 331](#).

Requests can be authenticated

- Between the CICS, IMS or the z/OS application, and the z/OS Connect EE server.
- Between the CICS, IMS or the z/OS application, and the RESTful API endpoint, passing the credentials through the z/OS Connect EE server (for example, when using OAuth 2.0, API key or JSON Web Token).
- Between the z/OS Connect EE server and the RESTful API endpoint.

When implementing authentication and identification, you can consider

- The authentication options (see [“API requester authentication options” on page 407](#)).
- User registries. These store information about users and groups that can be used for authentication and authorization. Typically a System Authorization Facility (SAF) registry is used with z/OS Connect EE, although z/OS Connect EE also supports Basic user registry and Lightweight Directory Access Protocol (LDAP) user registry. For more information about configuring user registries, see [User registries](#).
- Caching authentication credentials. An authentication cache is provided to store a subject after successful authentication of a user to reduce the potential performance impact of creation of a subject. For more information, see [Configuring the authentication cache in Liberty](#) in the WebSphere Application Server for z/OS Liberty documentation.
- Identity assertion. If you want to invoke an API requester from a z/OS application by using the ID that is provided in the application context, you can use the identity assertion function. For more information, see [“Identity assertion for API requesters” on page 441](#).

By default, z/OS Connect EE requires that all requests are authenticated. Successful authentication using any of the supported authentication methods results in an authenticated user ID being associated with the request. The authenticated user ID is also checked to ensure that it is authorized to access z/OS Connect EE.

Authentication is governed by the `requireAuth` attribute of the `zosconnect_zosConnectManager` element in the `server.xml` configuration file. If this attribute is set to `true`, all requests to a z/OS Connect EE server are authenticated. You can override this global setting by specifying the `requireAuth` attribute on the `zosconnect_apiRequesters` for all API requesters or on the `apiRequester` element for a particular API requester. The setting for a particular requester takes precedence over the setting for all API requesters.

API requester authentication options

Learn about several types of authentication that are supported by z/OS Connect EE.

Consider the type of authentication that you require between the CICS, IMS or z/OS application and a z/OS Connect EE server. The authentication required between the z/OS Connect EE server and the RESTful API is determined by the RESTful API.

API requester basic authentication to z/OS Connect EE

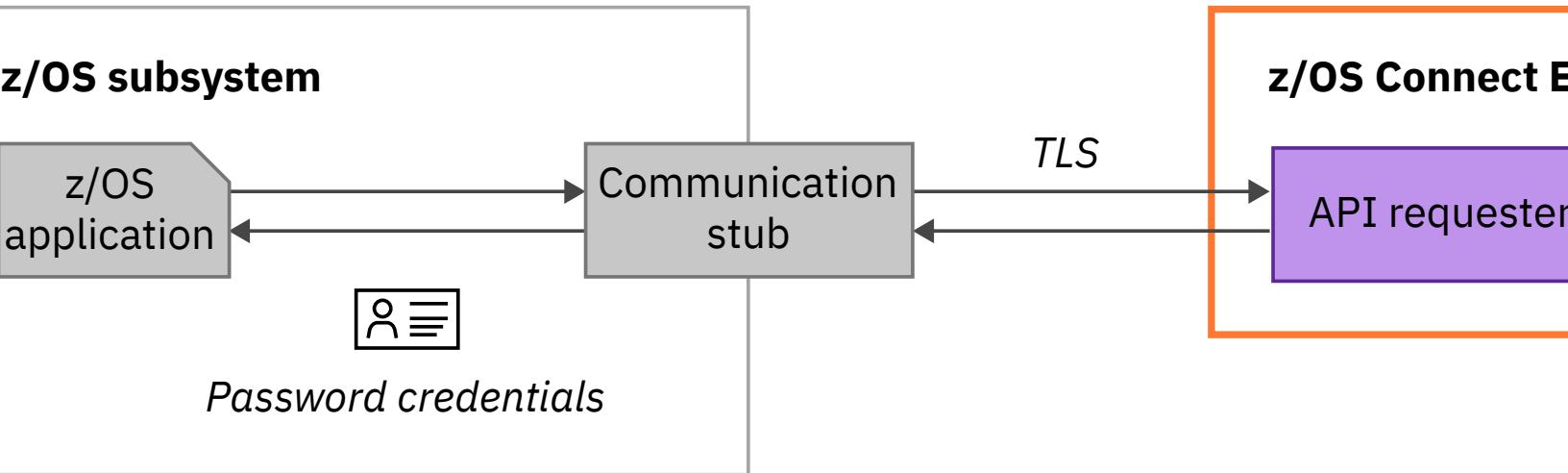
Basic authentication can be used between the CICS, IMS, or a z/OS application and the z/OS Connect EE server.

Basic authentication is a simple authentication scheme that is built into the HTTP protocol. It requires the CICS, IMS, or a z/OS application to provide a user ID and password in the request.

- For IMS or a z/OS application, a user ID and password must be sent by specifying the values for the BAQUSERNAME and BAQPASSWORD environment variables in the CEEOPTS DD statement.
- For CICS, a user ID and password must be sent by using the CICS XWBAUTH user exit.

The z/OS Connect EE server validates the user ID and password against a configured user registry. The user ID is set as the authenticated user.

The following diagram shows basic authentication between CICS, IMS or a z/OS application, and a z/OS Connect EE server.



Typically a SAF user ID and password are provided by the CICS, IMS, or the z/OS application for basic authentication when z/OS Connect EE acts as an API requester. Alternatively an LDAP distinguished name (or uid) and password, or a basic user registry user ID and password can be used.

When basic authentication is used, the credentials are encoded, but are not encrypted. Therefore, it is typically used with HTTPS (TLS) to provide confidentiality.

By default, z/OS Connect EE uses client certificate authentication. You can use one of the following methods to implement basic authentication:

- Configure z/OS Connect EE to fail over to use basic authentication when the client certificate authentication does not succeed. For example, when the client does not send a certificate or when the client sends a certificate but the certificate is not mapped to a user ID in the user registry.

To configure fail over to basic authentication, add the following element to the `server.xml` configuration file:

```
<webAppSecurity allowFailOverToBasicAuth="true"/>
```

- Configure z/OS Connect EE to override the client certificate authentication default. However, this configuration applies globally, so this option is not suitable when any requests to the same z/OS Connect EE server require client certificate authentication. This option also provides improved performance. For more information, see [“Performance best practices for a z/OS Connect EE production system” on page 205](#).

To configure z/OS Connect EE to override the client certificate authentication default with basic authentication, add the following element to the `server.xml` configuration file:

```
<webAppSecurity overrideHttpAuthMethod="BASIC"/>
```

How to configure basic authentication from CICS

Configure basic authentication from CICS to a z/OS Connect EE server.

This task is applicable when z/OS Connect EE is used as an API requester.

About this task

To use basic authentication to authenticate your CICS application you must

- Use the CICS XWBAUTH user exit to specify credentials (user ID and password) to be sent to z/OS Connect EE.
- Customize the supplied URIMAP BAQURIMP specifying
 - USAGE(CLIENT) because CICS is the HTTP client.
 - AUTHENTICATE(BASIC) to indicate that the user ID and password credentials are to be provided by the XWBAUTH global user exit.

Note that your CICS application should not provide credentials in an EXEC CICS WEB SEND or WEB CONVERSE command as this is not supported by z/OS Connect EE and will be ignored.

For more information, see [CICS as an HTTP client: authentication and identification](#), [HTTP client send exit XWBAUTH](#) and [URIMAP attributes](#) in the CICS documentation.

How to configure basic authentication from IMS or a z/OS application

Configure basic authentication from IMS or a z/OS application to a z/OS Connect EE server.

This task is applicable when z/OS Connect EE is used as an API requester.

About this task

You configure basic authentication in an IMS or z/OS application by specifying the environment variables BAQUSERNAME and BAQPASSWORD in the CEEOPTS DD statement.

Procedure

Specify the user ID and password on the environment variables BAQUSERNAME and BAQPASSWORD in the CEEOPTS DD statement. For example,

```
CEEPTS DD *
  POSIX(ON),
  ENVAR("BAQURI=MYSERVER.MY.COM", "BAQPORT=10053", "BAQUSERNAME=OMVSADM",
"BAQPASSWORD=IMSMOBILE")
```

How to configure basic authentication with a SAF user registry

Configure a z/OS Connect EE server to perform basic authentication with a SAF user registry.

This task is applicable when z/OS Connect EE is used as an API requester.

Before you begin

- You should be familiar with the information in [“API requester authentication and identification” on page 407](#).
- You must complete the task [“How to activate and configure the SAF user registry” on page 456](#) to configure the z/OS Connect EE server to use z/OS authorized services and a SAF user registry.
- You must have write access to the `server.xml` configuration file.

About this task

You configure the z/OS Connect EE server to require authentication, by setting the attribute `requireAuth="true"`. This task then configures the server to use basic authentication.

This task does not include information on how to configure the z/OS Connect EE server to use TLS. If the attribute `requireSecure` is set to `true` (the default), you must configure a TLS connection between the client and the z/OS Connect EE server, for example, by completing the task [“How to configure TLS with RACF key rings” on page 400](#).

Procedure

For more information about configuration elements, see [“Configuration elements” on page 753](#) in the *Reference* section.

1. Ensure that the server is configured to require authentication for the request.

This configuration can be set at different scopes in the `server.xml` configuration file:

- To require authentication globally for the server, set `requireAuth="true"` on the `zosconnect_zosConnectManager` element. For example,

```
<zosconnect_zosConnectManager requireAuth="true" ... />
```

- To require authentication for all API requesters, which takes precedence over the global setting, set `requireAuth="true"` on the `zosconnect_apiRequesters` element. For example,

```
<zosconnect_apiRequesters requireAuth="true">
  <apiRequester ... />
</zosconnect_apiRequesters>
```

- To require authentication for a specific API requester, which has the highest precedence, set `requireAuth="true"` on the `apiRequester` element. For example,

```
<zosconnect_apiRequesters>
  <apiRequester name="Stock_Control" requireAuth="true"/>
</zosconnect_apiRequesters>
```

Important: When the `requireAuth` attribute is specified at more than one scope, the value set on the `apiRequester` element takes precedence over the value set on the `zosconnect_apiRequesters` element, which takes precedence over the value on the `zosconnect_zosconnectManager` element.

2. Configure the server to use basic authentication.

z/OS Connect EE attempts to use a TLS client certificate for authentication, unless an alternative authentication mechanism is configured. Use one of the following methods to configure basic authentication:

- Configure fail-over to basic authentication, by adding the following element to the `server.xml` configuration file:

```
<webAppSecurity allowFailOverToBasicAuth="true" />
```

- Configure basic authentication to override the client certificate authentication default, by adding the following element to the `server.xml` configuration file:

```
<webAppSecurity overrideHttpAuthMethod="BASIC" />
```

3. Assign users and groups to the `zosConnectAccess` role.

Follow the instructions in task [“How to configure the `zosConnectAccess` role with a SAF user registry”](#) on page 449.

Results

User IDs and groups in the SAF user registry can be used to authenticate with the z/OS Connect EE server. Additionally, the SAF user IDs and groups that are assigned to the `zosConnectAccess` role now have authorization to access z/OS Connect EE.

API requester client certificate authentication to z/OS Connect EE

Client certificate authentication can be used between the CICS, IMS or a z/OS application and the z/OS Connect EE server.

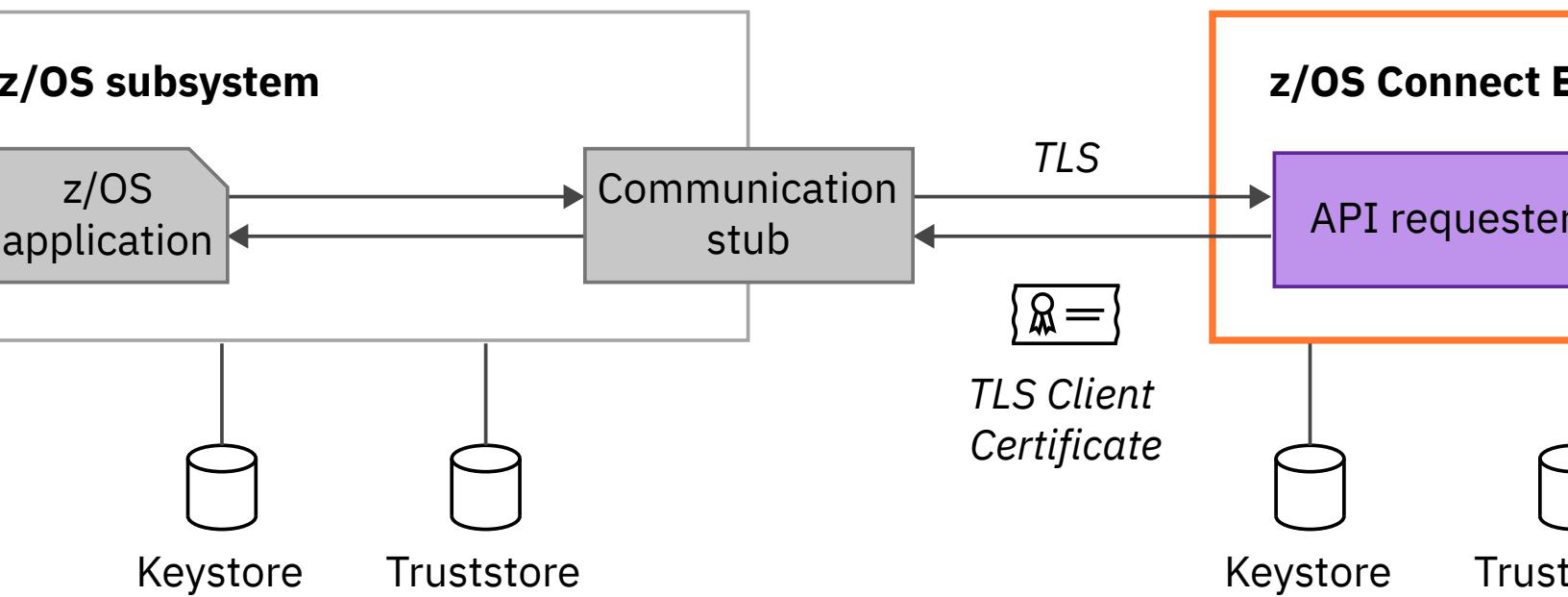
Client certificate authentication uses information that is provided in the CICS, IMS or the z/OS application's TLS certificate to map to an associated user ID. It also provides all of the normal benefits that are associated with a secure TLS connection.

Client certificate authentication requires the CICS, IMS or a z/OS application to send a certificate on the request.

- For IMS or a z/OS application, an AT-TLS outbound policy must be configured.
- For CICS, either an AT-TLS outbound policy can be configured, or CICS native TLS support can be used.

Note: If you configure an AT-TLS inbound policy to secure the connection into a z/OS Connect EE server, then client certificate authentication cannot be used.

[Figure 86 on page 413](#) shows client certificate authentication between a CICS, IMS, or z/OS application, and z/OS Connect EE.



When TLS client authentication is required by the z/OS Connect EE server, the z/OS Connect EE server must be configured with an HTTPS port. For each HTTPS connection, the z/OS Connect EE server asks the CICS, IMS or the z/OS application to provide its certificate and validates the chain of trust by using its truststore. That is, it validates that the client certificate issuer is trusted. This is standard TLS behavior and if the CICS, IMS or the z/OS application certificate is successfully validated, the connection can be established to the z/OS Connect EE server.

Mapping the client certificate to a user ID

To authenticate to the z/OS Connect EE server, the CICS, IMS or the z/OS application certificate must also be mapped to a user ID in the user registry. If the certificate is successfully mapped to a user ID, then that user ID is set as the authenticated user.

Typically a System Authorization Facility (SAF) registry is used with z/OS Connect EE, although z/OS Connect EE also supports Basic user registry and Lightweight Directory Access Protocol (LDAP) user registry. For more information about configuring the user registries, see [User registries](#).

If using a SAF registry, a DIGTCERT profile is generated from the information in the certificate, such as the certificate's serial number and the issuer's distinguished name. The profile must be associated with a SAF user ID. Client certificates can be associated with a RACF user ID in two ways:

- Use the RACDCERT MAP command to define a certificate name filter, which is also called a user ID mapping. This maps the certificate subject's distinguished name (DN) to a RACF user ID. Certificate name filtering supports generic filters allowing multiple certificates to be associated with a single RACF user ID.
- Use the RACDCERT ADD command to add the certificate into RACF and specify the user ID it is to be associated with.

For more information, see [RACDCERT ADD](#) (add certificate) and [RACDCERT MAP](#) (create mapping) in the z/OS Security Server RACF Command Language Reference.

If using a basic registry, the user identity is the common name (CN) from the distinguished name (DN) of the certificate. For more information about using client authentication with a basic registry, see [Basic certificate map mode](#) in the WebSphere Application Server for z/OS Liberty documentation.

If using an LDAP registry, the DN from the client certificate must be in the LDAP registry. For more information about using client authentication with LDAP, see [LDAP certificate map mode](#) in the WebSphere Application Server for z/OS Liberty documentation.

Configuring CICS, IMS or a z/OS application for client certificate authentication

The CICS, IMS or z/OS application must first be configured to use a TLS connection with TLS client authentication enabled to the z/OS Connect EE server. Work with your z/OS Connect EE security administrator to agree the distinguished names of the client certificates to be sent on the requests, and the RACF user IDs they should be mapped to.

For more information, see [“How to configure TLS from CICS” on page 397](#) or [“How to configure AT-TLS from IMS or a z/OS application” on page 397](#).

Configuring z/OS Connect EE for client certificate authentication

The z/OS Connect EE server is configured for client certificate authentication using an SSL configuration. The `ssl` element in the `server.xml` configuration file must set attribute `clientAuthentication` to `true` to request the client to provide a certificate.

The SSL configuration associated with the HTTP endpoint element affects all requests to the HTTPS port. If some of your requests do not require client authentication then you can configure multiple ports to allow for different SSL configurations.

How to configure client certificate authentication with RACF

Configure a z/OS Connect EE server to perform authentication of the identity in a TLS client certificate, mapping the certificate to a RACF user ID, and then granting that user ID authority to access z/OS Connect EE resources.

This task is applicable when z/OS Connect EE is used as an API requester.

Before you begin

- You should be familiar with the information in [“API requester authentication and identification” on page 407](#).
- You must have completed the task [“How to activate and configure the SAF user registry” on page 456](#) to configure the z/OS Connect EE server to use z/OS authorized services and a SAF user registry.
- You must have configured a TLS connection between the CICS, IMS or a z/OS application, and the z/OS Connect EE server with TLS client authentication enabled. For example, by completing the steps in the task [“How to configure TLS with RACF key rings” on page 400](#), and [“How to configure TLS from CICS” on page 397](#) or [“How to configure AT-TLS from IMS or a z/OS application” on page 397](#).
- You need to know the subject value of the client certificate to be mapped.
- You need to know the user ID to which the TLS client certificate will be mapped, and this user ID must exist and have an OMVS segment.
- You must have authorization to issue the RACDCERT MAP command. For more information about the RACDCERT commands and the authorizations that are required, see [RACDCERT \(Manage RACF digital certificates\)](#) in the *z/OS Security Server RACF Command Language Reference*.
- You must have write access to the `server.xml` configuration file.

About this task

This task assumes that RACF is used as security manager. If you are using an alternative External Security Manager, refer to the appropriate product documentation for the equivalent commands.

You use RACF certificate name filtering, also called user ID mapping, to map the TLS client certificate to a RACF user ID. You then configure the z/OS Connect EE server to require authentication, by setting the attribute `requireAuth="true"`.

During authentication, the z/OS Connect EE server will call RACF to perform the mapping resulting in the mapped RACF user ID being the authenticated user ID.

Procedure

1. Activate the RACF DIGTNMAP class to allow certificate name filters to be created or changed.

Enter the following RACF command:

SETROPTS CLASSACT(DIGTNMAP) RACLST(DIGTNMAP)

2. Map the TLS client certificate to a RACF user ID.

Enter the following command to use RACF certificate name filtering to map the client certificate to a RACF user ID.

**RACDCERT MAP ID(EMPLOY1) SDNFILTER('CN=myClient.host.com.O=IBM.C=US')
WITHLABEL('ClientCertEMPLOY1')**

The command uses the following values:

- EMPLOY1 is the RACF user ID to which the client certificate is to be mapped.
- CN=myClient.host.com.O=IBM.C=US is the subject distinguished name filter which corresponds to the client certificate subject's distinguished name value of CN=myClient.host.com, O=IBM, C=US. The syntax of the SDNFILTER is significant, use periods to separate the components of the distinguished name and remove any spaces between DN components.
- ClientCertEMPLOY1 is a label for the mapping.

For the full syntax of the RACDCERT MAP command, see [RACDCERT MAP \(Create mapping\)](#) in the z/OS Security Server RACF Command Language Reference.

3. Refresh the DIGTNMAP RACF class.

For the changes to take effect, enter the following RACF command:

SETROPTS RACLIST(DIGTNMAP) REFRESH

4. Ensure that the server is configured to require authentication for the request.

This can be set at various scopes in the `server.xml` configuration file:

- To require authentication globally for the server, set `requireAuth="true"` on the `zosconnect_zosConnectManager` element. For example,

```
<zosconnect_zosConnectManager requireAuth="true" ... />
```

- To require authentication for all API requesters, which takes precedence over the global setting, set `requireAuth="true"` on the `zosconnect_apiRequesters` element. For example,

```
<zosconnect_apiRequesters requireAuth="true">
  <apiRequester ... />
</zosconnect_apiRequesters>
```

- To require authentication for a specific API requester, which has the highest precedence, set `requireAuth="true"` on the `apiRequester` element. For example,

```
<zosconnect_apiRequesters>
  <apiRequester name="Stock_Control" requireAuth="true" />
</zosconnect_apiRequesters>
```

Important: When the `requireAuth` attribute are specified at more than one scope, the value set on the `apiRequester` element takes precedence over the value set on the `zosconnect_apiRequesters` element, which takes precedence over the value on the `zosconnect_zosConnectManager` element.

For more information about configuration elements, see [“Configuration elements” on page 753](#) in the *Reference* section.

5. Assign the mapped RACF user ID to the `zosConnectAccess` role.

Follow the instructions in the task [“How to configure the `zosConnectAccess` role with a SAF user registry” on page 449](#).

6. Ensure the Liberty profile angel process is running.

To use z/OS authorized services for SAF authentication and authorization, the Liberty profile angel process must be running for the server to connect to. In one of the prerequisite tasks listed in the “Before you begin” section, you created a started task to run the Liberty angel process and granted permission for the z/OS Connect EE server to access it.

To start the angel process, start the associated started task. Enter the SDSF command,

/S BAQZANGL

For more information about starting the angel process and checking that it started successfully, see [Start the Angel process as a started task](#) in *Configuring the Liberty Angel process and z/OS authorized services*.

7. Start, or restart the server if it was already running, to pick up the changes made to the RACF class profiles.

Results

The TLS client certificate is mapped to a RACF user ID, and is authorized to access z/OS Connect EE.

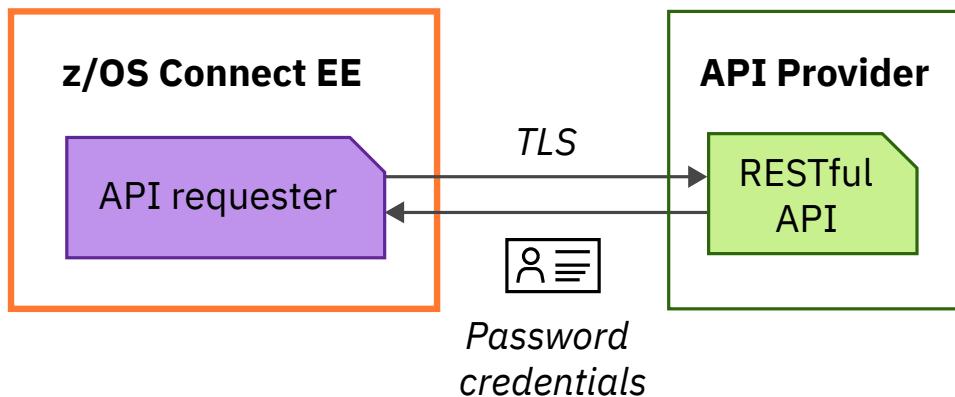
What to do next

You might now be interested in configuring more specific authorization. See the task “[How to configure authorization levels with a SAF user registry](#)” on page 450.

API requester basic authentication to a RESTful API endpoint

Basic authentication can be used between the z/OS Connect EE server and the RESTful API endpoint.

You can specify credentials for basic authentication for connections between the z/OS Connect EE server and the RESTful API endpoint by using the `zosconnect_authData` element in the `server.xml` configuration file. These credentials are different to those used for basic authentication between the CICS, IMS or z/OS application, and the z/OS Connect EE server.



Configuring z/OS Connect EE elements for basic authentication

The following excerpt from a `server.xml` configuration file shows an example of how to configure basic authentication for the Watson RESTful API.

```
<zosconnect_authData id="zosconnectBasicAuthConfig"
    password="passw0rd"
    user="Fred"/>

<zosconnect_endpointConnection id="watson-api-explorer"
    host="http://watson-api-explorer.mybluemix.net"
    port="80"
    authenticationConfigRef="zosconnectBasicAuthConfig"/>
```

To configure z/OS Connect EE to call the RESTful API using basic authentication, you must configure the `zosconnect_authData` and `zosconnect_endpointConnection` elements.

On the `zosconnect_authData` element you can use the WebSphere® Liberty profile `securityUtility encode` command to encode the password for each user. The `securityUtility` command-line tool is available in the `<installation_path>/wlp/bin` directory. For more information, see [securityUtility command](#) in the *WebSphere Application Server for z/OS Liberty* documentation.

On the `zosconnect_endpointConnection` element

- a host, specifying the HTTP or HTTPS protocol. If you choose to use the HTTPS protocol, you must also specify the `sslCertsRef` attribute and configure TLS.
- a port, specifying the target port for the API provider.

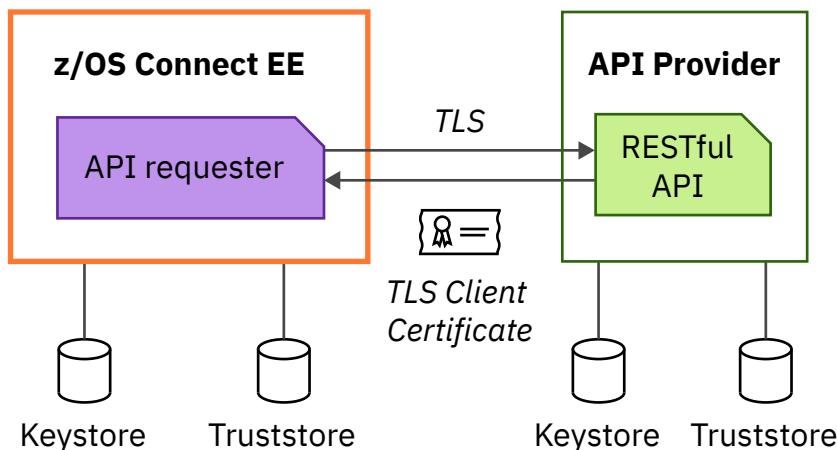
- an authenticationConfigRef, to reference the basic authentication configuration. This attribute should be used in preference to basicAuthRef which is now deprecated.

For more information about the elements shown in the example above, see “[Configuration elements](#)” on page 753 in the Reference section.

API requester TLS client authentication to a RESTful API endpoint

TLS client authentication can be used to secure communications between the z/OS Connect EE server and the RESTful API endpoint.

When a request is made to establish a TLS connection from the z/OS Connect EE server to the API provider, an SSL handshake is initiated. The API provider requests a client certificate from the z/OS Connect EE server, which in turn supplies the certificate.



TLS client authentication, also known as mutual TLS authentication, requires the following preparation before an SSL handshake can take place.

- The API provider's keystore must contain the certificate it uses to identify itself.
- The Certificate Authority which issued the API provider's certificate, must be added to the z/OS Connect EE server's truststore so that the z/OS Connect EE server can trust the API provider.
- The z/OS Connect EE server's keystore must contain a certificate to identify itself.
- The Certificate Authority which issued the certificate for the z/OS Connect EE server, must be added to the API provider's truststore so that the API provider can trust the z/OS Connect EE server.

The API provider may additionally use the client certificate to establish an authenticated identity which is used to run the RESTful API.

Configuring z/OS Connect EE SSL elements for TLS client authentication

z/OS Connect EE TLS is configured by using elements in the `server.xml` configuration file. z/OS Connect EE includes a default SSL configuration. This default configuration is typically customized to add your own keystores and truststores. The following excerpt from a `server.xml` configuration file that uses a SAF key ring shows an example of how to configure the TLS client authentication for the Watson RESTful API.

```

<keyStore id="zosconnectKeyStore"
    fileBased="false"
    location="safkeyring:///Keyring.ZOSCONN"
    password="password"
    readOnly="true"
    type="JCERACFKS" />
    
```

```

<keyStore id="zosconnectTrustStore"
  fileBased="false"
  location="safkeyring:///Keyring.ZOSCONN"
  password="password"
  readOnly="true"
  type="JCERACFKS" />

<ssl id="zosconnectSSLConfig" keyStoreRef="zosconnectKeyStore"
  trustStoreRef="zosconnectTrustStore"/>

<zosconnect_endpointConnection id="watson-api-explorer"
  host="https://watson-api-explorer.mybluemix.net"
  port="443"
  sslCertsRef="zosconnectSSLConfig"/>

```

To configure z/OS Connect EE to use a TLS connection to the API provider, you must configure the `zosconnect_endpointConnection` element with

- a host, specifying the https protocol
- a port, specifying the HTTPS port for the API provider
- a `sslCertsRef`, to reference the SSL configuration. This references the keystore and truststore that store the certificates. If this attribute is not specified, the SSL configuration referenced by the `sslDefault` element is used.

If the z/OS Connect EE server's keystore, for example, `zosconnectKeyStore`, contains more than one private key, you must specify the `clientKeyAlias` attribute on the `ssl` element. This must be the alias of the certificate in the keystore that contains the key to be used during the TLS handshake.

For more information about the elements shown in the example above, see “[Configuration elements](#)” on page 753 in the Reference section.

By default the z/OS Connect EE server verifies the common name (CN) of the RESTful API personal certificate. This check can be disabled by setting the `disableCNCheck` attribute of the `webTarget` element. See [JAX-RS Client Properties \(`webTarget`\)](#) in the *WebSphere Application Server for z/OS Liberty* documentation for further information.

Call an API secured with an API key

An API key is a code passed by an application that is calling an API. It is used to establish the identity of the calling application.

The API key can act as both a unique identifier and a secret for authentication, and will typically have a set of access rights on the API associated with it.

To call a RESTful API protected by an API key, the CICS, IMS, or z/OS application, must include the API key as an authentication or authorization credential in the request.

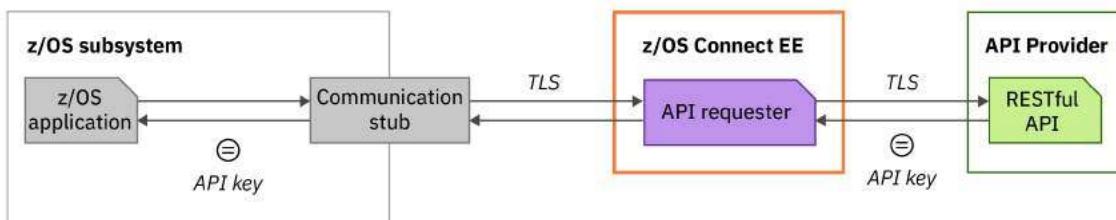


Figure 87. Calling an API secured with an API key.

Figure 87 on page 419 shows the API key credentials provided by the z/OS application, being sent by the communication stub to the z/OS Connect EE server and then propagated to the RESTful API either in a query string or as a request header.

The API key credentials can be a Client ID, or a Client ID and Client secret. [Table 34 on page 420](#) shows the different API key options and example parameter names.

Table 34. Client ID and Client secret parameter name examples

Location of credentials	Type of credentials	Parameter name
Header	Client ID	X-IBM-Client-Id
Header	Client secret	X-IBM-Client-Secret
Query	Client ID	client_id
Query	Client secret	client_secret

API key definitions can be provided using either a Swagger file or z/OS Connect EE build toolkit properties. For information on enabling the CICS, IMS, or z/OS application, to send API key credentials, see “[Securing your z/OS application calls to APIs](#)” on page 637.

How to configure an API key

To call a REST API protected by an API key, your CICS, IMS or z/OS application must include the API key as an authentication or authorization credential in the request. The API key credentials are propagated by the z/OS Connect EE server to the RESTful API in either a query string or as a request header.

This task is applicable when z/OS Connect EE is used as an API requester.

To enable API key authentication, you must have API key definitions to generate API key parameters in request data structures by using the build toolkit. You must also specify values for these API key parameters in your CICS, IMS or z/OS application.

API key definitions

API key definitions can be provided using either a Swagger file or z/OS Connect EE build toolkit properties.

Swagger files

You can find API key definitions in the security definitions section of a Swagger file. In the following example, a security schema `apiKeyHeader` specifies an API key X-IBM-Client-ID. When calling a REST API, the request header X-IBM-Client-ID: <keyValue> must be provided for authentication or authorization, where <keyValue> is the value of the API key X-IBM-Client-ID.

```
"securityDefinitions": {
  "apiKeyHeader": {
    "type": "apiKey",
    "name": "X-IBM-Client-ID",
    "in": "header"
  }
}
```

API key definitions in a Swagger file can be used to control access for all operations of the API or to a subset of the operations of the API.

Build toolkit properties

If the API key definition for an API is not specified in the Swagger file, you must specify the API key definition with the following build toolkit properties:

- **apiKeyParmNameInHeader:** The name of an API key that is sent as a request header, for example, header-IBM-Client-Secret. The value of this property can be a comma separated list to include client ID and client secret.
- **apiKeyParmNameInQuery:** The name of an API key that is sent in a query string, for example, query-IBM-Client-Secret. The value of this property can be a comma separated list to include client ID and client secret.

- **apiKeyMaxLength:** The maximum length of the values set for API key.

API key definitions specified using build toolkit properties can be used to control access for all operations of the API, but cannot be used to control access to a subset of the operations of the API.

For more information about build toolkit properties for API keys, see “[The build toolkit properties file](#)” on page 621.

Important:

- If both a Swagger file and z/OS Connect EE build toolkit properties are used to define API key names, only the API key names that are defined using the build toolkit properties are used to generate the names of the API key parameters that are in the request data structures.
- The Swagger file does not support specifying the maximum length of the key value of an API key. If you want to specify the maximum length of the key value of an API key, set the build toolkit property **apiKeyMaxLength**. The value of **apiKeyMaxLength** is applied to all the API keys for an API.

How to enable API key authentication

To enable the z/OS application to send the API key credentials with the API request, you must complete the following steps:

1. Run the build toolkit to generate API key parameters in the request data structure.

For example, if you set **apiKeyParmNameInHeader** to X-IBM-Client-Secret and **apiKeyMaxLength** to 255 in the build toolkit properties file, the following lines will be generated in the request data structure:

```
06 ReqHeaders.
09 X-IBM-Client-Secret-Length      PIC S9999 COMP-5 SYNC.
09 X-IBM-Client-Secret             PIC X(255).
```

X-IBM-Client-Secret-Length is a parameter that is automatically generated by the build toolkit based on the API key parameter **X-IBM-Client-Secret**. **X-IBM-Client-Secret-Length** specifies the length of the value of the **X-IBM-Client-Secret** parameter.

2. Specify the API key values, X-IBM-Client-Secret and X-IBM-Client-Secret-Length in your z/OS application program.

Call an API secured with OAuth 2.0

OAuth 2.0 is an authorization framework that enables a third-party application to request an HTTP service with limited access permission, either on behalf of a resource owner, or on behalf of the third-party client itself.

z/OS Connect EE API requester OAuth 2.0 support can be used when a CICS, IMS, or z/OS application, calls a RESTful API that requires an OAuth 2.0 access token.

There are four roles in an OAuth 2.0 flow:

Resource owner

An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end user.

In a z/OS Connect EE API requester scenario, the resource owner might be the user of the CICS, IMS, or z/OS application, or it might be another entity.

Resource server

The server that hosts the protected resources, and accepts and responds to protected resource requests by using access tokens.

In a z/OS Connect EE API requester scenario, the resource server is the request endpoint for the remote RESTful API.

Client

An application that makes protected resource requests on behalf of the resource owner and with its authorization.

In a z/OS Connect EE API requester scenario, the client is a combination of the CICS, IMS, or z/OS application, and the z/OS Connect EE server that calls the RESTful API on behalf of the CICS, IMS, or z/OS application.

Authorization server

The server that issues access tokens to the client after authenticating the resource owner and obtaining authorization.

In a z/OS Connect EE API requester scenario, the authorization server is called by the z/OS Connect EE server to retrieve an access token.

With OAuth 2.0, access tokens are used to access protected resources. An access token is normally a string that represents an authorization that is issued to the client. The string is usually opaque to the client. However, an access token can also be in the form of a JSON Web Token (JWT).

Access tokens represent specific scopes and durations of access, granted by the resource owner, and enforced by the resource server and authorization server.

In a z/OS Connect EE API requester scenario, the scope can be set by the CICS, IMS, or z/OS application, on behalf of the resource owner. It can give authorization to a particular subset of a resource or to particular actions (for example, read access but not write access).

Figure 88 on page 422 shows how an API secured with OAuth 2.0 can be called in a z/OS Connect EE API requester scenario.

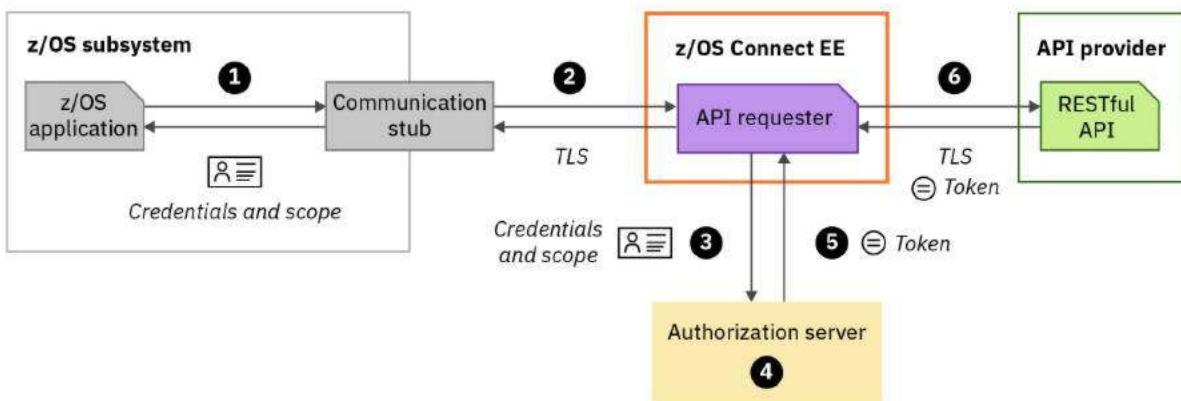


Figure 88. An illustration of how OAuth 2.0 works with z/OS Connect EE API requester.

The steps in Figure 88 on page 422 show how an API secured with OAuth 2.0 can be called by using z/OS Connect EE API requester:

1. The CICS, IMS, or z/OS application, sends credentials and access scope in the request. The type of credentials that can be sent depends on the OAuth 2.0 grant type that is being used (see “[OAuth 2.0 grant types supported by z/OS Connect EE](#)” on page 423).
2. The communication stub sends the request to the z/OS Connect EE server.
3. The z/OS Connect EE server extracts the OAuth-related information from the API request, inserts it into an HTTPS request for an access token, and sends this request to the authorization server.

4. The authorization server uses the client credentials to authenticate the CICS, IMS, or z/OS application, and verifies the scope that is requested by the CICS, IMS, or z/OS application.
5. If validation is successful, the authorization server returns an access token to the z/OS Connect EE server. The access token can be an opaque token or a JWT.
6. The access token is sent in the request to the RESTful API. When the request endpoint of the RESTful API accepts the access token, the z/OS Connect EE server is only allowed to access the resource owner's resources with the limited permission (access scope) that is specified in the access token.

For more information about OAuth 2.0, see [The OAuth 2.0 Authorization Framework](#)

For information about developing your CICS, IMS, or z/OS application to use OAuth 2.0, see “[Developing a z/OS application to call APIs](#)” on page 632 and “[OAuth parameters and variables](#)” on page 639.

OAuth 2.0 grant types supported by z/OS Connect EE

Four grant types are defined in the OAuth 2.0 specification, but z/OS Connect EE supports only the **Client Credentials** and **Resource Owner Password Credentials** grant types.

Client Credentials grant type

This grant type allows the CICS, IMS, or z/OS application, to request a resource on behalf of the resource owner without the resource owner's credentials.

When this grant type is used, the z/OS Connect EE server sends the client credentials and the access scope to the authorization server.

The client credentials can be set by the CICS, IMS, or z/OS application, in the BAQ-OAUTH-CLIENTID and BAQ-OAUTH-CLIENT-SECRET parameters of the BAQ-REQUEST-INFO data structure.

Alternatively, the client credentials can be specified in the `server.xml` of the z/OS Connect EE server by using a `zosconnect_authData` element, referenced by a `zosconnect_authorizationServer` element with the `basicAuthRef` attribute.

For the Client Credentials grant type, both client ID and client secret are required because the client is always confidential in the authorization server.

The scope can be set by using an OAuth scope variable BAQ-OAUTH-SCOPE.

Resource Owner Password Credentials grant type

When this grant type is used, the z/OS Connect EE server sends the resource owner's credentials, the client credentials, and the access scope to the authorization server.

The resource owner's credentials are set by the CICS, IMS, or z/OS application, in the BAQ-OAUTH-USERNAME and BAQ-OAUTH-PASSWORD parameters of the BAQ-REQUEST-INFO data structure.

For the Resource Owner Password Credentials grant type, **client ID** is required and **client secret** is optional. Specifying **client secret** depends on whether **client** is set to public in the authorization server. If **client** is set to public in the authorization server, only the client ID needs to be set. If **client** is set as confidential in the authorization server, both the **client ID** and **client secret** need to be set.

The scope can be set by using an OAuth scope variable BAQ-OAUTH-SCOPE.

This grant type is suitable for z/OS applications capable of obtaining the resource owner's credentials (username and password), for example, by using an interactive form. Enable this grant type only when the Client Credentials grant type is not a viable option.

Table 35 on page 424 summarizes where client credentials and resource owner password credentials can be set, depending on the OAuth 2.0 grant type used:

Table 35. Where client credentials and resource owner password credentials can be set

OAuth 2.0 specification entity	Resource owner password credentials	Client credentials	Where set
Client ID	Required	Required	In <code>server.xml</code> or in CICS, IMS, or z/OS application. See Note 1 .
Client Secret	Optional	Required	In <code>server.xml</code> or in CICS, IMS, or z/OS application. See Note 1 .
Username	Required	N/A	In CICS, IMS, or z/OS application.
Password	Required	N/A	In CICS, IMS, or z/OS application.

Note:

1. These values are set in the `user` and `password` attributes of a `zosconnect_authToken` element, which is referenced by the `basicAuthRef` attribute on the `zosconnect_authorizationServer` element.
2. If the client credentials are set in both the application and the `server.xml` of the z/OS Connect EE server, the credentials set in `server.xml` are used in the request to the authorization server.

OAuth access token caching

The z/OS Connect EE server stores new access tokens in the system cache and reuses the cached access tokens for API requests with matching OAuth-related information while the access tokens are within the expiration period. When expired, cached tokens are cleared from the system cache.

Figure 89 on page 424 shows how z/OS Connect EE caches OAuth 2.0 access tokens:

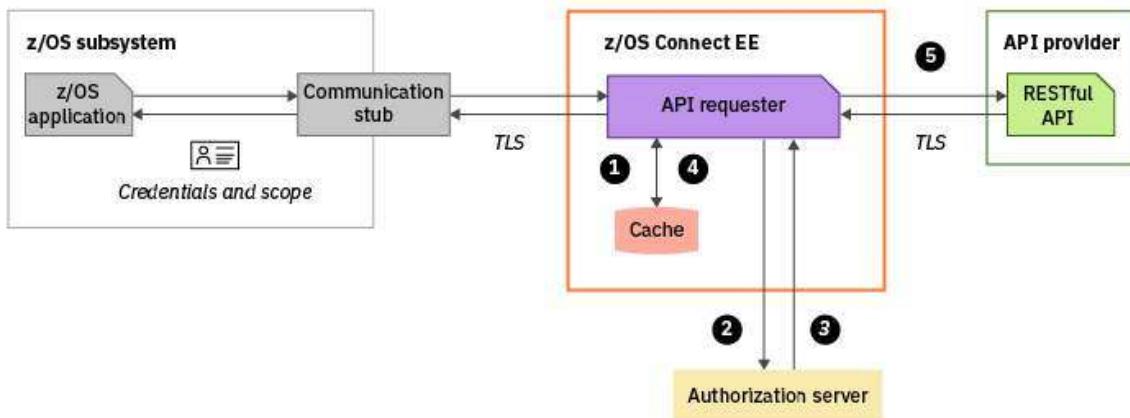


Figure 89. An illustration of the token caching function

The steps in Figure 89 on page 424 show how token caching works with z/OS Connect EE.

1. When the z/OS Connect EE receives an API request from the communication stub, it first extracts the OAuth-related information and checks whether there is a matching token (same request endpoint, client ID, and scope) in the system cache.

- If there is a match, the z/OS Connect EE server uses the cached access token. Skip to step “5” on page 425.
 - If there is no match, continue to step “2” on page 425.
2. The z/OS Connect EE server requests an access token from the authorization server.
 3. The authorization server returns an access token.
 4. The z/OS Connect EE server stores the access token in the system cache.
 5. The z/OS Connect EE server sends the access token with the API request.

Each access token has an expiration time that is set by the authorization server. When the cached token expires, it is cleared from the system cache.

Although an access token might be within the expiration period when loaded from the system cache, the token might be expired when it is received by the request endpoint of the RESTful API. In this situation, a 401 HTTP response code is returned to the z/OS Connect EE server, which clears the token from the cache. The z/OS Connect EE server then requests a new access token, stores it in the system cache, and sends the API request with the renewed access token to the request endpoint.

For information about developing your CICS, IMS, or z/OS application to use OAuth 2.0, see [“Developing a z/OS application to call APIs” on page 632](#) and [“OAuth parameters and variables” on page 639](#).

For information about configuring the z/OS Connect EE server for OAuth 2.0, see [“How to configure OAuth 2.0” on page 425](#).

How to configure OAuth 2.0

When a z/OS application calls a REST API that is protected by OAuth 2.0 on a request endpoint, the z/OS Connect EE server must be configured to identify the OAuth authentication data. The data is used for the z/OS Connect EE server to obtain an access token from an authorization server and to connect to the request endpoint.

The following example shows how to configure OAuth 2.0 for an endpoint connection.

```
<zosconnect_endpointConnection id="conn"
    host="https://api.server.com"
    port="8000"
    authenticationConfigRef="myoAuthConfig" />
<zosconnect_oAuthConfig id="myoAuthConfig"
    grantType="password|client_credentials"
    authServerRef="myAuthServer" />
<zosconnect_authorizationServer id="myAuthServer"
    tokenEndpoint="https://authorization.server.com:8001"
    basicAuthref="clientInfo"
    sslCertsRef="defaultSSLConfig" />
<zosconnect_authData id="clientInfo"
    user="clientID1"
    password="clientSecret1" />
```

As the example shows, the APIs to be called are protected on the server (<https://api.server.com>). When the z/OS Connect EE server tries to establish a connection to this server, it first identifies the information of the grant type and the authorization server by referring to the `zosconnect_oAuthConfig` element.

Specifying the grant type

Only the Resource Owner Password Credentials and Client Credentials grant types are supported in z/OS Connect EE. You can enable one of them by setting the value of the `grantType` attribute to `password` or `client_credentials`.

If the value of the `grantType` attribute is set to `password`, the z/OS Connect EE server requests an access token from the authorization server with the resource owner's credentials, which are used for the authorization server to verify the resource owner's account.

If the value of the `grantType` attribute is set to `client_credentials`, the z/OS Connect EE server requests an access token on behalf of itself.

Specifying the authorization server

You must specify the URL of the authorization server by setting the `tokenEndpoint` attribute in the `zosconnect_authorizationServer` element. As the TLS protocol is used in OAuth 2.0 to keep the access tokens confidential in transit, the URL must start with `https` and an TLS certificate must be provided. You can specify the TLS certificate information by configuring the `sslCertsRef` attribute in the `zosconnect_authorizationServer` element.

You can associate the `basicAuthref` attribute in the `zosconnect_authorizationServer` element with the `zosconnect_authData` element to indicate the information of client ID and client secret, which is used for the authorization server to authenticate the client.

For different grant types, the requirements for setting client ID and client secret are different:

- For the Resource Owner Password Credentials grant type, client ID is required and client secret is optional. Whether to specify client secret depends on whether client is set as public in the authorization server. If client is set as public in the authorization server, you need only to set client ID; if client is set as confidential in the authorization server, you must set both client ID and client secret.
- For the Client Credentials grant type, both client ID and client secret are required because the client is always confidential in the authorization server.

Important:

- The `sslCertsRef` attribute of the `zosconnect_authorizationServer` element is optional. If this attribute is not specified, the SSL configuration referenced by the `sslDefault` element is used.
- If you don't specify the `basicAuthRef` attribute of the `zosconnect_authorizationServer` element in the `server.xml` file, you must set client ID or client secret in your z/OS application program.

If client ID or client secret is specified in both the `server.xml` file and the z/OS application program, only the value set in the `server.xml` file is used.

Calling an API secured with a JSON Web Token (JWT)

JSON Web Token (JWT) is an open standard that defines a compact and URL-safe way to securely transmit information as a JSON object between parties. A JWT is often used to secure RESTful APIs because it can be used to authenticate a client that wants to access the APIs.

z/OS Connect EE provides three ways of calling an API secured with a JWT:

- Use the OAuth 2.0 support when the request is part of an OAuth 2.0 flow.

If a JWT is returned as an access token from an authorization server, the access token is sent in an API request by z/OS Connect EE in the HTTP authorization header. See “[Call an API secured with OAuth 2.0](#)” on page 421.

- When you need to send a JWT that is obtained from an authentication server in a custom flow, use the JWT support described in “[Calling an API secured with a third-party JWT](#)” on page 427.

For example:

- When you need to specify the HTTP verb that is used in the request to the authentication server.
- When you need to specify how the JWT is returned from the authentication server (for example, in an HTTP header or in a custom field in a JSON response message).
- When you need to use a custom header name for sending the JWT to the request endpoint.
- Use the locally generated JWT support when you need to send a JWT that is generated by the z/OS Connect EE server. The generated JWT contains the z/OS application asserted user ID in the “`sub`” (Subject) claim. The z/OS application asserted user ID is passed to the z/OS Connect EE server by the communications stub. For more information, see “[Calling an API secured with a locally generated JWT](#)” on page 431.

Calling an API secured with a third-party JWT

You can configure z/OS Connect EE to retrieve a JSON Web Token (JWT) from a third-party authorization server and pass it on a request to an API endpoint.

[Figure 1](#) shows how a JWT is obtained and used to call an API that is secured with a JWT.

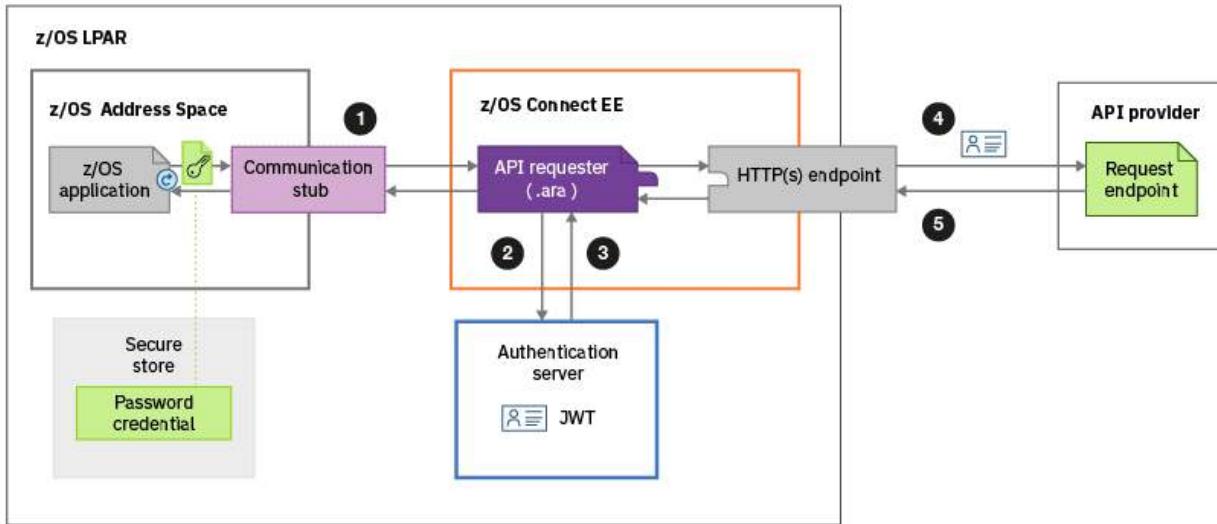


Figure 90. Illustration of how a JWT is retrieved and included in an API request

1. The communication stub transfers an API request that contains user credentials from the z/OS application to the z/OS Connect EE server. The user credentials include the user name and password that are required to authenticate with the authentication server to obtain a JWT. You can set the user credentials in the z/OS application or the `server.xml` file.
2. The z/OS Connect EE server builds an HTTPS request for a JWT based on the configuration in `server.xml` and sends the request to the authentication server. Basic authentication is the only supported method for authenticating the z/OS Connect EE connection to the authentication server. The user credentials are passed to the authentication server in the header or in the request body on the HTTPS request.
3. The authentication server authenticates the user and returns a JWT to the z/OS Connect EE server. You can configure z/OS Connect EE to expect the JWT to be returned in a header or response body, depending on the authentication server you are using to provide a JWT.
4. The z/OS Connect EE server sends an HTTP (or HTTPS) request for the API to the request endpoint, passing the JWT in the authorization header. By default, the HTTP Authorization header is used.
5. The request endpoint returns an HTTP (or HTTPS) response.

JSON Web Token (JWT) caching

When a JWT is initially requested from the authentication server, the z/OS Connect EE server stores the JWT in the system cache. With the JWT caching function, subsequent API requests reuse the cached JWTs if the JWT-related information is matched and the JWTs are within the expiration period. When expired, JWTs are cleared from the system cache.

[Figure 91 on page 428](#) shows how z/OS Connect EE caches JWTs:

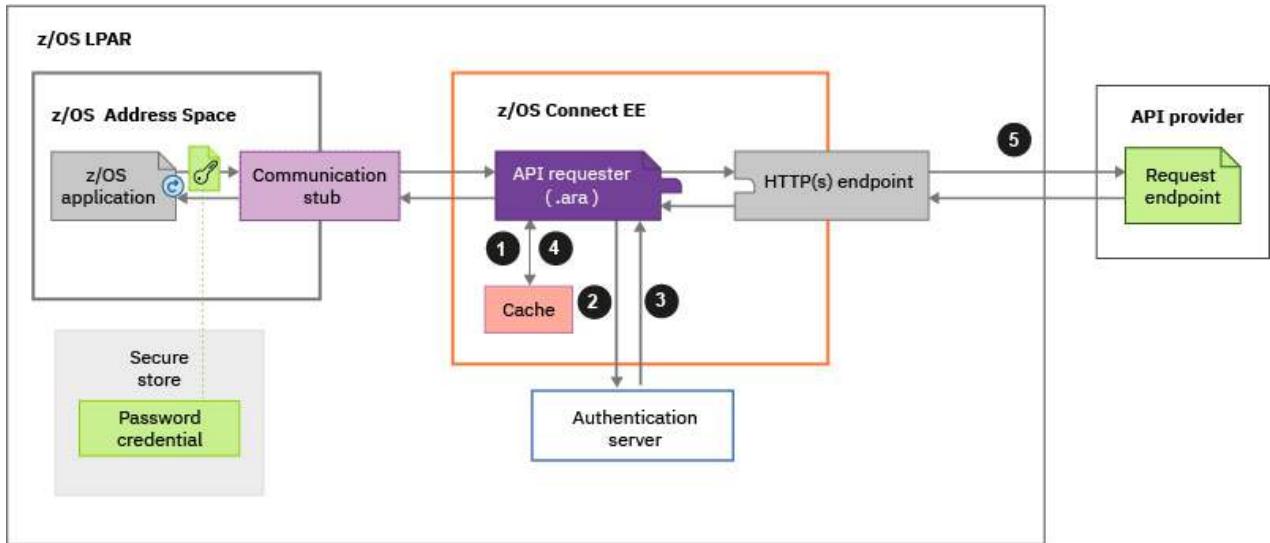


Figure 91. An illustration of the JWT caching function

The steps in Figure 91 on page 428 show how JWT caching works with z/OS Connect EE.

1. When the z/OS Connect EE server receives an API request from the communication stub, it first extracts the JWT-related information and checks whether there is a matching JWT (same authentication server, request method, credential location, user credentials) in the system cache.
 - If there is a match, the z/OS Connect EE server uses the cached JWT. Skip to step 5.
 - If there is no match, continue to step 2.
2. The z/OS Connect EE server requests a JWT from the authentication server.
3. The authentication server returns a JWT.
4. The z/OS Connect EE server stores the JWT in the system cache.
5. The z/OS Connect EE server sends the JWT with the API request.

The expiration time of a JWT is set on the `exp` claim by the authentication server. When the current date/time is not before the expiration date/time that is listed in the `exp` claim, the JWT is expired and cleared from the system cache.

Note:

- The z/OS Connect EE server does not cache a JWT that has no `exp` claim.
- The z/OS Connect EE server does not cache a JWT that has the `jti` claim.

For more information, see [JWT Claims](#).

Although a JWT might be within the expiration period when loaded from the system cache, the JWT might have expired when it is received by the request endpoint of the RESTful API. In this situation, a 401 HTTP response code is returned to the z/OS Connect EE server which will clear the JWT from the cache. Then the z/OS Connect EE server requests a new JWT, stores it in the system cache and sends the API request with the renewed JWT to the request endpoint. If a 401 HTTP response code is returned again, the z/OS Connect EE server clears the renewed JWT from the cache and an error message occurs.

How to enable a z/OS application to call an API secured with a JWT

To enable the z/OS application to call an API that is secured with a JWT through z/OS Connect EE, perform the following actions:

- Configure the z/OS Connect EE server as a client to retrieve a JWT from the authentication server and access the API with the JWT in the server.xml file. For more information about configuring the z/OS Connect EE server, see [“How to configure a third-party JWT” on page 429](#).

- Provide user credentials that are used to retrieve a JWT by including the user credentials in your z/OS application or specifying them in the `server.xml` file.

For how to modify your z/OS application to include user credentials in the API request, see “[JSON Web Token \(JWT\)](#)” on page 638 in “[Securing your z/OS application calls to APIs](#)” on page 637.

For how to specify user credentials in the `server.xml` file, see “[How to provide user credentials to the authentication server in server.xml](#)” on page 431 in “[How to configure a third-party JWT](#)” on page 429.

Important:

- The method of including user credentials in your z/OS application is available for both cases where the user credentials are sent in a header or in the body of the request to the authentication server. If the user credentials are sent in an HTTP header, z/OS Connect EE just passes the user credentials from the z/OS application in the header that is specified on the `header` attribute of the `tokenRequest` element in `server.xml`. If the user credentials are sent in a request body, you must ensure the JSON string specified for the `requestBody` attribute on the `tokenRequest` element uses the \$ syntax. Then z/OS Connect EE will replace the user credentials into the JSON string and pass them in the request body. For more information about setting the value of the `requestBody` attribute in this case, see “[Example B](#)” on page 761 in the *Reference* section.
- If both the z/OS application and the `server.xml` file are set up with user credentials for a JWT, the user credentials that are specified in the `server.xml` file are used.

How to configure a third-party JWT

Some REST APIs use a JWT for authentication. To enable a z/OS application to call such APIs, you must configure the z/OS Connect EE server to retrieve JWTs from an authentication server and then access the APIs with the JWTs.

Follow these instructions to configure an API requester to use the JWT support. To configure an API requester with the OAuth 2.0 support when the request is part of an OAuth 2.0 flow, see “[Call an API secured with OAuth 2.0](#)” on page 421. If you do not know which support you need to use, see “[Calling an API secured with a JSON Web Token \(JWT\)](#)” on page 426.

The following example shows how to configure the JWT authentication information for an endpoint connection.

```
<zosconnect_endpointConnection id="conn"
    host="https://api.server.com"
    port="8000"
    authenticationConfigRef="myJWTConfig" />
<zosconnect_authToken id="myJWTConfig" authServerRef="myJWTserver"
    header="myJWT-header-name" >
    <!--Configure the tokenRequest element by referring to Example 1 and Example 2 -->
    <tokenRequest />
    <!--Configure the tokenResponse element by referring to Example 3 and Example 4 -->
    <tokenResponse />
</zosconnect_authToken>
<zosconnect_authorizationServer id="myJWTserver"
    tokenEndpoint="https://jwt.server.com:9443/JWTTokenGenerator/getJwtToken"
    basicAuthRef="tokenCredential"
    sslCertsRef="defaultSSLConfig" />
<zosconnect_authData id="tokenCredential"
    user="jwtuser"
    password="jwtpassword"/>
```

As the example shows, the API to be called is secured on a request endpoint (`https://api.server.com:8000`) by using JWT. To enable the z/OS Connect EE server to request a JWT from the authentication server (`https://jwt.server.com:9443`), the `authenticationConfigRef` attribute on the `zosconnect_endpointConnection` element is associated with the `zosconnect_authToken` element.

How to use `zosconnect_authToken`

The `zosconnect_authToken` element defines the following JWT configuration information by using the following attributes and sub elements:

- Use the **header** attribute to specify the name of the header that contains the JWT on the API request to the request endpoint. If you don't specify the **header** attribute, the HTTP Authorization header is used by default.
- Use the **authServerRef** attribute to specify the authentication server. This attribute is associated with the **zosconnect_authorizationServer** element.
- Use the **tokenRequest** element to specify the format of the request to the authentication server for a JWT.
- Use the **tokenResponse** element to specify the format of the response from the authentication server.

The following examples show how to configure the **tokenRequest** and **tokenResponse** elements based on different requirements:

- [Example 1: User credentials are sent in a specified header](#)
- [Example 2: User credentials are sent in the request body](#)
- [Example 3: The JWT is returned in the specified header](#)
- [Example 4: The JWT is returned in the response body](#)

Example 1: User credentials are sent in a specified header

```
<tokenRequest
    credentialLocation="header"
    header="Authorization"
    requestMethod="GET" />
```

When user credentials are sent in a header on the request for a JWT, you must set the **credentialLocation** attribute to `header` and specify a header name for the **header** attribute. If you don't specify the **header** attribute, the HTTP Authorization header is used by default.

Example 2: User credentials are sent in the request body

```
<tokenRequest
    credentialLocation="body"
    requestMethod="POST"
    // Use XML escaped characters in requestBody
    requestBody="{&quot;apiuser&quot;:&quot;${userid}&quot;,&quot;apipassword&quot;:&quot;${password}&quot;}" />
```

When user credentials are sent in the body of the request for a JWT, you must set the **credentialLocation** attribute to `body`, and set the **requestBody** attribute to a JSON string with characters such as ", <, > and & replaced by using the XML escape characters. For more information, see: [XML escape characters in the Websphere Application Server Liberty documentation](#).

The **requestBody** attribute value has two formats: one directly contains the user credentials; the other one uses the \$ syntax as the example shows. For more information about how to set this attribute, see [requestBody](#).

Note: The **requestMethod** attribute must be set when user credentials are sent in a header or in the body of the request for a JWT. The acceptable values are GET, PUT or POST.

Example 3: The JWT is returned in the header

```
<tokenResponse
    tokenLocation="header"
    header="JWTAuthorization" />
```

When the JWT is returned in the header on the response from the authentication server, you must specify the **tokenLocation** attribute to `header` and specify a header name for the **header** attribute. If you don't specify the **header** attribute, the HTTP Authorization header is used by default.

Example 4: The JWT is returned in the response body

```
<tokenResponse  
    tokenLocation="body"  
    responseFormat="JSON"  
    tokenPath=".tokenname" />
```

When the JWT is returned in the response body from the authentication server, you must set the **tokenLocation** attribute to body and set the **responseFormat** attribute.

The **responseFormat** attribute can be set to Text or JSON. If the **responseFormat** attribute is set to Text, z/OS Connect EE considers the response body to be the JWT. If the **responseFormat** attribute is set to JSON, you must specify a JSONPath expression for the **tokenPath** attribute so that z/OS Connect EE can locate and get the JWT from the returned JSON string. For more information about how to set the **tokenPath** attribute, see [tokenPath](#).

For more information about the attributes and sub elements of the `zosconnect_authToken` element, see “[zosconnect_authToken](#)” on page 760 in “[Configuration elements](#)” on page 753.

How to provide user credentials to the authentication server in `server.xml`

The user credentials include the user name and password that are used for basic authentication between the z/OS Connect EE server and the authentication server. You can provide the user credentials in the `server.xml` file in the following two ways:

- Specify the user credentials in the **zosconnect_authData** element and associate the element with the **basicAuthRef** attribute on the **zosconnect_authorizationServer** element. This method can be used to send the user credentials to the authentication server as an HTTP header or in the request body.
Note: If the user credentials are sent in the request body, you must also set the **requestBody** attribute to a JSON string that contains the \$ character as shown in [Example 2](#). Thus, the user credentials specified on the **zosconnect_authData** element can be replaced into the JSON string and passed in the request body to the authentication server.
- Include user credentials explicitly in the JSON string that is specified for the **requestBody** attribute on the `tokenRequest` element. This method can only be used when the user credentials are sent in the request body.

Important:

When sending user credentials to the authentication server in the request body, if the user credentials are specified by using both the **requestBody** attribute and the **zosconnect_authData** element, the user credentials set on the **requestBody** attribute are used.

You can also modify your z/OS application to include user credentials. For more information, see “[JSON Web Token \(JWT\)](#)” on page 638 in “[Securing your z/OS application calls to APIs](#)” on page 637.

Calling an API secured with a locally generated JWT

You can call an API by sending a JSON Web Token (JWT) that is locally generated by the z/OS Connect EE server. The generated JWT contains a subject claim that is the z/OS application asserted user ID passed from the communication stub.

Note: If you want to call an API by sending a JWT that is obtained from an authentication server, see “[Calling an API secured with a third-party JWT](#)” on page 427.

Before you begin, ensure that you are familiar with the JWT and JWT claim specifications as defined in the [JSON Web Token \(JWT\) IETF documentation](#).

The ability to create a locally generated JWT requires SAF authorization to be configured in the z/OS Connect EE server. For more information about SAF authorization, see “[API requester authorization](#)” on page 446.

The z/OS Connect EE server can generate a JWT with the "sub" (Subject) claim set to the z/OS application asserted user ID, and send that JWT in the specified HTTP header on the request to the RESTful API.

The user ID asserted by the z/OS application depends on the application type:

- For CICS applications, it is the task owner ID.
- For IMS applications, it is the transaction owner ID.
- For other z/OS applications, it is the job owner ID.

The z/OS Connect EE server generates JWTs using the JSON Web Signature (JWS) serialization, as described in [JSON Web Signature \(JWS\) IETF documentation](#).

The z/OS Connect EE server stores the generated JWT in the system token cache, unless it contains a "jti" (JWT ID) claim. Expired JWTs are cleared from the system token cache. You can clear this token cache by using the **modify** command. For more information, see ["MODIFY command syntax" on page 794](#) in the *Reference* section. If the attribute values of the `jwtBuilder` or `zosconnect_authTokenLocal` server.xml configuration file elements are changed, then the token cache should be cleared using the modify command, so that a new JWT is generated using the new configured attributes on the next request.

[Figure 1](#) shows the data flow when a z/OS application calls an API with a locally generated JWT that contains the z/OS application asserted user ID as the "sub" claim.

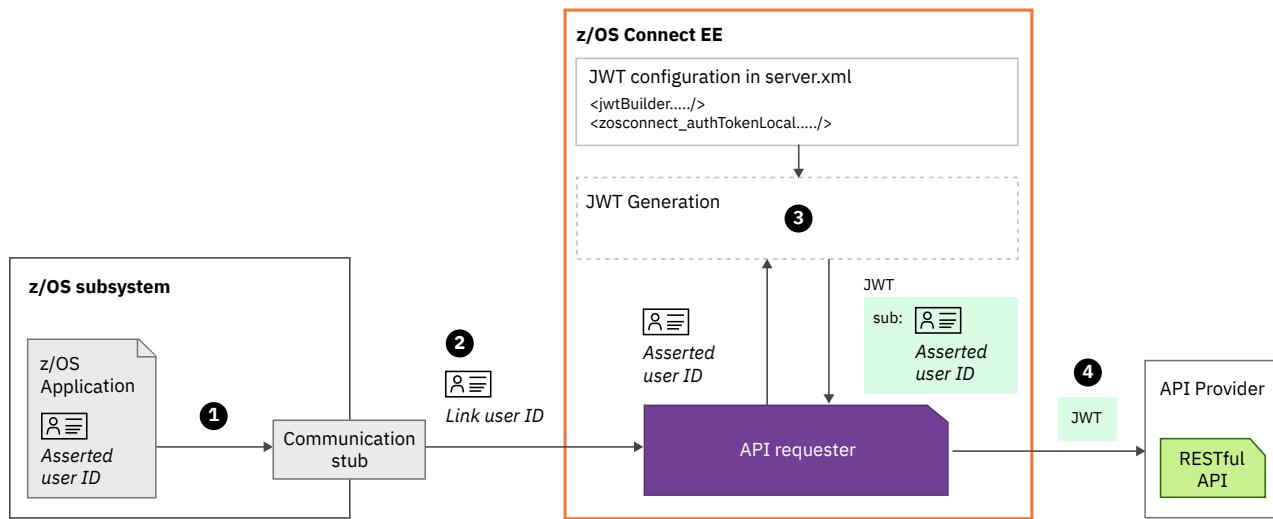


Figure 92. Illustration of how a local JWT is generated and included in an API request

1. The communication stub extracts the z/OS application asserted user ID from the z/OS application environment.
2. Typically, the z/OS subsystem that calls the z/OS Connect EE server is authenticated using basic authentication or client authentication. The authenticated link user ID requires authorization to connect to the z/OS Connect EE server. Either the authenticated link user ID, or the z/OS application asserted user ID, requires authorization to the API requester, depending on whether identity assertion is configured.
3. The z/OS Connect EE server performs a SAF SURROGAT profile check to ensure that the authenticated link user ID associated with the API requester request is allowed to generate a JWT on behalf of the z/OS application asserted user ID. If there is no authenticated link user ID, then the check is performed against the SAF unauthenticated user ID. For more information, see “[SAF unauthenticated user ID](#)” on page 458. If the surrogate check is successful, then the z/OS Connect EE server generates a JWT based on the parameters that are configured in `server.xml`, and sets the z/OS application asserted user ID as the “sub” (Subject) claim of the JWT. The parameters include the registered claims, for example, the “aud” (Audience) claim, that are specified on the `jwtBuilder` element, and the public and private claims that are specified on the `zosconnect_authTokenLocal` element. For more information about how to configure the two elements, see “[How to configure a locally generated JWT](#)” on page 434.
4. The z/OS Connect EE server sends the request to the RESTful API, passing the JWT in an HTTP header. By default, the HTTP Authorization header is used. The HTTP header includes the “Bearer” scheme keyword followed by the JWT.

Note: It is recommended that the SAF unauthenticated user ID is authorized to use only the minimum number of SAF resources.

The locally generated JWT support is independent of API requester identity assertion. However, they can be configured to work together if you additionally want to invoke an API requester with the z/OS application asserted user ID before generating the JWT. In both scenarios, the same z/OS application asserted user ID and same link user ID are used, but the names of the SURROGAT profiles are different (BAQTOKEN and BAQASSRT). For more information about API requester identity assertion, see “[Identity assertion for API requesters](#)” on page 441.

How to configure a locally generated JWT

To call an API with a locally generated JWT, you must configure the `jwtBuilder` and `zosconnect_authTokenLocal` elements in the `server.xml` configuration file and define SAF SURROGAT profiles.

Before you begin

- You must be familiar with the information in “[Calling an API secured with a locally generated JWT](#)” on page 431.
- Set up the connection from the z/OS subsystem to the z/OS Connect EE server.
 - For CICS, see “[Configure CICS to access z/OS Connect EE to call APIs](#)” on page 306.
 - For IMS, see “[Configuring IMS to access z/OS Connect EE for API calls](#)” on page 309.
 - For other z/OS applications, see “[Configuring other z/OS applications to access z/OS Connect EE for API calls](#)” on page 312.
- Complete the task “[How to activate and configure the SAF user registry](#)” on page 456 to configure the z/OS Connect EE server to use z/OS authorized services and a System Authorization Facility (SAF) user registry.
- For the surrogate check on the z/OS application asserted user ID, you must ensure that the following requirements are completed:
 - The user ID that is used to run the z/OS Connect EE server instance must have READ access to the BPX.SERVER FACILITY class profile.
 - The Java library `libifaedjreg64.so` must be program-controlled. For example, enter the following command:

```
extattr +p /usr/lib/java_runtime/libifaedjreg64.so
```

- Third-party native code or native libraries, such as those used by a z/OS Connect EE interceptor must be program-controlled. To define the native code or native library to be program-controlled, enter the following command under z/OS UNIX System Services:

```
extattr +p <your library file path>
```

Note: When your administrator applies maintenance to the Java library, this setting might be reset, so you must reenter this command.

About this task

You configure the z/OS Connect EE server to generate a JWT with the "sub" (Subject) claim set to the z/OS application asserted user ID, and send that JWT in the specified HTTP header on the request to the RESTful API.

The z/OS application asserted user ID needs to be defined in the SAF security manager on the LPAR where z/OS Connect EE is running.

The authenticated link user ID must be a SAF user ID. Therefore, if authenticating with a TLS client certificate, the certificate must be mapped to a SAF user ID. If using a distributed identity to authenticate with basic authentication, then that distributed identity must be mapped to a SAF user ID. The link user ID requires surrogate access to generate a JWT on behalf of the z/OS application asserted user ID. This is achieved by using SAF SURROGAT class profiles. If there is no authenticated link user ID associated with the API requester request, then the check is performed against the SAF unauthenticated user ID. See “[SAF unauthenticated user ID](#)” on page 458.

The authenticated link user ID requires authorization to connect to the z/OS Connect EE server. Either the authenticated link user ID, or the z/OS application asserted user ID, requires authorization to the API requester, depending on whether identity assertion is configured. For more information about authorization, see “[API requester authorization](#)” on page 446.

This task assumes that RACF is used as security manager. If you are using an alternative SAF security manager, refer to the appropriate product documentation for the equivalent commands.

To configure a locally generated JWT, you must update the `server.xml` configuration file to:

- Configure the `jwtBuilder` and `zosconnect_authTokenLocal` elements to define the parameters that are used to generate a JWT locally. The parameters include the claims that are registered in the IANA "JSON Web Token Claims" registry (for example, the "aud" (Audience) claim), public and private claims, and additional attributes of a JWT. For more information about the IANA "JSON Web Token Claims" registry, see [JSON Web Token Claims Registry](#).
- Reference the `zosconnect_authTokenLocal` element from the `zosconnect_endpointConnection` element.

Procedure

1. Typically, the z/OS subsystem that calls the z/OS Connect EE server is authenticated using basic authentication or client authentication.
 - For more information on enabling basic authentication, see “[API requester basic authentication to z/OS Connect EE](#)” on page 407.
 - For more information on enabling client authentication, see “[API requester client certificate authentication to z/OS Connect EE](#)” on page 412.

Note: Locally generated JWTs can be used without an authenticated link user ID, in this case the SAF SURROGAT profile checks are performed against the SAF unauthenticated user ID.

2. Define the appropriate SURROGAT profile definition in RACF. Consult your security administrator to confirm the appropriate definitions to use for your environment.

- a. Define a profile named `assertedid.BAQTOKEN` in the SURROGAT class, where `assertedid` represents the z/OS application asserted user ID. For example, if the asserted identity is FRED, enter the RACF command:

```
RDEFINE SURROGAT FRED.BAQTOKEN UACC(NONE)
```

- b. Grant the authenticated link user ID READ access to the SURROGAT profile created in the previous step. For example, if the link user ID is LINKID1 and is allowed to generate JWTs containing a "sub" claim value of FRED, enter the RACF command:

```
PERMIT FRED.BAQTOKEN CLASS(SURROGAT) ID(LINKID1) ACCESS(READ)
```

- c. Repeat these commands for other z/OS application asserted user IDs and authenticated link user IDs that are used for locally generated JWTs.

- d. After you have entered all the RACF SURROGAT class profile commands, refresh the SURROGAT class. Enter the RACF command:

```
SETROPTS RACLIST(SURROGAT) REFRESH
```

Alternatively: If the authenticated link user ID is allowed to generate JWTs containing the "sub" claim value for any z/OS application asserted user IDs, you can choose to use a generic profile definition, for example:

```
RDEFINE SURROGAT *.BAQTOKEN UACC(NONE)
```

```
PERMIT *.BAQTOKEN CLASS(SURROGAT) ID(LINKID1) ACCESS(READ)
```

Note: It is recommended that the SAF unauthenticated user ID is authorized to use only the minimum number of SAF resources. An authenticated link user ID is typically used. However, if you choose not to have a link user ID and allow API requester requests to run under the SAF unauthenticated user ID, then you will need to permit that user ID READ access to the appropriate SURROGAT profile definitions. The default SAF unauthenticated user ID is WSGUEST. As the SAF unauthenticated user ID is defined with the RESTRICTED attribute, it must be explicitly permitted access to a non-generic profile name. For more information, see ["SAF unauthenticated user ID" on page 458](#) and [Defining restricted user IDs](#).

3. Configure the `jwtBuilder` element to specify the registered claims like the `aud` claim and additional attributes that are used to generate a JWT. For example, `scope` and `signatureAlgorithm`. The example in [Figure 93 on page 436](#) shows how to configure a `jwtBuilder` element called `myLocalJWTBuilder`.

```
<jwtBuilder
    id="myLocalJWTBuilder"
    scope="scope1"
    audiences="myApp1"
    signatureAlgorithm="RS256"
    expiresInSeconds="30m"
    jti="true"
    nbfOffset="50s"
    keyStoreRef="myKeyStore"
    keyAlias="keyLabel" />
```

Figure 93. Example jwtBuilder element configuration

By default the "iss" (Issuer) claim is set to `https://{{hostname}}:{{httpsPort}}/jwt/{{jwtBuilderId}}` where `hostname` and `httpsPort` are the hostname and HTTPS port of the z/OS Connect EE server and `jwtBuilderId` is the `id` attribute value of the `jwtBuilder` element. To specify an alternative value for the "iss" (Issuer) claim, configure the `issuer` attribute of the `jwtBuilder` element. For more information about how to configure the `jwtBuilder` element, see [JWT Builder \(jwtBuilder\)](#) in the *WebSphere Application Server for z/OS Liberty documentation*.

Note:

- The `claims` attribute on the `jwtBuilder` allows only existing user attributes that are defined for the subject of this JWT in the user registry to be included as claims in the JWT. To specify public and private claim name value pairs, use the `claims` subelement on the `zosconnect_authTokenLocal` element as shown in Step 4.
 - The value for the registered claim "iat" (Issued At) is automatically set when the JWT is generated locally by the z/OS Connect EE server.
4. Configure the `zosconnect_authTokenLocal` element to specify the parameters that are used to generate the JWT that is used to authenticate with the RESTful API endpoint. The parameters include a reference to the `jwtBuilder` element, public and private claims, and the HTTP header name that is used to contain the JWT on the request to the RESTful API endpoint.
- The example in Figure 94 on page 437 shows how to configure a `zosconnect_authTokenLocal` element called `myLocalJWTConfig`.

```
<zosconnect_authTokenLocal id="myLocalJWTConfig"
    tokenGeneratorRef="myLocalJWTBuilder" 1
    header="myJWTHeader" 2
    <claims>{"employees" : 3
        [{"ID": "1234567890", "Name": "Tom" },
        {"ID": "1234567891", "Name": "Jerry" },
        {"ID": "1234567892", "Name": "Hash" }],
        "BRANCH": "HR_PROG1",
        "ACTION": "QUERY_STATUS"}</claims>
</zosconnect_authTokenLocal>
```

1 The `tokenGeneratorRef` attribute references the `jwtBuilder` element with the ID value `myLocalJWTBuilder` that is defined in Step 3.

2 The `header` attribute specifies the name of the HTTP header that contains the JWT on the API request to the RESTful API endpoint. If you do not specify the `header` attribute, the HTTP Authorization header is used by default.

3 The `claims` subelement specifies the public and private claims as a JSON string.

Figure 94. Example `zosconnect_authTokenLocal` element configuration

For more information about the `zosconnect_authTokenLocal` element, see [zosconnect_authTokenLocal](#) in the configuration elements section.

Note:

- The `claims` subelement is intended to specify only public and private claims. If registered claims, such as "aud" (Audience), are specified on the `claims` subelement, then these values overwrite the corresponding values that are configured on the `jwtBuilder` element that is referenced by the `tokenGeneratorRef` attribute of the `zosconnect_authTokenLocal` element. If the "sub" claim is specified on the `claims` subelement, its value is overwritten by the z/OS Connect EE server to be the z/OS application asserted user ID. Registered claims are defined in the IANA [JSON Web Token Claims Registry](#).
- If the JSON string value of the `claims` subelement contains XML markup characters, such as <, > and &, then include the JSON string inside a CDATA section so that those characters are treated as literals. For example, if one of the private claims above was "BRANCH": "<HR_PROG1>" then the `claims` subelement value must be specified as:

```
<claims><! [CDATA[{"employees" :
        [{"ID": "1234567890", "Name": "Tom" },
        {"ID": "1234567891", "Name": "Jerry" },
        {"ID": "1234567892", "Name": "Hash" }],
        "BRANCH": "<HR_PROG1>",
        "ACTION": "QUERY_STATUS"}]]></claims>
```

For more information about the CDATA section, see [CDATA](#).

5. Configure the `zosconnect_endpointConnection` element to identify the locally generated JWT authentication data by associating the `authenticationConfigRef` attribute to the `zosconnect_authTokenLocal` element.

For example,

```
<zosconnect_endpointConnection id="conn"
    host="https://api.server.com"
    port="9443"
    authenticationConfigRef="myLocalJWTConfig" />
```

Figure 95. Example `zosconnect_endpointConnection` element configuration

In Figure 95 on page 438, the `authenticationConfigRef` attribute references the `zosconnect_authTokenLocal` element with the ID value `myLocalJWTConfig` that was defined in Step 4.

Results

The z/OS Connect EE server performs a SAF SURROGAT profile check to ensure that the authenticated link user ID (or the SAF unauthenticated user ID) associated with the API requester request is allowed to generate a JWT on behalf of the z/OS application asserted user ID. If the surrogate check is successful, then the z/OS Connect EE server generates a JWT based on the configuration on the `jwtBuilder` and `zosconnect_authTokenLocal` elements. The generated JWT payload contains the specified claim names and values, and the "sub" (Subject) claim of the JWT is set to the z/OS application asserted user ID. The JWT is sent on the API request to the RESTful API in the specified HTTP header (by default the Authorization header). The HTTP header includes the "Bearer" scheme keyword followed by the JWT.

Calling an API secured with multiple authentication and authorization methods

z/OS Connect EE API requester provides the capability that allows a CICS, IMS or z/OS application to call a RESTful API that is secured with multiple authentication or authorization methods.

The supported authentication and authorization methods are client certificate authentication, basic authentication, JWT and OAuth 2.0.

- To configure z/OS Connect EE to support client certificate authentication to the RESTful API endpoint, you must specify the `sslCertsRef` on the `zosconnect_endpointConnection` in the `server.xml` file. For more information, see [“API requester TLS client authentication to a RESTful API endpoint” on page 418](#).
- To configure z/OS Connect EE to support basic authentication to a RESTful API endpoint, calling an API secured with a JWT or calling an API secured with OAuth 2.0, you must specify the `authenticationConfigRef` attribute on the `zosconnect_endpointConnection` element in the `server.xml` file. Follow [Table 1](#) to find out how to use the `authenticationConfigRef` attribute to specify the combination use of basic authentication, JWT and OAuth 2.0.

Table 36. Supported combinations of basic authentication, JWT and OAuth 2.0

Authentication / authorization methods	Elements to be referenced by the <code>authenticationConfigRef</code> attribute
Multiple JWTs (either generated by an external authentication server or locally generated or both)	<ul style="list-style-type: none">• <code>zosconnect_authToken</code> (for JWT generated by an external authentication server)• <code>zosconnect_authTokenLocal</code> (for JWT generated locally)

Table 36. Supported combinations of basic authentication, JWT and OAuth 2.0 (continued)

Authentication / authorization methods	Elements to be referenced by the authenticationConfigRef attribute
OAuth 2.0 and one or more JWTs (either generated by an external authentication server or locally generated or both)	<ul style="list-style-type: none"> • zosconnect_oAuthConfig (for OAuth 2.0) • zosconnect_authToken (for JWT generated by an external authentication server) • zosconnect_authTokenLocal (for JWT generated locally)
Basic authentication and one or more JWTs (either generated by an external authentication server or locally generated or both)	<ul style="list-style-type: none"> • zosconnect_authData (for basic authentication) • zosconnect_authToken (for JWT generated by an external authentication server) • zosconnect_authTokenLocal (for JWT generated locally)

As z/OS Connect EE needs to use the same HTTP Authorization header to pass either user credentials for basic authentication or the OAuth access token to the request endpoint by default, so the combination of basic authentication and OAuth 2.0 cannot be supported. And when one or more JWTs are used with basic authentication or OAuth 2.0, you must specify the header attribute in the zosconnect_authToken and zosconnect_authTokenLocal element(s) with a name other than Authorization.

When more than one JWT is used at the same time, you must consider the following two things:

- Ensure different header names are specified for the zosconnect_authToken and zosconnect_authTokenLocal elements.
- For the JWTs generated by an external authentication server, where user credentials are not provided in the `server.xml` file, z/OS Connect EE uses the user credentials that are specified in your z/OS application.

Related concepts

[“API requester basic authentication to a RESTful API endpoint” on page 417](#)

[“Call an API secured with OAuth 2.0” on page 421](#)

Related information

[“Calling an API secured with a JSON Web Token \(JWT\)” on page 426](#)

[“Calling an API secured with OAuth 2.0 and a JWT” on page 439](#)

Calling an API secured with OAuth 2.0 and a JWT

z/OS Connect EE API requester provides the capability that allows a CICS, IMS or z/OS application to call a RESTful API that is secured by using both JWT and OAuth 2.0, where JWT is used to authenticate a client, and OAuth 2.0 is used to authorize the client to access the API.

In a z/OS Connect EE API requester scenario, z/OS Connect EE acts as the client to request both a JWT (from an external authentication server) and an access token and access the RESTful API with the two kinds of tokens on behalf of the CICS, IMS or z/OS application.

Figure 1 shows how a JWT and an access token are obtained and used to access an API when a user wants to call an API from the z/OS application.

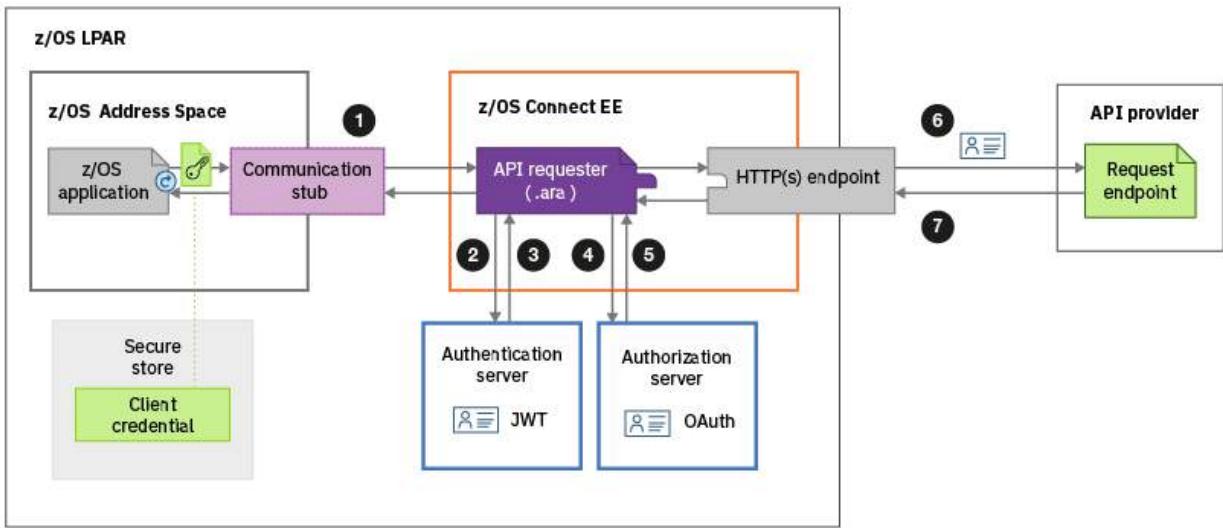


Figure 96. Illustration of how JWT and OAuth 2.0 are applied to API requester

Note: You can also use the same server to provide multiple token types.

1. The communication stub transfers an API request that contains credentials from the z/OS application to the z/OS Connect EE server.
 - For JWT authentication, the credentials include a user name and password.
 - For OAuth authorization, the type of information that is included in the credentials depends on the grant type that is used. For example, if the Resource Owner Password Credential grant type is used, the credentials must contain a user name and password and a client ID and client secret. For more information about the OAuth grant types that are supported by the API requester, see [“Call an API secured with OAuth 2.0” on page 421](#).
2. The z/OS Connect EE server sends an HTTPS request for a JWT to the authentication server. The HTTPS request contains the user credentials.
3. The authentication server authenticates the user and returns a JWT to the z/OS Connect EE server.
4. The z/OS Connect EE server sends an HTTPS request that contains the credentials to the authorization server.
5. The authorization server validates the credentials, verifies the grant scope, and returns an access token to the z/OS Connect EE server.
6. The z/OS Connect EE server sends an HTTP (or HTTPS) request for the API to the request endpoint with the JWT and the access token. The OAuth access token is passed in the HTTP Authorization header. The JWT is passed in a bespoke header that is specified in `server.xml`.
7. The request endpoint returns an HTTP (or HTTPS) response.

Steps for implementing JWT and OAuth 2.0 with API requester

To call an API that is secured with both OAuth 2.0 and a JWT from your z/OS application, take the following steps:

1. Implement JWT for the API request. The following excerpt from the `server.xml` file shows an example configuration for requesting a JWT.

```
<zosconnect_authToken id="myJWTConfig"
  authServerRef="myJWTserver"
  header="JWT-header-name" >
```

```

<tokenRequest credentialLocation="header"
    header="JWTReqAuthorization"
    requestMethod="GET" />
<tokenResponse tokenLocation="header"
    header="JWTResAuthorization" />
</zosconnect_authToken>

```

The **header** attribute on the `zosconnect_authToken` element must be specified with a name other than `Authorization` because the HTTP `Authorization` header is used to pass the OAuth access token by default.

For more information about how to implement JWT, see [“How to enable a z/OS application to call an API secured with a JWT” on page 428](#).

2. Implement OAuth for the API request. The following excerpt from the `server.xml` file shows an example configuration for requesting an OAuth access token.

```

<zosconnect_oAuthConfig id="myoAuthConfig"
    grantType="password"
    authServerRef="myAuthServer" />

```

For more information about implement OAuth 2.0, see [“Call an API secured with OAuth 2.0” on page 421](#).

3. Reference both the JWT and OAuth configuration on the endpoint connection, for example:

```

<zosconnect_endpointConnection id="conn"
    host="https://api.server.com"
    port="8000"
    authenticationConfigRef="myoAuthConfig,myJWTConfig" />

```

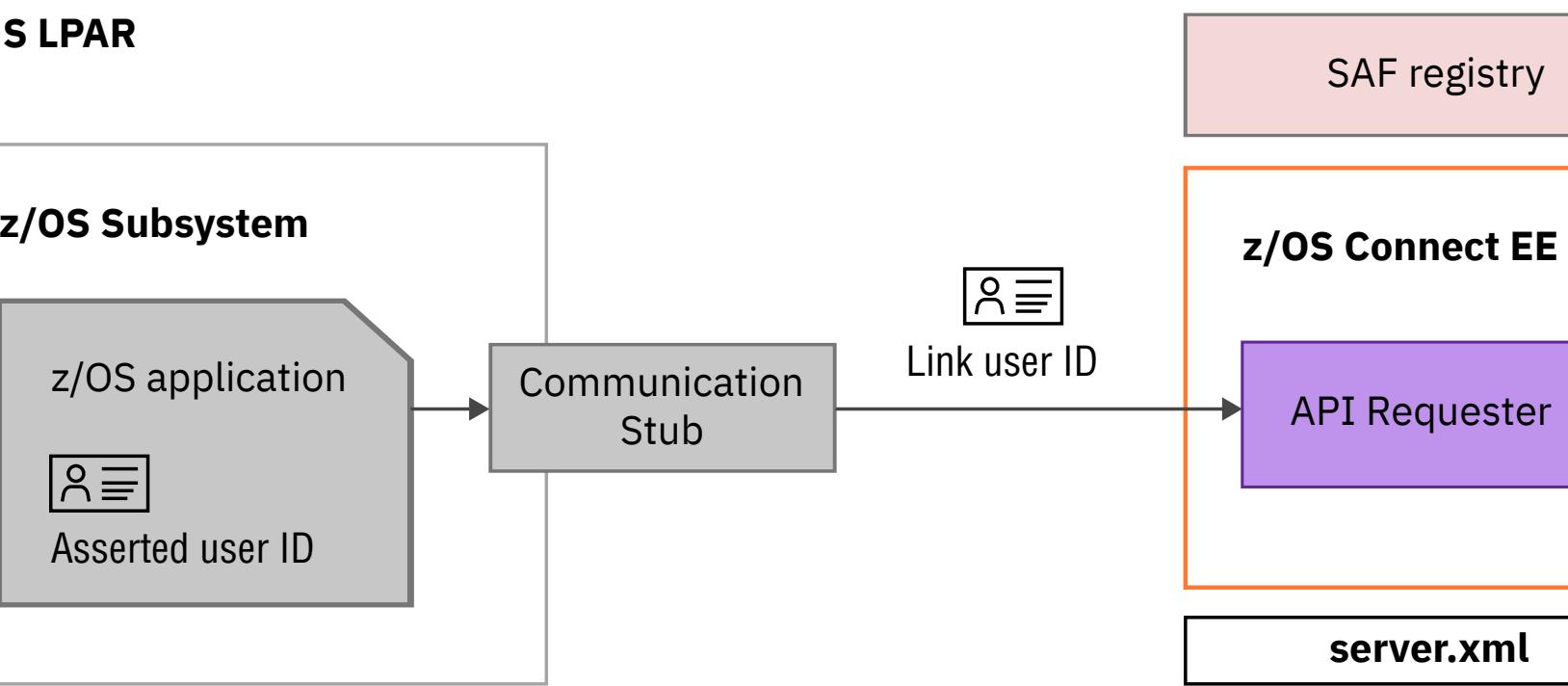
Important: The `authenticationConfigRef` attribute can reference one or more elements. Apart from referencing both a `zosconnect_authToken` and a `zosconnect_oAuthConfig` element to call an API that is secured with a JWT and OAuth 2.0, z/OS Connect EE supports other combinations of multiple authentication or authorization methods. For more information, see [“Calling an API secured with multiple authentication and authorization methods” on page 438](#).

Identity assertion for API requesters

z/OS Connect EE provides the capability of allowing a z/OS application to invoke an API requester with an asserted identity that is provided in the application context.

Before you study this topic, you should be familiar with the information in [“Overview of z/OS Connect EE security” on page 331](#), [“API requester authentication and identification” on page 407](#), and [“API requester authorization” on page 446](#).

Figure 97 on page 442 illustrates how identity assertion works in z/OS Connect EE.



Two user IDs are included in [Figure 97 on page 442](#):

The link user ID

The identity that is used to authenticate the z/OS subsystem access to the z/OS Connect EE server. It is configured using basic authentication or client authentication, for more information about configuring this user ID, see “[API requester basic authentication to z/OS Connect EE](#)” on page 407 or “[API requester client certificate authentication to z/OS Connect EE](#)” on page 412.

The z/OS application asserted user ID

An identity that is provided in the z/OS application context.

- For CICS applications, the z/OS application asserted user ID is the task owner ID.
- For IMS applications, the z/OS application asserted user ID is the transaction owner ID.
- For other z/OS applications, the z/OS application asserted user ID is the job owner ID.

As illustrated in [Figure 97 on page 442](#), a secure connection is set up from the z/OS subsystem to z/OS Connect EE by using the link user ID for authentication. Instead of using the link user ID, the z/OS application attempts to invoke an API requester by using the z/OS application asserted user ID as an asserted identity. When the z/OS application sends a request to the z/OS Connect EE server, the z/OS application asserted user ID is flowed to the z/OS Connect EE server in a proprietary header. Depending on which values you configure for the **requireAuth** and **idAssertion** attributes in the `server.xml` file, the z/OS Connect EE server performs different actions on the user IDs.

Important: Authorization interceptor is required for the identity assertion function to work.

The following table shows the actions that are performed by the z/OS Connect EE server based on the value of the **requireAuth** and **idAssertion** attributes.

Note: The following actions assume that the authorization interceptor is enabled so that z/OS Connect EE can perform the authorization check.

*Table 37. z/OS Connect EE actions on different value combinations of **requireAuth** and **idAssertion***

Value of requireAuth	Value of idAssertion	Actions performed by z/OS Connect EE
true	OFF	Identity assertion is disabled. z/OS Connect EE authenticates the link user ID, and checks whether it has the authority to invoke an API requester.
	ASSERT_SURROGATE	Identity assertion is enabled. The z/OS Connect EE server authenticates the link user ID, and performs a SAF SURROGAT profile check, to ensure it is a surrogate of the z/OS application asserted user ID. If the link user ID is a surrogate of the z/OS application asserted user ID, the server further checks whether the z/OS application asserted user ID has the authority to invoke an API requester; otherwise, a BAQR7114E message occurs.
	ASSERT_ONLY	Identity assertion is enabled. The z/OS Connect EE server authenticates the link user ID, and directly checks whether the z/OS application asserted user ID has the authority to invoke an API requester.

Table 37. z/OS Connect EE actions on different value combinations of **requireAuth** and **idAssertion** (continued)

Value of requireAuth	Value of idAssertion	Actions performed by z/OS Connect EE
false	OFF	Identity assertion is disabled. A BAQR0407W message is issued to indicate that no authenticated user ID is associated with the request.
	ASSERT_SURROGATE	Identity assertion is enabled. The z/OS Connect EE server checks whether the z/OS application asserted user ID has the authority to invoke an API requester, and a warning message occurs to indicate that the ASSERT_ONLY value is used instead of the ASSERT_SURROGATE value.
	ASSERT_ONLY	Identity assertion is enabled. The z/OS Connect EE server checks whether the z/OS application asserted user ID has the authority to invoke an API requester.

Note:

1. The link user ID and z/OS application asserted user ID might be granted with different authorities to perform actions on API requesters. When authentication is required for the access from the z/OS subsystem to the z/OS Connect EE server and identity assertion is enabled, the link user ID is only used for API requester authentication and the z/OS application asserted user ID is used for the API requester authorization check.
2. When you enable identity assertion without authentication on the z/OS subsystem access to the z/OS Connect EE server or surrogate check, you must ensure that the z/OS application asserted user ID is trusted and allowed to access the z/OS Connect EE server, which means that the z/OS application asserted user ID must be authorized to the `zosConnectAccess` role. For more information about configuring the `zosConnectAccess` role, see [“How to configure the zosConnectAccess role with a SAF user registry” on page 449](#).
3. SAF credentials can be cached to improve performance. The SAF cache contains SAF user IDs and any associated RACF groups in which the user ID resides. The SAF cache is only applicable to API requester, and only when ID assertion is enabled. For more information, see [“zosconnect_authorizationInterceptor” on page 758](#). You can clear this cache by using the **Modify** command. For more information, see [“MODIFY command syntax” on page 794](#).

When you enable identity assertion, it is applied to every API requester because the **idAssertion** attribute is set on the `zosconnect_apiRequesters` element.

For information about configuring the z/OS Connect EE server for identity assertion, see:

How to configure identity assertion

To enable identity assertion for API requesters, you must configure the `idAssertion` attribute in the `server.xml` file. You might also need to define SAF SURROGAT profiles.

Before you begin

- Study the information in [“Identity assertion for API requesters” on page 441](#).
- Set up the connection from the z/OS subsystem to the z/OS Connect EE server.
 - For CICS, see [“Configure CICS to access z/OS Connect EE to call APIs” on page 306](#).
 - For IMS, see [“Configuring IMS to access z/OS Connect EE for API calls” on page 309](#).
 - For other z/OS applications, see [“Configuring other z/OS applications to access z/OS Connect EE for API calls” on page 312](#).

- Complete the task “How to activate and configure the SAF user registry” on page 456 to configure the z/OS Connect EE server to use z/OS authorized services and a System Authorization Facility (SAF) user registry.
- Configure security for the z/OS subsystem connection to the z/OS Connect EE server according to your needs.
 - For basic authentication, see “API requester basic authentication to z/OS Connect EE” on page 407.
 - For client authentication, see “API requester client certificate authentication to z/OS Connect EE” on page 412.
- If you want to perform a surrogate check on the asserted identity, you must ensure that the following requirements are completed:
 - The user ID that is used to run the z/OS Connect EE server instance must have READ access to the BPX.SERVER FACILITY class profile.
 - The Java library `libifaedjreg64.so` must be program-controlled. For example, enter the following command:

`extattr +p /usr/lib/java_runtime/libifaedjreg64.so`

- Third-party native code or native libraries, such as those used by a z/OS Connect EE interceptor must be program-controlled. To define the native code or native library to be program-controlled, enter the following command under z/OS UNIX System Services:

`extattr +p <your library file path>`

Note: When your administrator applies maintenance to the Java library, this setting might be reset, so you must re-enter this command.

About this task

You configure the z/OS Connect EE server to specify that the API requester should be invoked by the z/OS application asserted user ID. Optionally, you additionally configure that a SAF SURROGAT profile check is required, to ensure that the link user ID which has authenticated with the z/OS Connect EE server is authorized to act as a surrogate of the z/OS application asserted user ID.

This task assumes that RACF is used as security manager. If you are using an alternative External Security Manager, refer to the appropriate product documentation for the equivalent commands.

The asserted identity needs to be defined in a RACF system that can be accessed by the LPAR where z/OS Connect EE is running.

Procedure

The following steps demonstrate how to enable a z/OS application to invoke an API requester with an asserted identity that is provided in the z/OS application context.

1. Configure the **idAssertion** attribute in the `server.xml` file.

- If you want to enable the z/OS application to invoke an API requester with an asserted identity and don't need the z/OS Connect EE server to perform a surrogate check, set the **idAssertion** attribute to `ASSERT_ONLY`.

```
<zosconnect_apiRequesters location="/myserver/resources/zosconnect/apiRequesters"
updateTrigger="polled" idAssertion="ASSERT_ONLY">
```

- If you want to enable the z/OS application to invoke an API requester with an asserted identity and also want the z/OS Connect EE server to perform a surrogate check, ensure the **requireAuth** attribute for the request is set to `true` and set the **idAssertion** attribute to `ASSERT_SURROGATE`.

```
<zosconnect_apiRequesters location="/myserver/resources/zosconnect/apiRequesters"
updateTrigger="polled" idAssertion="ASSERT_SURROGATE">
```

For more information about how to set the **requireAuth** attribute for a request, see [Step 1 in “How to configure basic authentication with a SAF user registry” on page 411](#)

Note: When the z/OS Connect EE server is configured not to require authentication for a request by setting the **requireAuth** attribute to `false`, if you set the **idAssertion** attribute to `ASSERT_SURROGATE`, a warning message occurs and the z/OS Connect EE server uses the value `ASSERT_ONLY` instead.

2. Configure authorization for the asserted identity to perform the *Invoke* action on the API requester.

- Ensure that the authorization interceptor is configured to allow the z/OS Connect EE server to perform authorization checks.
- Ensure that the asserted identity is added into a SAF group that is assigned to the *Admin* or *Invoke* authorization level so that the asserted identity can pass the authorization check for the API requester.

For more information about how to configure authorization for the asserted identity, see [“How to configure authorization levels with a SAF user registry” on page 450](#).

3. Optional: If you have the **requireAuth** attribute for the request set to `true`, then z/OS Connect EE can also check whether the link user ID used for authenticating the z/OS subsystem access to the z/OS Connect EE server is a surrogate of the z/OS application asserted user ID. To authorize the link user ID as a surrogate of the z/OS application asserted user ID, perform the following steps:

- a) Define a profile named `assertedid.BAQASSRT` in the SURROGAT class, where `assertedid` represents the z/OS application asserted user ID that is provided in the z/OS application context.

For example, if the asserted identity is FRED, issue RACF command: **RDEFINE SURROGAT FRED.BAQASSRT UACC(NONE) OWNER(FRED)**.

- b) Grant the identity that is used for authentication READ access to the `assertedid.BAQASSRT` profile in the SURROGAT class.

For example, if the link user ID is LINKID1 and is allowed to act as a surrogate of z/OS application asserted user ID FRED, issue RACF command: **PERMIT FRED.BAQASSRT CLASS(SURROGAT) ID(LINKID1) ACCESS(READ)**

API requester authorization

Learn how z/OS Connect EE authorizes both access to z/OS Connect EE and the actions that can be performed on API requesters.

Before you study this topic, you should be familiar with the information in [“Overview of z/OS Connect EE security” on page 331](#) and [“API requester authentication and identification” on page 407](#).

z/OS Connect EE authorization normally occurs after authentication completes successfully. The following checks are performed:

- Authorization to access z/OS Connect EE. The server checks the `requireAuth` attribute value set in the `server.xml` configuration file. By default, this value is set to `true`. When `true`, an authorization check is performed to ensure that the authenticated user ID is assigned to the `zosConnectAccess` role. This behavior is described in [“Authorization to access z/OS Connect EE” on page 447](#).
- Authorization to perform specific actions on API requesters. z/OS Connect EE checks whether the authorization interceptor is enabled. If so, then it checks whether the authenticated user ID is a member of a user registry group, and whether that group is assigned to the appropriate authorization level. This behavior is described in [“Authorization to perform specific actions on API requesters” on page 447](#).

Figure 81 on page 374 shows the normal z/OS Connect EE authorization flow. In the diagram, z/OS application can be a CICS, IMS, or z/OS application.

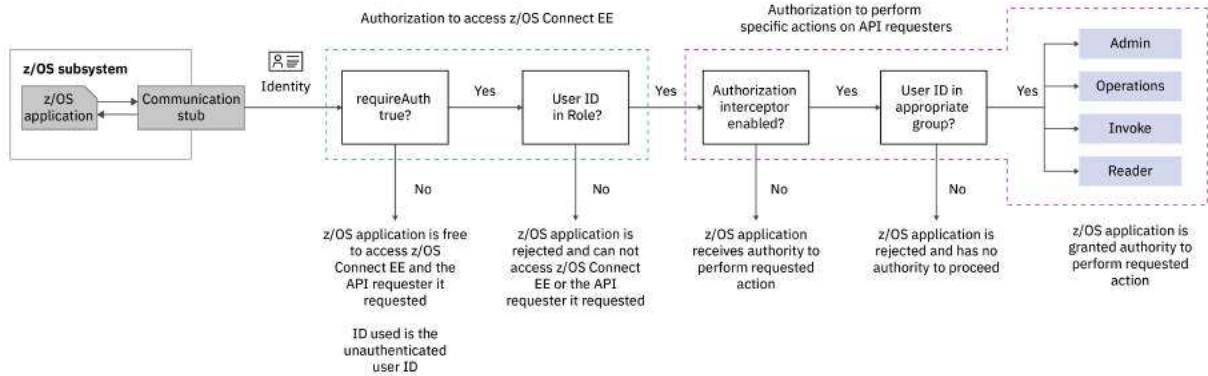


Figure 98. Authorization flow for API requester

Authorization to access z/OS Connect EE

The first authorization check that is performed by the z/OS Connect EE server is to control whether the authenticated user ID is authorized to access z/OS Connect EE. This check occurs when the `server.xml` configuration file attribute `requireAuth` is set to `true`. The attribute `requireAuth` can be set globally for all API requesters in the z/OS Connect EE server, or for individual API requesters. The `requireAuth` attribute is primarily used to control authentication, see “[API requester authentication and identification](#)” on page 407, but is also used for this authorization check.

This authorization is controlled by the `zosConnectAccess` role, and access is granted by assigning user IDs associated with requests to that role. The request user ID must be defined in the z/OS Connect EE server’s user registry. If a distributed ID is mapped to a user registry user ID, then the authorization check is performed against the mapped user ID.

There are two methods of assigning user IDs to the `zosConnectAccess` role:

- The SAF EJBROLE class profile `profilePrefix.zos.connect.access.roles.zosConnectAccess`. This can be defined with users and groups given READ access to it. For more information, see “[How to configure the zosConnectAccess role with a SAF user registry](#)” on page 449.
- The `authorization-roles` element defined in the `server.xml` configuration file. You can either assign individual users and groups to the role, or you can authorize all authenticated users to the role by using the `<special-subject type="ALL_AUTHENTICATED_USERS"/>`.

Authorization to perform specific actions on API requesters

You can use a more granular level of authorization to control which users can perform specific actions on API requesters. This authorization is implemented by the z/OS Connect EE *authorization interceptor*.

The authorization interceptor defines four authorization levels. A comma-separated list of user registry groups can be assigned to each level. Both the authorization interceptor and the user registry groups can be defined at various scopes. Access to a specific authorization level is granted by ensuring that the user ID associated with the request is a member of the user registry group that is assigned to that authorization level at the appropriate scope. If a distributed identity is mapped to a SAF user ID, then the authorization check is performed against the mapped SAF user ID and its group in the SAF registry. When a SAF registry is used by the z/OS Connect EE server, then OMVS segments must be defined for both the SAF user ID and group.

You can use existing user registry groups, or create new groups. If you create new groups, consider whether you want the group names to indicate the authorization level they control, and also the associated scope. For example, you can create separate groups to control API requesters that belong to different lines of business.

The authorization levels are as follows:

Admin

All z/OS Connect EE actions are allowed, including all *Operations*, *Invoke*, and *Reader* actions.

Operations

All z/OS Connect EE operations and actions are allowed except for *Invoke*.

Invoke

Ability to invoke APIs and services. *Invoke* authority does not provide access to z/OS Connect EE *Operations* actions.

Reader

Ability to obtain information about API requesters. *Reader* authority does not provide access to z/OS Connect EE *Operations* actions nor does it provide access to invoke API requesters.

See [Table 31 on page 375](#) for the authorization levels required to perform specific requests.

The z/OS Connect EE authorization interceptor can be configured at the following scopes:

- Globally for the z/OS Connect EE server. By default, this configuration enables the authorization interceptor for all API requesters. An individual API requester can opt out of running the global interceptors.
- For specific API requesters. If only required for a few API requesters, this configuration is an alternative to enabling the authorization interceptor globally.

The user registry groups assigned to each of the authorization levels can also be configured at the following scopes:

- Globally for the z/OS Connect EE server. The user registry groups that are assigned at this scope become the default groups for actions that are performed on all API requesters.
- For specific API requesters. If authorization is required for only a few API requesters, this configuration is an alternative to enabling the authorization interceptor globally.

The user registry groups assigned to each of the authorization levels can also be configured at the following scopes:

- Globally for the z/OS Connect EE server. The user registry groups that are assigned at this scope become the default groups for actions that are performed on all API requesters.
- For specific API requesters. The user registry groups that are assigned at this scope take precedence over the global groups.

If you require all or most API requester requests to be authorized, you can configure the authorization interceptor globally for the z/OS Connect EE server and assign the default user registry groups. If you need to restrict specific API requesters to different users, you can then assign specific user registry groups to those individual API requesters, to override the global groups. See [Table 31 on page 375](#) for details of the authorization levels required to perform different requests.

Table 38. Authorization levels required to perform requests

Request type	Global authorization level configuration	Specific API requester level configuration (overrides global level)
Invoke an API requester. (See note).	Admin or Invoke.	Admin or Invoke.
Get a list of API requesters.	Admin, Operations, or Reader.	Only global authorization applies.
Get details of an API requester.	Admin, Operations, or Reader.	Admin, Operations, or Reader.
Deploy an API requester.	Admin or Operations.	Only global authorization applies.
Update an API requester.	Admin or Operations.	Admin or Operations.
Delete an API requester.	Admin or Operations.	Admin or Operations.

Table 38. Authorization levels required to perform requests (continued)

Request type	Global authorization level configuration	Specific API requester level configuration (overrides global level)
Change status of an API requester (start or stop).	Admin or Operations.	Admin or Operations.
Get z/OS Connect EE admin Swagger doc.	Admin, Operations, or Reader.	Only global authorization applies.

Note: When a user is authorized to invoke an API requester, the user is allowed to invoke all operations of the RESTful API as described in its Swagger document.

For more information about the authorization levels that are required to perform specific RESTful administration actions for API requesters, see [How to use the RESTful administration interface to manage API requesters](#).

How to configure the `zosConnectAccess` role with a SAF user registry

Assign SAF users and groups to the `zosConnectAccess` role to control which users, specified on the call from the CICS, IMS or z/OS application, can access z/OS Connect EE.

This task is applicable when z/OS Connect EE is used as an API requester.

Before you begin

- You should be familiar with the information in [“API requester authorization” on page 446](#).
- You need to know the SAF user IDs and groups that are to be granted access to z/OS Connect EE.
- You must have completed the task [“How to configure basic authentication with a SAF user registry” on page 411](#), or have authenticated with TLS client authentication and mapped the CICS, IMS or a z/OS application certificate to a SAF user ID.
- You must have write access to the `server.xml` configuration file.

About this task

Now you have configured the z/OS Connect EE server to require authentication by setting the attribute `requireAuth="true"`, you assign SAF users and groups to the `zosConnectAccess` role.

This task assumes that RACF is used as security manager. If you are using an alternative External Security Manager, refer to the appropriate product documentation for the equivalent commands.

Procedure

1. Configure the server to use SAF for authorization.

This configures the z/OS Connect EE server to perform authorization checks against the SAF registry. The `zosConnectAccess` role check is performed against the authenticated SAF user ID. Add the following element to the `server.xml` configuration file:

```
<safAuthorization id="saf-authorization"/>
```

To display SAF authorization messages such as RACF ICH408I, when unauthorized users attempt to access z/OS Connect EE, specify the attribute `racRouteLog="ASIS"`.

For more information about the `safAuthorization` element, see the [Server configuration section](#) in the *IBM WebSphere Application Server for z/OS Liberty* documentation.

2. Assign users and groups to the `zosConnectAccess` role.

For SAF authorization, access is controlled using the SAF EJBROLE profile `profilePrefix.zos.connect.access.roles.zosConnectAccess`.

- a) Define the SAF EJBROLE profile.

Enter the following command:

```
RDEFINE EJBROLE profilePrefix.zos.connect.access.roles.zosConnectAccess UACC(NONE)
```

In this command, `profilePrefix` is the value used for this server as specified on the `profilePrefix` attribute of the `safCredentials` element in `server.xml`. The default profile prefix value is `BBGZDFLT`.

- b) Activate the EJBROLE class

Enter the following command

```
SETROPTS CLASSACT(EJBROLE)
```

- c) Assign the user IDs and groups who require authority to access.z/OS Connect EE READ access to this profile.

For example, to assign group "STAFF" and user "EMPLOY1" to the `zosConnectAccess` role, enter the following commands:

```
PERMIT profilePrefix.zos.connect.access.roles.zosConnectAccess CLASS(EJBROLE) ID(STAFF)  
ACCESS(READ)
```

```
PERMIT profilePrefix.zos.connect.access.roles.zosConnectAccess CLASS(EJBROLE)  
ID(EMPLOY1) ACCESS(READ)
```

- d) Refresh the EJBROLE class.

Enter the following command:

```
SETROPTS RACLIST(EJBROLE) REFRESH
```

- e) Ensure the user IDs and groups have also been granted READ access to the `profilePrefix APPL` profile.

This is described in the task [“How to activate and configure the SAF user registry” on page 456](#). Enter the following command:

```
PERMIT profilePrefix CLASS(APPL) ACCESS(READ) ID(EMPLOY1)
```

3. Start, or restart the server if it was already running, to pick up the changes made to the RACF class profiles.

Results

The SAF users and groups assigned to the `zosConnectAccess` role now have authorization to access z/OS Connect EE.

How to configure authorization levels with a SAF user registry

Configure authorization to control which users can perform specific actions on z/OS Connect EE API requesters using a SAF user registry.

This task is applicable when z/OS Connect EE is used as an API requester.

Before you begin

- You should be familiar with the information in [“API requester authorization” on page 446](#).
- You need to know which user IDs and groups are to be granted various authorization levels to which API requesters.
- Both the SAF user ID associated with any request, and the SAF groups assigned to the authorization levels must have OMVS segments. If you use RACF, specify the following OMVS segment parameters:
 - AUTOGID or GID(group-identifier) on the ADDGROUP or ALTGROUP command.

- AUTOUID or UID(user-identifier), HOME and PROGRAM on the ADDUSER or ALTUSER command. For example: OMVS(AUTOUID HOME(/u/user) PROGRAM(/bin/sh)).
- You must have write access to the `server.xml` configuration file.
- You must have configured the z/OS Connect EE server to authenticate the SAF user IDs, and ensure that each of those user IDs is associated with a group in the SAF registry. See “[How to configure basic authentication with a SAF user registry](#)” on page 411 or “[How to configure client certificate authentication with RACF](#)” on page 415.

About this task

Configure a z/OS Connect EE server to perform authorization checks by using the z/OS Connect EE authorization interceptor. You assign SAF groups to each of the authorization levels: *admin*, *operations*, *invoke* and *reader* globally or for a specific API requester.

You configure the authorization interceptor at the global scope with SAF groups that are assigned to each of the global authorization levels. The configuration examples demonstrate the following options:

- How specific API requesters can opt out of using the globally configured authorization interceptor. See examples for “Book_Inventory”.
- How the authorization interceptor can be configured only for specific API requesters instead of at the global scope. See examples for “Stock_Control”.
- How different SAF groups can be assigned to each of the authorization levels for specific API requesters. See examples for “Hospital_Patients”.

Note:

- If you want to configure authorization for the RESTful administration actions such as deploy an API requester, or get a list of API requesters, additional configuration is needed. For more information, see the “[API requester authorization](#)” on page 446.

Procedure

1. Configure the z/OS Connect EE authorization interceptor.

Add the following element to the `server.xml` configuration file:

```
<zosconnect_authorizationInterceptor id="zosConnectAuthorizationInterceptor" />
```

For more information about the `zosconnect_authorizationInterceptor` element see “[zosconnect_authorizationInterceptor](#)” on page 758 in the *Reference* section.

2. Configure which interceptors are to be called.

The z/OS Connect EE authorization interceptor is called by referencing it in the following element of the `server.xml` configuration file:

```
<zosconnect_zosConnectInterceptors id="interceptorList1"
    interceptorRef="zosConnectAuthorizationInterceptor" />
```

In this example, `zosConnectAuthorizationInterceptor` is the `id` attribute value of the referenced `zosconnect_authorizationInterceptor` element.

For more information about the `zosconnect_zosConnectInterceptors` element, see “[zosconnect_zosConnectInterceptors](#)” on page 784 in the *Reference* section.

3. Optional: Configure the authorization interceptor at the global scope.

Perform this step to assign the SAF groups that are authorized by default to perform actions on all API requesters. These global groups are also used to authorize access to RESTful administration actions to deploy an API requester, and list all API requesters.

To configure the authorization interceptor globally, add the `globalInterceptorsRef` attribute to the `zosconnect_zosConnectManager` element. For example:

```
<zosconnect_zosConnectManager  
    globalInterceptorsRef="interceptorList1"  
    ... />
```

In this example, `interceptorList1` is the `id` attribute value of the referenced `zosconnect_zosConnectInterceptors` element.

Perform step “4” on page 452 only if you have completed step “3” on page 451.

4. Optional: Configure the SAF groups and default groups to be assigned to each authorization level at the global scope.

The SAF groups are authorized by default to perform actions on all API requesters. These global groups are also used to authorize access to RESTful administration operations to deploy an API requester and list all API requesters.

To configure the authorization level groups globally, add the attributes to define the groups to the `zosconnect_zosConnectManager` element. For example:

```
<zosconnect_zosConnectManager  
    globalInterceptorsRef="interceptorList1"  
    globalAdminGroup="GMADMIN"  
    globalInvokeGroup="GMINVOKE"  
    globalOperationsGroup="GMOPER"  
    globalReaderGroup="GMREADER,GMGUEST"  
    ... />
```

The values of the group attributes must be one or more SAF group names. The values are case-insensitive. To specify multiple groups for any role, use a comma-separated list, as shown for the `globalReaderGroup`.

For more information about the `zosconnect_zosConnectManager` element, see “[zosconnect_zosConnectManager](#)” on page 784 in the *Reference* section.

5. Optional: Opt out of using the authorization interceptor's global scope for an individual API requester.

If the authorization interceptor is configured at the global scope, authorization checks are applied to all requests for API requesters because, by default, API requesters run all the global interceptors, in addition to any interceptors configured specifically at the API requester scope.

To opt out of running the authorization interceptor for API requester “Book_Inventory”, ensure that a `apiRequester` element for “Book_Inventory” is configured and add the `runGlobalInterceptors="false"` attribute to it. For example:

```
<zosconnect_apiRequesters>  
    <apiRequester name="Book_Inventory"  
        runGlobalInterceptors="false"  
        ... />  
</zosconnect_apiRequesters>
```

6. Optional: Configure the authorization interceptor for specific API requesters.

Perform this step only if you do not require the authorization interceptor to be configured globally and only need to control authorization for specific API requesters.

a) Remove `globalInterceptorsRef="interceptorList1"` from the `zosconnect_zosConnectManager` element, if specified.

b) Optional: Configure the authorization interceptor for the specific API requester.

For example, for the API requester “Stock_Control”, ensure that a `apiRequester` element for “Stock_Control” is configured and add the `interceptorsRef` attribute to it. For example:

```
<zosconnect_apiRequesters>  
    <apiRequester name="Stock_Control" interceptorsRef="interceptorList1"  
        ... />  
</zosconnect_apiRequesters>
```

In this example, `interceptorList1` is the `id` attribute value of the referenced `zosconnect_zosConnectInterceptors` element.

For more information about the `apiRequester` elements, see [apiRequester](#) in the *Reference* section.

7. Optional: Assign the SAF groups to authorization levels at the API requester scope.

These groups take precedence over the global groups except when controlling authorization to the RESTful administration actions to deploy an API requester or list all API requesters.

- a) Define either `globalInterceptorsRef="interceptorList1"` on the `zosconnect_zosConnectManager` element, or `interceptorsRef="interceptorList1"` on the `apiRequester` element.
- b) Optional: To configure the authorization level groups for the API requester “Hospital_Patients”, ensure that a `apiRequester` element for “Hospital_Patients” is configured and add the attributes to define the groups to it.

For example:

```
<zosconnect_apiRequesters>
    <apiRequester name="Hospital_Patients"
        adminGroup="A3ADMIN"
        invokeGroup="A3STAFF"
        operationsGroup="A3OPS"
        readerGroup="A3NURSES"
        ...
    />
</zosconnect_apiRequesters>
```

In this example, the values of the group attributes must be the SAF group names. The values are case-insensitive.

Note: If a group is to be assigned to the same authorization level at all scopes, then it can be specified at the global scope and inherited by all API requesters. For example, to assign the same group to the reader authorization level at all scopes, set the `globalReaderGroup` attribute of the `zosconnect_zosConnectManager` element, and omit the `readerGroup` attribute of each individual API requester element that inherits it.

8. Update the server configuration or restart the server.

Results

Your z/OS Connect EE server is configured to perform authorization checks by using the z/OS Connect EE authorization interceptor, and SAF groups are assigned to each of the authorization levels.

User registries

To use authentication or authorization with z/OS Connect EE you require a user registry.

Before you begin these tasks, you should be familiar with

- [“API provider authentication and identification” on page 349](#) when z/OS Connect EE is acting as an API provider.
- [“API requester authentication and identification” on page 407](#) when z/OS Connect EE is acting as an API requester.

The steps required to configure a user registry in a z/OS Connect EE server depend on the type of user registry you have chosen. Select the appropriate task below:

How to configure a basic user registry

Configure a z/OS Connect EE server to use a basic registry, with a predefined set of users and groups.

Before you begin

- You should be familiar with

- “[API provider authentication and identification](#)” on page 349 when z/OS Connect EE is acting as an API provider.
- “[API requester authentication and identification](#)” on page 407 when z/OS Connect EE is acting as an API requester.
- You need to know the users and groups to be defined.
- You must have write access to the `server.xml` configuration file.

About this task

Configure a basic user registry that defines a set of users and groups directly into the `server.xml` configuration file, which is used for authentication and authorization. This configuration is typically used in a development environment.

The user IDs are "gjones", "mlee", and "rkumar" and the groups are "administrators" and "students". The user "gjones" is a member of the "administrators" group and users "mlee" and "rkumar" are members of the "students" group.

Procedure

Add the `basicRegistry` element to the `server.xml` configuration file.

Add user and group subelements to define each of your users and groups.

```
<basicRegistry id="basic" realm="customRealm">
    <user name="gjones" password="{$xor}Lz4sLCgwLTs=" />
    <user name="mlee" password="p@ssw0rd" />
    <user name="rkumar" password="pa$$w0rd" />
    <group name="administrators">
        <member name="gjones" />
    </group>
    <group name="students">
        <member name="mlee" />
        <member name="rkumar" />
    </group>
</basicRegistry>
```

For more information about `basicRegistry`, see the [Server configuration](#) section in the *IBM WebSphere Application Server for z/OS Liberty* documentation.

Be aware of the following considerations when you use this code sample:

- You must use unique names for your users and groups.
- Remove all leading and trailing spaces from the user and group names.
- If the user name or password contains characters other than US-ASCII, make sure that the file is saved by using UTF-8 character encoding.
- You can use the WebSphere Liberty profile `securityUtility encode` command to encode the password for each user. The `securityUtility` command-line tool is available in the `<installation_path>/wlp/bin` directory. When you run the `securityUtility encode` command, you either supply the password to encode as an input from the command line or, if no arguments are specified, the tool prompts you for the password. The tool returns the encoded value. Copy the encoded value and use it for the password attribute value. An example is shown on the entry for user "gjones" above.

For example, to encode the password `openSesame`, run the following command:

`securityUtility encode openSesame`

For more information, see `securityUtility` command in the *WebSphere Application Server for z/OS Liberty* documentation.

Results

The pre-defined set of users and groups are now defined in the user registry. After the server configuration is updated, or the server is started, this user registry will be available for use.

How to configure an LDAP user registry

Configure a Lightweight Directory Access Protocol (LDAP) user registry in a z/OS Connect EE server.

Before you begin

- You should be familiar with
 - [“API provider authentication and identification” on page 349](#) when z/OS Connect EE is acting as an API provider.
 - [“API requester authentication and identification” on page 407](#) when z/OS Connect EE is acting as an API requester.
- You need the following information about the LDAP server:
 - The type of LDAP server. For example, IBM Secure Directory Server or Microsoft Active Directory Server.
 - Connection details such as host, port and whether the connection to the LDAP server is secured with TLS.
 - For Microsoft Active Directory Server, the distinguished name (DN) for the application server, which is used to bind to the directory service and the bind password.
 - The base distinguished name (DN) which indicates the starting point for LDAP searches in the directory service.
- You must have write access to the `server.xml` configuration file.
- You need to know the users and groups that are to be granted access to access z/OS Connect EE.

About this task

You configure a Lightweight Directory Access Protocol (LDAP) user registry that references an existing LDAP server, so that its users and groups can be used for authentication and authorization.

Procedure

1. Follow the instructions in [Configuring LDAP user registries in Liberty](#) in the *WebSphere Application Server for z/OS Liberty* documentation.

References to Liberty refer to the z/OS Connect EE server.

For more information about the `ldapRegistry` element and its attributes, see [LDAP User Registry \(`ldapRegistry`\)](#) in the *WebSphere Application Server for z/OS Liberty* documentation.

The following example shows an LDAP registry for an IBM Secure Directory Server. Note that the `ldapType` element must be set to "IBM Tivoli Directory Server" rather than "IBM Secure Directory Server".

```
<ldapRegistry id="ldap" realm="SampleLdapIDSRealm"
    host="ourLDAP.ibm.com" port="389" ignoreCase="true"
    baseDN="o=mop,c=fr"
    ldapType="IBM Tivoli Directory Server">
    <idsFilters>
        userFilter="(&uid=%v)(objectclass=ePerson)"
        groupFilter="(&cn=%v)
            (|(objectclass=groupOfNames)
            (objectclass=groupOfUniqueNames)
            (objectclass=groupOfURLs)))"
        userIdMap="*:uid"
        groupIdMap="*:cn"
        groupMemberIdMap="ibm-allGroups:member;ibm-allGroups:uniqueMember;
            groupOfNames:member;groupOfUniqueNames:uniqueMember">
    </idsFilters>
</ldapRegistry>
```

2. After you have configured your `server.xml` configuration file with the `features` and `ldapRegistry` elements, restart your z/OS Connect EE server.

Check the messages.log file to ensure the LDAP registry feature has been installed and that there are no error messages resulting from a misconfigured ldapRegistry entry. The following information messages in the messages.log file show that the ldapRegistry feature has been installed:

```
CWIMK0009I: The user registry federation service is ready.  
CWWKS0008I: The security service is ready.  
CWWKF0012I: The server installed the following features: [... ldapRegistry-3.0 ...]
```

Results

The LDAP users and groups matching the defined filters are now defined in the user registry and can be used by the z/OS Connect EE server, for authentication and authorization.

How to activate and configure the SAF user registry

Activate the System Authorization Facility (SAF) user registry and configure it to use z/OS authorized services.

Before you begin

- You should be familiar with
 - [“API provider authentication and identification” on page 349](#) when z/OS Connect EE is acting as an API provider.
 - [“API requester authentication and identification” on page 407](#) when z/OS Connect EE is acting as an API requester.
- You must have write access to the `server.xml` configuration file.

About this task

You activate and configure the SAF user registry in the z/OS Connect EE server. This user registry has access to the user IDs and groups that are defined to SAF on the same z/OS LPAR as the z/OS Connect EE server, so they can be used for authentication and authorization.

By default, the SAF user registry uses unauthorized UNIX System Services services such as `__passwd` to perform authentication. For better performance, you can configure the SAF user registry to use authorized services such as `initACEE` to perform authentication by configuring the SAFCREDS resources.

Procedure

1. Define the SAFCREDS resources and permit the z/OS Connect EE server access to use z/OS authorized services.
For more information, see [“Configuring the Liberty Angel process and z/OS authorized services” on page 465](#).
2. Define a SAF user ID to act as the SAF user registry's unauthenticated user (the default value is `WSGUEST`).
For more information, see [“SAF unauthenticated user ID” on page 458](#).
3. Define the SAF APPL profile to be used by the server and permit the SAF unauthenticated user ID and all SAF user IDs that are to be authenticated, READ access to that profile.
See [Authenticating a user](#) in the topic [Accessing z/OS security resources using WZSSAD](#) of the *WebSphere Application Server for z/OS Liberty* documentation.
4. Activate the SAF user registry.
Add the `zosSecurity-1.0` feature into the `featureManager` element in the `server.xml` configuration file.

```
<featureManager>  
  ...  
  <feature>zosSecurity-1.0</feature>  
</featureManager>
```

5. Configure the SAF user registry.

Add a `safRegistry` element in the `server.xml` configuration file:

```
<safRegistry/>
```

For more information about the `safRegistry` element, see the [Server configuration](#) section in the *IBM WebSphere Application Server for z/OS Liberty* documentation.

6. Configure the server to control the operations of the SAF credentials.

The server uses the SAF APPL profile and SAF unauthenticated user ID that you defined in steps “[2](#)” on [page 456](#) and “[3](#)” on [page 456](#).

The default profile prefix value is BBGZDFLT. If you chose a custom value, you must specify that value on `profilePrefix` attribute of the `safCredentials` element in the `server.xml` configuration file.

The default SAF unauthenticated user ID value is WSGUEST. If you chose a custom value, you must specify that value on the `unauthenticatedUser` attribute of the `safCredentials` element in the `server.xml` configuration file.

For example, to use a custom profile prefix value of "MYPROFILE" and a custom SAF unauthenticated user ID of "MYGUEST", add the following element to the `server.xml` configuration file.

```
<safCredentials  
profilePrefix="MYPROFILE"  
unauthenticatedUser="MYGUEST"/>
```

If unauthorized users attempt to access the WLP z/OS System Security Access Domain (WZSSAD) and attribute `suppressAuthFailureMessages="false"` is specified, SAF authorization messages such as RACF ICH408I are displayed. For more information about `safCredentials`, see the [Server configuration](#) section in the *IBM WebSphere Application Server for z/OS Liberty* documentation.

To use the default values, omit the `safCredentials` element.

7. Ensure that the Liberty profile angel process is running.

To use z/OS authorized services, the server must be able to connect to the Liberty profile angel process. You created a started task to run the Liberty angel process and permit the z/OS Connect EE server to access it in step “[1](#)” on [page 456](#).

To start the angel process, start the associated started task. Enter the following command in SDSF:

```
/S BAQZANGL
```

For more information about starting the angel process and checking that it started successfully, see “[Configuring the Liberty Angel process and z/OS authorized services](#)” on [page 465](#).

8. Configure the z/OS Connect EE server to require an angel process.

Set `com.ibm.ws.zos.core.angelRequired` to true to require a successful connection to an angel process for the server startup to continue. For more details, see “[Configuring named angels](#)” on [page 468](#).

9. Start the z/OS Connect EE server.

Start, or restart the server if it was already running, so that it connects to the angel process. The following messages are written to the `messages.log` file:

CWWKB0122I: This server is connected to the default angel process.
CWWKB0103I: Authorized service group KERNEL is available.
CWWKB0103I: Authorized service group SAFCRED is available.

Other authorized services, as defined in previous steps of this task, should also be listed as available.

Results

The SAF user IDs and groups are now available to the SAF user registry and can be used by the z/OS Connect EE server, for authentication and authorization.

SAF unauthenticated user ID

This task shows you how to define the SAF unauthenticated user ID and group in RACF

Before you begin

If you are using a SAF user registry, it is necessary to specify a SAF user ID that represents the unauthenticated state. It is important to define this user ID correctly in your SAF registry.

If you are using a RACF SAF user registry, the unauthenticated user needs a unique default group (DFLTGRP) with no other user IDs connected to that group, an OMVS segment, but not a TSO segment, and the options NOPASSWORD, NOOIDCARD, and RESTRICTED.

If you have another SAF user registry, instead of RACF, then find the user ID options that are provided by that SAF registry that are equivalent to these RACF options.

The default authenticated user ID value is WSGUEST.

Procedure

1. Run the ADDGROUP command. The following example uses WSGUESTG as the group name.

```
ADDGROUP WSGUESTG SUPGROUP(SYS1) OWNER(SYS1) DATA('Unauthenticated User Group') OMVS(AUTOGID)
```

2. Run the ADDUSER command. The following example uses the default value of WSGUEST as the user ID name.

```
ADDUSER WSGUEST DFLTGRP(WSGUESTG) OWNER(SYS1) OMVS(AUTOUID HOME(/u/wsguest) PROGRAM(/bin/sh)) NAME('Unauthenticated User') NOPASSWORD NOOIDCARD RESTRICTED
```

The options NOPASSWORD and NOOIDCARD protect this user ID from being revoked by repeated attempts to guess the password.

The option RESTRICTED means that this user ID cannot gain access to protected resources unless it is explicitly permitted to access that resource, even if that resource has a general access setting of **UACC(READ)**.

Note: After the unauthenticated user ID is defined to the SAF registry, ensure that the user ID is authorized to use only the minimum number of SAF resources. If the z/OS Connect EE server uses the SAF APPL resource check to control which users can connect to the Liberty z/OS System Security Access Domain (WZSSAD), then the unauthenticated user ID must be given access to the APPL profile. For more information about WZSSAD, see [Accessing z/OS security resources using WZSSAD](#).

3. Run the PERMIT command.

```
PERMIT profilePrefix CLASS(APPL) ID(unauthenticated_user_ID) ACCESS(READ)
```

Where:

- *profilePrefix* is the value of the *profilePrefix* attribute that is specified on the *safCredentials* element in *server.xml*. The default value is BBGZDFLT.
- *unauthenticated_user_ID* is the SAF user ID you defined as the unauthenticated user. The default value is WSGUEST.

Note: If you chose a user ID value other than WSGUEST, you must specify the value on the *unauthenticatedUser* attribute of the *safCredentials* element in the *server.xml* file for each of your z/OS Connect EE servers.

4. Give READ access to the SAF unauthenticated user ID.

The SAF unauthenticated user ID, for example WSGUEST, needs READ access to the APPLID in the SAF APPL class that is used by the z/OS Connect EE server for its SAF domain. For detailed instructions, see the section *Authenticating a user of Accessing z/OS security resources using WZSSAD* in the *WebSphere Application Server for z/OS Liberty* documentation.

Using a Trust Authentication Interceptor (TAI) to allow selected unauthenticated requests

Where z/OS Connect EE is configured for basic authentication, but selected requests do not present the required credential, a Trust Authentication Interceptor (TAI) can be developed, configured and deployed with a z/OS Connect EE server to allow selected "unauthenticated" requests to be processed.

For example, where an API gateway component has already authenticated requests that presented a credential, such as a client certificate, and no associated credential is available for the onward request to z/OS Connect EE, a TAI can be developed to identify such requests and allow them to be processed under a fixed application or task identity. Requests that do not match the criteria of the TAI must still present the required credential to proceed. The criteria by which a TAI decides to block or allow a request is entirely due to the design of the TAI implementation.

In the event that the TAI allows an unauthenticated request to be processed, it is possible, depending upon overall configuration, that one or more warning messages are written to the log for each unauthenticated request. For example:

CWWKS1100A: Authentication did not succeed for user ID cn=unknown,o=ibm,c=us. An invalid user ID or password was specified.

ACF2 users might see one instance of the ACF1097 message per associated request:

ACF01097 NO USERID SPECIFIED ON SYSTEM ENTRY VALIDATION REQUEST

To avoid these warning messages, use the Liberty configuration element, `webAppSecurity`, to set the `useAuthenticationDataForUnprotectedResource` attribute to `false`. For example:

```
<webAppSecurity useAuthenticationDataForUnprotectedResource="false"/>
```

For more information about this element, see [Web Container Application Security \(webAppSecurity\)](#) in the [WebSphere Application Server for z/OS Liberty](#) documentation.

For more information about creating a TAI, see [Developing a custom TAI for Liberty](#) in the [WebSphere Application Server for z/OS Liberty](#) documentation.

Security capabilities using z/OS Connect EE

z/OS Connect EE security is built on top of standard protocols like TLS, and security capabilities in z/OS, the IBM Java SDK for z/OS and Liberty z/OS. This section provides a brief review of these capabilities.

To secure your z/OS Connect EE environment you need to consider the security options available to you. The following topics describe various security capabilities supported by z/OS Connect EE for confidentiality and integrity.

Transport Layer Security (TLS)

Learn about TLS.

The TLS protocol consists of two layers, the TLS Record Protocol and the TLS Handshake Protocol.

- The TLS Record Protocol provides connection security and has the following properties:
 - The connection is private. Secret key cryptography is used for data encryption. The keys for this secret key encryption are generated uniquely for each connection and are based on a secret that is negotiated by a handshake.
 - The connection is reliable. Message transport includes a message integrity check by using a keyed-hashed MAC (HMAC). Secure hash functions such as SHA-1 or MD5, are used for MAC computations.
- The TLS Handshake Protocol operates with the TLS Record Protocol to allow the z/OS Connect EE server and REST client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before any data is transmitted. The TLS Handshake Protocol provides connection security that has the following properties:
 - The peer's identity can be authenticated by using public key cryptography.

- The negotiation of a shared secret is secure and the negotiated secret is not available to eavesdroppers. For any authenticated connection, the secret cannot be obtained, even by an attacker who can intercept the connection.
- The negotiation is reliable. No attacker can modify the negotiation communication without being detected by the parties in communication.

The TLS protocol supersedes the Secure Sockets Layer (SSL) protocol and the terms are often used interchangeably. This documentation uses the term TLS to refer to either TLS or SSL, unless it is explicitly referencing a version of the specification or the name of a `server.xml` configuration file element.

An encrypted TLS connection is established after a handshake takes place between the client and the server. A digital certificate for the server is passed to the client. The client knows that the server can be trusted by which Certificate Authority (CA) signed the server certificate. If the CA is trusted, then the server certificate is trusted, and the TLS session is set up. For more information about the TLS handshake, see [The SSL Protocol](#) in the *IBM SDK Java Technology Edition* documentation.

For TLS requests sent to a z/OS Connect EE server, the z/OS Connect EE server acts as the "server" and the "client" is either the REST client when z/OS Connect EE acts as an API provider, or CICS, IMS or a z/OS application when z/OS Connect EE acts as an API requester.

For TLS requests sent from a z/OS Connect EE server, the z/OS Connect EE server acts as the "client" and the "server" is either the System of Record when z/OS Connect EE acts as an API provider, or a RESTful API endpoint when z/OS Connect EE acts as an API requester.

To learn how z/OS Connect EE can use TLS, see either ["API provider confidentiality and integrity"](#) on page 336 when z/OS Connect EE is acting as an API provider, or ["API requester confidentiality and integrity"](#) on page 394 when z/OS Connect EE is acting as an API requester.

Application Transparent Transport Layer Security (AT-TLS)

Learn about AT-TLS.

Application Transparent Transport Layer Security (AT-TLS) is a capability of z/OS Communications Server that can create a secure session on behalf of z/OS Connect EE (or other z/OS applications). Instead of implementing TLS in z/OS Connect EE, AT-TLS provides encryption and decryption of data based on policy statements that are coded in the Policy Agent. z/OS Connect EE sends and receives cleartext (unencrypted data) as usual while AT-TLS encrypts and decrypts data at the TCP transport layer.

AT-TLS supports different types of application:

- An unaware application is unaware that AT-TLS is performing encryption or decryption of data.
- An aware application is aware of AT-TLS and can query information such as AT-TLS status and the partner certificate.
- A controlling application is aware of AT-TLS and can control the secure session.

z/OS Connect EE is an unaware AT-TLS application and therefore does not have access to the partner certificate. This means that a z/OS subsystem cannot use a client certificate to authenticate with z/OS Connect EE when the connection between the z/OS subsystem and z/OS Connect EE is secured using AT-TLS.

For more information on AT-TLS see [Application Transparent Transport Layer Security data protection](#) in the *z/OS Communications Server* documentation.

To learn how z/OS Connect EE can use AT-TLS, see either ["API provider confidentiality and integrity"](#) on page 336 when z/OS Connect EE is acting as an API provider, or ["API requester confidentiality and integrity"](#) on page 394 when z/OS Connect EE is acting as an API requester.

Java Secure Sockets Extension (JSSE)

Learn about JSSE.

JSSE provides a framework and Java implementation that handles the handshake negotiation and protection capabilities that are provided by TLS. JSSE relies on X.509 certificate-based asymmetric key

pairs for secure connection protection and some data encryption. Key pairs effectively encrypt session-based secret keys that encrypt larger blocks of data.

Java Cryptography Extension (JCE) is a standard extension to the Java Platform that provides the underlying implementation for cryptographic services, including encryption, key generation, and Message Authentication Codes (MAC).

The IBM Java SDK for z/OS provides two main JCE providers, IBMJCE, and IBMJCECCA. The default provider is IBMJCE, which uses CPACF. The IBMJCECCA provider uses ICSF to drive the IBM Crypto Express card.

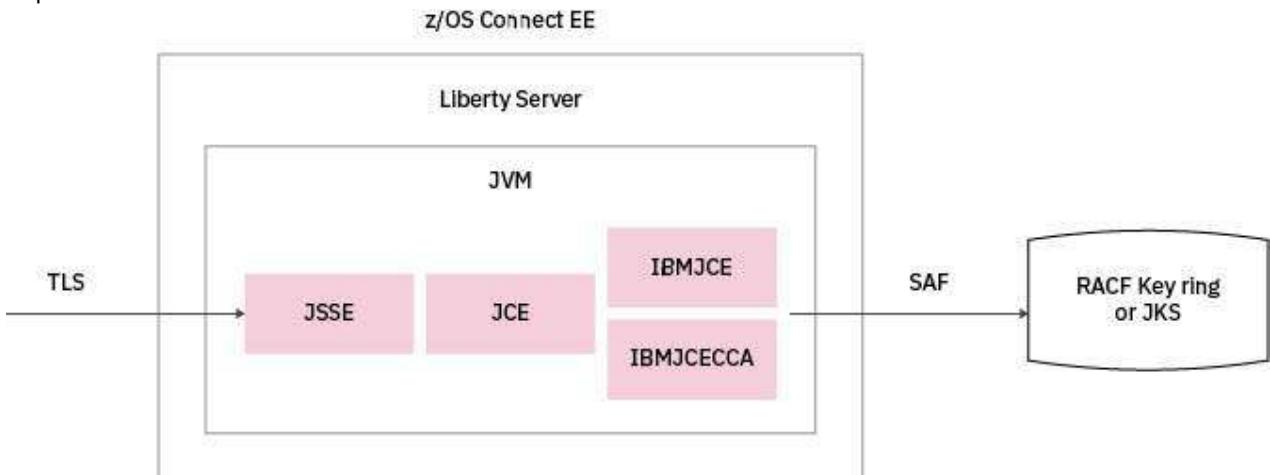


Figure 99. The two main JCE providers used for cryptographic services

The IBM Java SDK also supports the IBMJCEHYBRID provider, which is designed to use cryptographic hardware and processors when they are available but continues without those cryptographic features when they are not available.

The use of the JCE provider is controlled by editing the list of security providers in the `java.security` file for the JVM.

For more information about hardware cryptography, see [“Hardware cryptography” on page 463](#).

To learn how z/OS Connect EE can use JSSE, see either [“API provider confidentiality and integrity” on page 336](#) when z/OS Connect EE is acting as an API provider, or [“API requester confidentiality and integrity” on page 394](#) when z/OS Connect EE is acting as an API requester.

Keystores and truststores

Learn about keystores and truststores.

Keystores and truststores are repositories that contain cryptographic artifacts like certificates and private keys that are used for cryptographic protocols such as TLS.

A *keystore* contains personal certificates, plus the corresponding private keys that are used to identify the owner of the certificate.

For TLS, a *personal certificate* represents the identity of a TLS endpoint. Both the client (for example, a REST client) and the server (for example, a z/OS Connect EE server) might have personal certificates to identify themselves.

A *truststore* contains the signer certificates (also known as certificate authority certificates) which the endpoint trusts.

A *signer certificate* contains a *public key*, which is used to validate personal certificates. By installing the server's signer certificate into the client's truststore, you are allowing the client to trust the server when it establishes a TLS connection. The same principle is true for a server to trust a client when TLS client authentication is enabled.

z/OS Connect EE supports Java KeyStores (JKS), Public Key Cryptography Standards #12 (PKCS12), and SAF key rings.

JKS keystore type

A Java Keystore (JKS) is a common keystore type that is used for Java environments because it is easier to set up. JKSs use files with a .jks extension that are stored in the zFS file system. The JKS is referenced by the keyStore element in the server.xml configuration file. You can use a JKS for both keystores and truststores.

PKCS12 keystore type

Public Key Cryptography Standards #12 (PKCS12) is an industry standard keystore type, which makes it compatible with other products. PKCS12 keystores use files with a .p12 extension that are stored in the zFS file system. The PKCS12 keystore is referenced by the keyStore element in the server.xml configuration file. You can use a PKCS12 type for both keystores and truststores.

PKCS12 is the default keystore type in z/OS Connect EE.

SAF key rings

You can also use SAF key rings as keystores and truststores. SAF key rings are under the control of SAF security administrators. z/OS Connect EE supports the following keystore types for SAF key rings:

- JCERACFKS when the IBMJCE security provider is used.
- JCECCARACFKS when the IBMJCECCA security provider is used.
- JCEHYBRIDRACFKS when the IBMJCEHYBRID security provider is used.

For more information about using different Java Cryptography Extension (JCE) security providers, see [“Hardware cryptography” on page 463](#).

The SAF key ring is referenced by the keyStore element in the server.xml configuration file, and the type of keystore is specified in the location and type attributes.

Table 39. Relationship between JCE provider, keystore location, and keystore type.. Shows the location and type of keystore for different security providers		
JCE provider	server.xml	
	ssl keystore location attribute	ssl keystore type attribute
IBMJCE	location="safkeyring:/// "	type="JCERACFKS"
IBMJCECCA	location="safkeyringhw:// /"	type="JCECCARACFKS"
IBMJCEHYBRID	location="safkeyringhybrid:/// "	type="JCEHYBRIDRACFKS"

To learn how z/OS Connect EE can use keystores and truststores, see either [“API provider confidentiality and integrity” on page 336](#) when z/OS Connect EE is acting as an API provider, or [“API requester confidentiality and integrity” on page 394](#) when z/OS Connect EE is acting as an API requester.

Cipher suites

Learn about cipher suites.

A cipher suite is a set of cryptographic algorithms that are used to create keys and encrypt information.

During the TLS handshake, the TLS protocol and data exchange cipher are negotiated.

- The client (for example, a REST client) sends the highest version of TLS that it supports and the list of the ciphers that it supports.

- The server (for example, a z/OS Connect EE server) chooses the highest version of TLS that is jointly supported by itself and the client, and one cipher that is jointly supported by itself and the client.

Use the most recent version of TLS as it generally contains fixes to previous versions and brings enhancements such as the support for stronger cipher suites. To control the TLS version that is supported by the z/OS Connect EE server, the `sslProtocol` attribute on the `ssl` element can be updated in `server.xml`. To modify the list of ciphers that are supported by the z/OS Connect EE server, a list of ciphers can be specified in the `enabledCiphers` attribute on the `ssl` element.

Note: The choice of cipher and key length have an impact on performance. Some algorithms are more CPU-intensive than others.

To learn how z/OS Connect EE can use cipher suites, see either “[API provider confidentiality and integrity](#)” on page 336 when z/OS Connect EE is acting as an API provider, or “[API requester confidentiality and integrity](#)” on page 394 when z/OS Connect EE is acting as an API requester.

Hardware cryptography

Learn about hardware cryptography.

z/OS Connect EE can be configured to use cryptographic hardware.

Two cryptographic hardware devices are available on IBM Z, the CP Assist for Cryptographic Function (CPACF) and the IBM Crypto Express cards. These devices are supported in different ways.

CPACF is a set of cryptographic instructions available on all CPs, including zIIPs, IFLs, and General Purpose CPUs. Various symmetric algorithms are supported by the CPACF including DES, 3DES, and AES-CBC, and SHA-based digest algorithms. CPACF provides the potential for significantly improved performance for these operations.

The IBM JVM default security provider (IBMJCE) with IBM Java 8 automatically detects and uses the CPACF. However, to benefit from the IBM Crypto Express cards you need to configure an alternative security provider, either IBMJCECCA or IBMJCEHYBRID.

z/OS Connect EE

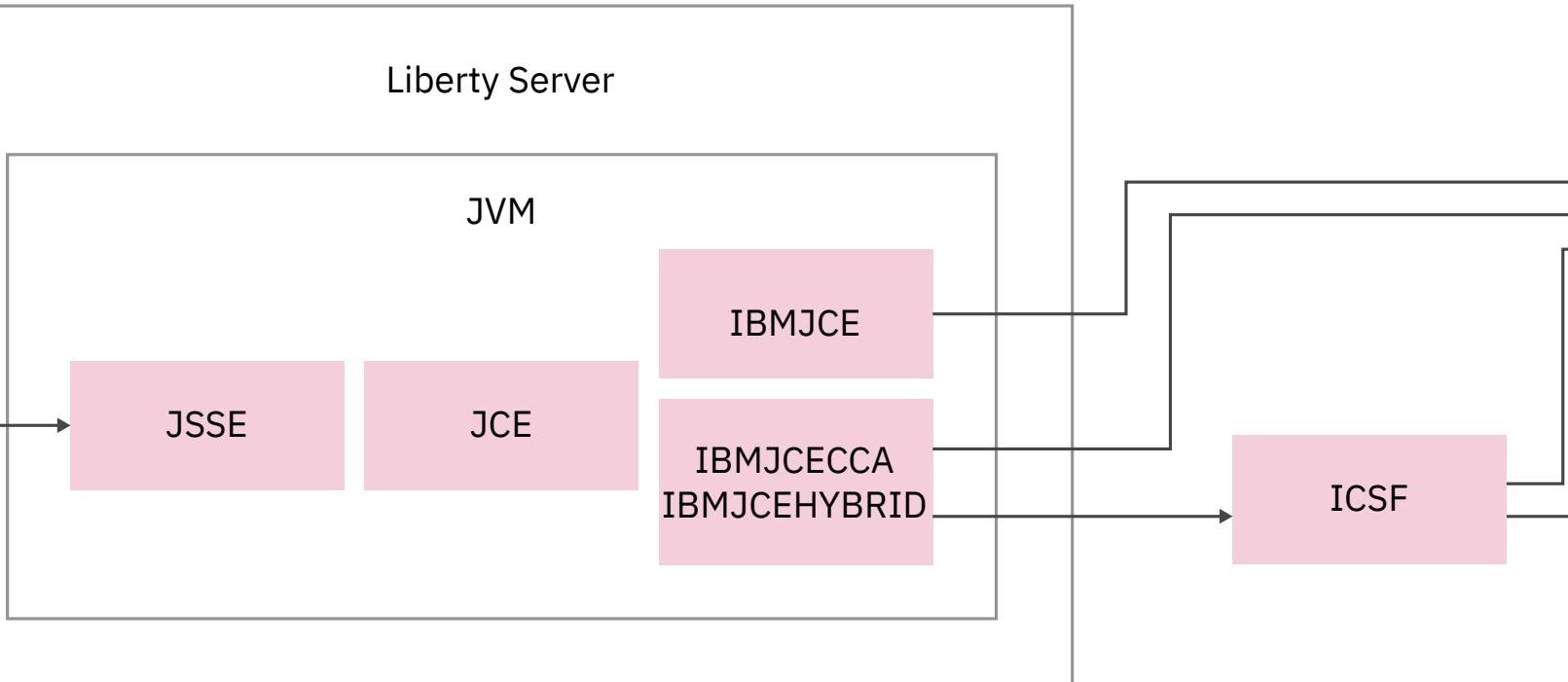


Figure 100. Encryption using the CPACF

Note: For IBMJCECCA to initialize, ICSF must be started and at least one coprocessor must be available. If IBMJCECCA cannot be loaded at initialization, the next provider in the Java security provider list is loaded. You must define the order in which to use security providers in the `java.security` configuration file.

The IBM Crypto Express cards are optional I/O attached cards that implement additional cryptographic functions. On an IBM z14, this feature is available as a Crypto Express 6S (CEX6S) adapter, or Crypto Express 5S (CEX5S).

By default, the Crypto Express card is a coprocessor (CEX6C) and can support a wider range of callable services that include secure key and clear key support for PKA decrypt, digital signature verify, digital signature generate, including RSA and ECC variants. Alternatively, the card can be configured as an accelerator (CEXCA). In this mode, the card supports only three clear key cryptographic APIs, associated with RSA public key encryption, decryption, and verification. When the cryptographic coprocessor is configured as an accelerator it provides better throughput at the expense of supporting fewer services.

For more information about configuring the JCE providers on z/OS to support hardware cryptography, see [z/OS Java Security Frequently Asked Questions](#).

To learn how z/OS Connect EE can use hardware cryptography, see either [“API provider confidentiality and integrity” on page 336](#) when z/OS Connect EE is acting as an API provider, or [“API requester confidentiality and integrity” on page 394](#) when z/OS Connect EE is acting as an API requester.

SP800-131a

Learn about the SP800-131a specification.

You can set up z/OS Connect EE to meet the SP800-131a specification originated by the National Institute of Standards and Technology (NIST). This specification requires longer key lengths and stronger cryptography.

For more information, see [Setting up Liberty to run in SP800-131a in the WebSphere Application Server for z/OS Liberty documentation](#).

To learn how z/OS Connect EE can use SP800-131a, see either “[API provider confidentiality and integrity](#)” on page 336 when z/OS Connect EE is acting as an API provider, or “[API requester confidentiality and integrity](#)” on page 394 when z/OS Connect EE is acting as an API requester.

Configuring the Liberty Angel process and z/OS authorized services

You need to configure the Liberty Angel process so that z/OS Connect Enterprise Edition can use z/OS authorized services.

About this task

To use z/OS authorized services such as System Authorization Facility (SAF), Workload Manager (WLM), Resource Recovery services (RRS), SVCDUMP, or WebSphere Optimized Local Adapters (WOLA) you must set up a Liberty Angel process and grant access for your z/OS Connect EE server to use these services. SAF is used by Liberty security mechanisms to call RACF®. RRS is used by IBM® MQ resource adapter when the connection to IBM® MQ is made in BINDINGS mode.

The Liberty profile Angel process must be run as a started task, but is lightweight, has no configuration or TCP ports, and consumes almost no CPU.

To create the Angel process started task, you must customize the sample JCL and create SAF definitions to associate the started task with a user ID and authorize your z/OS Connect EE server to use the z/OS authorized services. The following examples use RACF commands. Two copies of the sample JCL are provided, *h1q.SBAQSAMP(BAQZANGL)* and *<installation_path>/wlp/templates/zos/procs/bbgzangl.jcl* but they provide the same function.

Note: Each LPAR can have multiple named Angel processes but only one default Angel process. Ensure that the Angel processes are running at the most recent installed level of Liberty on the LPAR. If a Liberty server instance that is embedded in z/OS Connect EE server connects to an Angel process that is running at an earlier service level, some features of the server might not be available. For more information about named angels, see [“Configuring named angels” on page 468](#).

Procedure

1. Create the JCL start procedure for the Angel process.
 - a) To set up the started task, customize the sample JCL provided in *<h1q>.SBAQSAMP(BAQZANGL)* and add it to your PROCLIB library.
 - b) Customize the sample JCL by updating the SET ROOT value to your z/OS Connect EE installation directory.

The sample JCL defines the default directory:

```
SET ROOT='/usr/lpp/IBM/zosconnect/v3r0/wlp'
```

In the following steps, work with your security administrator to create the necessary authorizations and artifacts for the Angel process to run as a started task and to authorize your z/OS Connect EE server to use z/OS authorized services.

2. Started tasks must be associated with a user ID. If you do not have a suitable user ID, use the following commands to create a new user ID and group.

For example: to define a user ID called *angel_id* in a group called *admin_group*, your security administrator needs to enter the following commands:

```
ADDGROUP admin_group OMVS(GID(gid))
ADDUSER angel_id DFLTGRP(admin_group) OMVS(UID(uid) HOME(/u/angel_id) PROGRAM(/bin/sh))
NAME('Liberty angel') NOPASSWORD NOOIDCARD
```

The user ID used to run the angel process requires read and execute permissions to the z/OS Connect EE UNIX System Services installation directory.

3. Grant the required SAF authorization to associate the user ID with the started task.

For example:

```
RDEF STARTED BAQZANGL.* UACC(NONE) STDATA(USER(angel_id) GROUP(admin_group)
PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
SETROPTS RACLIST(STARTED) REFRESH
```

4. Create a set of SAF SERVER profiles to grant your z/OS Connect EE server authority to use the required z/OS authorized services.

Define the following SAF SERVER profiles. Grant your z/OS Connect EE server user ID READ access to each, by using the following commands, where *server_id* is the user ID used to run the z/OS Connect EE server started task.

You can authorize access at a group level, by replacing *server_id* with the name of the group. The user ID used to run the z/OS Connect EE server started task must be connected to this group.

- SERVER profile for the angel process to authorize the *server_id* user ID read access to it. This action grants a z/OS Connect EE server access to the angel process, which is required for the z/OS authorized services. You can create a named or unnamed angel server profile. If you are using both named and unnamed angels, you must define an angel server profile for each.
 - To create an unnamed angel server profile and enable a server that is running as *server_id* to connect to it, enter the following commands.

```
RDEF SERVER BBG.ANGEL UACC(NONE)
```

```
PERMIT BBG.ANGEL CLASS(SERVER) ACCESS(READ) ID(server_id)
```

- To create a named angel server profile and enable a server that is running as *server_id* to connect to it, enter the following commands.

```
RDEF SERVER BBG.ANGEL.<namedAngelName> UACC(NONE)
```

```
PERMIT BBG.ANGEL.<namedAngelName> CLASS(SERVER) ACCESS(READ) ID(server_id)
```

The profile name that you specify for the *namedAngelName* variable is the name of the new angel. You can use generic profiles such as BBG.ANGEL.* to grant a user ID access to multiple named angels.

- SERVER profile for the authorized module BBGZSAFM to allow server access to z/OS authorized services.

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE)
```

```
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) ACCESS(READ) ID(server_id)
```

- SERVER profile for the authorized client module BBGZSCFM.

```
RDEF SERVER BBG.AUTHMOD.BBGZSCFM UACC(NONE)
```

```
PERMIT BBG.AUTHMOD.BBGZSCFM CLASS(SERVER) ACCESS(READ) ID(server_id)
```

- SERVER profile for WLM services (ZOSWLM).

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.ZOSWLM UACC(NONE)
```

```
PERMIT BBG.AUTHMOD.BBGZSAFM.ZOSWLM CLASS(SERVER) ACCESS(READ) ID(server_id)
```

- SERVER profile for RRS transaction services (TXRRS).

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.TXRRS UACC(NONE)
```

```
PERMIT BBG.AUTHMOD.BBGZSAFM.TXRRS CLASS(SERVER) ACCESS(READ) ID(server_id)
```

- SERVER profile for SVCDUMP services (ZOSDUMP).

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.ZOSDUMP UACC(NONE)
```

```
PERMIT BBG.AUTHMOD.BBGZSAFM.ZOSDUMP CLASS(SERVER) ACCESS(READ) ID(server_id)
```

- SERVER profile for IFAUSAGE services (PRODMGR)

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.PRODMGR UACC(NONE)
```

```
PERMIT BBG.AUTHMOD.BBGZSAFM.PRODMGR CLASS(SERVER) ACCESS(READ) ID(server_id)
```

- SERVER profile for SAF authorized user registry services and SAF authorization services (SAFCRED).

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED UACC(NONE)
```

```
PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED CLASS(SERVER) ACCESS(READ) ID(server_id)
```

Note: If you use the IBM MQ for z/OS service provider, you must grant WebSphere Liberty Profile ALTER access to the MVSADMIN.RRS.COMMANDS resource in the RACF® FACILITY class.

5. Optional: If you wish to enable the AsyncIO on z/OS (ZOSAIO) service, configure security to permit your z/OS Connect EE server to use the authorized AsyncIO service.

For example,

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.ZOSAIO UACC(NONE)
```

```
PERMIT BBG.AUTHMOD.BBGZSAFM.ZOSAIO CLASS(SERVER) ACCESS(READ) ID(server_id)
```

For more information, see “[Asynchronous TCP/IP sockets I/O for Liberty \(AsyncIO\)](#)” on page 205

6. Optional: If you are using WOLA, you must also create the following profiles.

- SERVER profiles for the optimized local adapter authorized service.

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.LOCALCOM UACC(NONE)
```

```
PERMIT BBG.AUTHMOD.BBGZSAFM.LOCALCOM CLASS(SERVER) ACCESS(READ) ID(server_id)
```

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.WOLA UACC(NONE)
```

```
PERMIT BBG.AUTHMOD.BBGZSAFM.WOLA CLASS(SERVER) ACCESS(READ) ID(server_id)
```

- SERVER profiles for optimized local adapter authorized client service.

```
RDEF SERVER BBG.AUTHMOD.BBGZSCFM.WOLA UACC(NONE)
```

```
PERMIT BBG.AUTHMOD.BBGZSCFM.WOLA CLASS(SERVER) ACCESS(READ) ID(server_id)
```

7. Refresh to activate the definitions:

```
SETROPTS RACLIST(SERVER) REFRESH
```

8. Start the Angel process as a started task:

From the z/OS operator console, enter the following command:

```
S BAQZANGL
```

The following log messages indicate that the Angel process started successfully:

```
IRR812I PROFILE BAQZANGL.* (G) IN THE STARTED CLASS WAS USED  
TO START BAQZANGL WITH JOBNAME BAQZANGL.  
$HASP100 BAQZANGL ON STCINRDR  
IEF695I START BAQZANGL WITH JOBNAME BAQZANGL IS ASSIGNED TO  
USER angel_id, GROUP admin_group  
$HASP373 BAQZANGL STARTED  
CWWKKB0056I INITIALIZATION COMPLETE FOR ANGEL
```

Leave the Angel process running for any z/OS Connect EE server that requires access to z/OS authorized services. To stop the Angel process, enter the following command at the z/OS operator console:

```
P BAQZANGL
```

Note: Best practice is to NEVER cancel the angel process. Server tasks and application tasks can be dependant on the angel process running. However, there are rare cases in which to avoid a severe system shutdown, such as a re-IPL, you might find it necessary to cancel the angel process. If it becomes necessary to cancel the angel started task or if the angel started task abends, your system administrator must cancel all the servers with applications that depend on the angel started task, including z/OS Connect EE. If servers and applications are left running after the angel started task is stopped, a server hang condition can occur.

Configuring named angels

Configuring multiple angel processes in an LPAR.

About this task

Liberty V16.0.0.4 added support for concurrent angel processes on the same LPAR. At startup, each angel is identified by a unique name and an angel process with no name is used as the default process.

All Liberty servers, including a z/OS Connect EE server, can be configured to select which angel process is used to provide z/OS authorized services by specifying the name in the property

com.ibm.ws.zos.core.angelName. If no angel process name is specified by a Liberty server, then the default process, one with no name is selected. Another property,

com.ibm.ws.zos.core.angelRequired can be set to require a successful connection to an angel process for the server startup to continue. If this property is set, and an angel process is not available, the Liberty server stops.

Note:

1. All files must be EBCDIC encoded.
2. If you are using both named and unnamed angels, you must define an angel server profile for each.
3. Ensure that the version of the JCL you use to start the Angel process supports the NAME parameter.
Two copies of the sample JCL are provided, hlq.SBAQSAMP(BAQZANGL) and
<installation_path>/wlp/templates/zos/procs/bbgzangl.jcl but they provide the same function.
4. The Liberty documentation describes another method that uses a `bootstrap.properties` file in the server's configuration directory, but this method is supported only in z/OS Connect EE V3.0.5 and later. Earlier releases support only the `JVM_OPTIONS` method.

Procedure

1. Define SERVER SAF profiles and grant permissions to the SAF user ID that the z/OS Connect EE server is running under, by issuing the following commands:

For example, if the angel process is started with a name of PRODUCTION, then a SAF SERVER profile for this name, BBG . ANGEL . PRODUCTION must be defined and the z/OS Connect EE server that is running under identity USER1 must be given READ access as in the following definition.

```
RDEFINE SERVER BBG.ANGEL.PRODUCTION UACC(NONE) OWNER(SYS1)
PERMIT BBG.ANGEL.PRODUCTION CLASS(SERVER) ACCESS(READ) ID(USER1)
SETROPTS RACLIST(SERVER) REFRESH
```

Tip: You can use generic profiles such as BBG . ANGEL . * to grant a user ID access to multiple named angels.

The **SETROPTS** command is issued to refresh the profiles.

2. Restart z/OS Connect EE to use the new profile.
3. Define a named angel by coding the **NAME** parameter of the operator **START** command.

The angel name can be 1 - 54 characters in length, and must use only the following characters: 0-9 A-Z ! # \$ + - / : < > = ? @ [] ^ _ { } | ~ The angel name must be unique on the LPAR. Two angels with the

same name cannot be running concurrently on the LPAR. You can have a maximum of 4092 named angels on an LPAR.

The following steps are required only if you run z/OS Connect EE as a stand-alone server.

If you run z/OS Connect EE embedded in CICS, you must add these directives to the JVMServer profile that is configured for the z/OS Connect EE server.

Choose one of the following methods to configure the server for named angels. Choose either **JVM_OPTIONS** or bootstrap properties.

4. Optional: Use **JVM_OPTIONS** to configure the server for named angels.

- a) Create a file called <name>.options in the z/OS Connect EE Liberty server configuration directory.

This file defines the Java system properties **com.ibm.ws.zos.core.angelName** and **com.ibm.ws.zos.core.angelRequired**. In this example, the angel name is PRODUCTION. The angel name is case-sensitive and must be all uppercase. For example,

```
-Dcom.ibm.ws.zos.core.angelName=PRODUCTION  
-Dcom.ibm.ws.zos.core.angelRequired=true
```

- b) Edit the **JVM_OPTIONS** parameter of the STDENV JCL procedure to add a reference to this options file.

```
//STDENV DD *  
BPX_SHAREAS=YES  
JAVA_HOME=/usr/lpp/java/J8.0_64/  
WLP_USER_DIR=/var/zosconnect  
JVM_OPTIONS=-Xoptionsfile=/var/zosconnect/servers/<serverName>/<name>.options  
/*
```

5. Optional: Use bootstrap properties file to configure the server for named angels.

- a) Edit the **bootstrap.properties** file in the server configuration.

If you do not already have a file called **bootstrap.properties** in the z/OS Connect EE server configuration directory, create one now. The file can be in ASCII or EBCDIC. This file defines the properties **com.ibm.ws.zos.core.angelName** and **com.ibm.ws.zos.core.angelRequired**. The angel name is case-sensitive and must be all uppercase.

For example, if the angel name is PRODUCTION, then the **bootstrap.properties** file would contain the following statements,

```
com.ibm.ws.zos.core.angelName=PRODUCTION  
com.ibm.ws.zos.core.angelRequired=true
```

Results

When z/OS Connect EE starts with the **com.ibm.ws.zos.core.angelRequired** property set, an angel process is requested. If the Liberty server cannot reach an angel process, it stops. Because **com.ibm.ws.zos.core.angelName** is also defined, the Liberty server attempts to connect to that specific angel process, in this case, PRODUCTION. If Liberty fails to connect to the PRODUCTION angel process, it stops. If you have **com.ibm.ws.zos.core.angelRequired** but you don't specify a value for **com.ibm.ws.zos.core.angelName**, the Liberty server connects to any angel it can. If it cannot reach any angel process, it stops.

Chapter 10. Operating

Operating z/OS Connect EE

Starting and stopping z/OS Connect EE

Use these commands to start and stop your z/OS Connect EE server.

Use a started procedure to run your z/OS Connect EE server.

Setting up a started task to run z/OS Connect EE servers

A single instance of the started task JCL can be used to start multiple z/OS Connect EE servers because the server name is passed as a parameter on the start command. You might require multiple copies of the started task JCL, if your z/OS Connect EE servers require different settings, for example: JVM_OPTIONS.

To set up the started task, customize the sample JCL provided in `<hlq>.SBAQSAM`(BAQSTRT) and add it to your PROCLIB library. The sample BAQSTRT JCL turns on the heap pools algorithms, which can improve performance for C/C++ code that might be called by z/OS Connect EE or Liberty functions. For more information, see the z/OS documentation [Using heap pools to improve performance](#) and the two options `HEAPPOOLS` and `HEAPPOOLS64`.

The started task must be associated with a SAF user ID with an OMVS segment defined.

- For RACF, specify the OMVS segment parameters HOME and PROGRAM, for example: `HOME(/u/<myuser>)` and `PROGRAM(/bin/sh)`. The values are case-sensitive.
- For non-IBM External Security Managers, specify the OMVS segment HOME parameter value to be the location where the server instances are stored. This location should be the same as the value you specify for the WLP_USER_DIR environment variable. For example, if you are using the default WLP_USER_DIR value of `/var/zosconnect`, set `HOME(/var/zosconnect)`. Additionally set any other OMVS segment properties, for example: OMVSPGM, as required by your security manager.

Then, enter the SAF command to associate the name of the started task procedure with a user ID. For example, to define the BAQSTRT procedure name to run under the user ID `<userid>`, use the following RACF command:

```
RDEF STARTED BAQSTRT.* UACC(NONE) STDATA(USER(<userid>) GROUP(<group>)
PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
```

If you use a non-IBM External Security Manager and receive a CHECK_PROC_OWNER error message when you start z/OS Connect EE as a started task, check that the HOME parameter of the OMVS segment of the started task user ID is correctly defined.

Starting a z/OS Connect EE server

To start the server started procedure, enter the following command:

```
/S BAQSTRT,PARMS='<serverName> [--clean]'
```

Note: You must use the **System Command Extension** window within SDSF to preserve the case of your server name. From SDSF, enter / to open the **System Command Extension** window, then enter the **START** command. If you have restored z/OS Connect EE from a later version, then you must specify the --clean option when starting a server.

Stopping a z/OS Connect EE server

To stop the server started procedure, enter the following command:

```
/P <JOB NAME>
```

Note: Liberty's default behavior is to open an ephemeral command listener port at startup, but this port is not needed by z/OS Connect EE, so z/OS Connect EE closes this port at startup, unless it is specifically defined in `bootstrap.properties`.

System automation

System automation of z/OS software usually involves trapping messages issued to the z/OS console and performing actions in response to those messages. For example, to confirm a z/OS Connect EE server has started or shutdown successfully.

Specifying z/OS Connect EE messages to be written to z/OS operator console

By default, z/OS Connect EE logs all messages to the UNIX System Services file `messages.log`, and only a subset of these messages are written to the z/OS console. However, the server configuration file element `zosLogging` can be used to specify additional server messages to be logged to the z/OS console. For more information about the `zosLogging` server configuration element, see [z/OS Logging \(zosLogging\)](#) in the *WebSphere Application Server* documentation.

For example, you might find the following messages useful:

- BAQR7000I: z/OS Connect API archive file `apiName` installed successfully.
This message indicates that an API was installed successfully.
- SRVE0250I: Web Module z/OS Connect has been bound to `default_host`.
CWWKT0016I: Web application available (`default_host`): `http://hostname:port/`
These messages indicate that a z/OS Connect EE server has completed start up successfully.
- CWWKE1101I: Server quiesce complete.
This message indicates that a z/OS Connect EE server has completed shutdown successfully

To log these messages to the z/OS console, specify the following in the server configuration element:

```
<zosLogging enableLogToMVS="true" wtoMessage="BAQR7000I,SRVE0250I,CWWKT0016I,CWWKE1101I"/>
```

When the z/OS Connect EE server is started with a deployed API, the following messages are written to the z/OS console:

```
N 4000000 MVxx yyddd hh:mm:ss.sss J0Bnnnnn nnnnnnn +CWWKE0001I: The server serverName has been
launched.
N 4000000 MVxx yyddd hh:mm:ss.sss J0Bnnnnn nnnnnnnn +BAQR7000I: z/OS Connect API archive file
apiName
                                         installed successfully.
N 4000000 MVxx yyddd hh:mm:ss.sss J0Bnnnnn nnnnnnnn +CWWKF0011I: The server serverName is ready to
run
                                         a smarter planet.
N 4000000 MVxx yyddd hh:mm:ss.sss J0Bnnnnn nnnnnnnn +SRVE0250I: Web Module z/OS Connect has been
bound
                                         to default_host.
M 4000000 MVxx yyddd hh:mm:ss.sss J0Bnnnnn nnnnnnnn +CWWKT0016I: Web application available
                                         (default_host):
                                         870 nnnnnnnn http://hostname:port/
```

When the z/OS Connect EE server is stopped, the following messages are written to the z/OS console:

```
N 4000000 MVxx yyddd hh:mm:ss.sss J0Bnnnnn nnnnnnnn +CWWKB0001I: Stop command received for server
serverName.
N 4000000 MVxx yyddd hh:mm:ss.sss J0Bnnnnn nnnnnnnn +CWWKE1101I: Server quiesce complete.
```

Note: This function cannot be used to log the following shutdown messages, because they occur too late in the shutdown processing:

CWWKB0111I: IBM Corp product z/OS Connect version `vv.rr.mm` successfully deregistered from z/OS.
CWWKE0036I: The server `serverName` stopped after `m` minutes, `s.sss` seconds.

Using IBM Tivoli System Automation for z/OS with z/OS Connect EE

IBM Tivoli® System Automation for z/OS can be used to trap and respond to z/OS console messages issued by z/OS Connect EE servers to keep them highly available.

IBM Tivoli System Automation for z/OS V3.5 APAR [OA49939](#) supports a simple setup for automating and monitoring the Angel process and z/OS Connect EE servers. This setup uses a pre-defined z/OS Connect EE configuration that is provided by the *IBMCOMP add-on policy, similar to sample solutions for products like IMS, CICS, Db2 and others. For more information see the APAR problem description and the developerWorks® article [Using SA z/OS Sample Policies](#) in the *IBM Tivoli System Automation for z/OS, Version 3.5* documentation.

Note: The support added by APAR OA49939 does not automatically use the additional WTO messages, but you can use the instructions above to produce the WTO messages and manually include these in your policy.

With IBM System Automation for z/OS V3.5 APAR [OA51440](#), messages CWWKB0001I, CWWKF0011I, CWWKT0016I and CWWKE1101I are used to automate and monitor z/OS Connect EE. This automation also uses a pre-defined z/OS Connect EE configuration that is provided by the *IBMCOMP add-on policy.

The MODIFY command

Use the **MODIFY** command to refresh the z/OS Connect EE server artifacts.

To use the **MODIFY** command, add the following feature to the `server.xml` configuration file:

```
<feature>zosconnect:zosConnectCommands-1.0</feature>
```

You can use the **MODIFY** command to update Bind files, policies, .aar, .sar and .ara files, and the `server.xml` configuration file. When the **MODIFY** command is issued, the server is updated with changes to the `server.xml` configuration file and any new, modified, or deleted .aar, .sar, .ara, .dataXform, or policy files from the configured location directories.

Note:

1. To update the configuration file, the configuration must contain the statement `<config updateTrigger="mbean"/>`.
2. If an API or service is invoked while a refresh is in progress, interceptor calls might not occur as expected. For example, you change the `server.xml` file to add an interceptor, and issue the refresh command. While the refresh occurs, a service is invoked. The `preInvoke` interceptor method is not called because the refresh is not finished. After the refresh is complete, the `postInvoke` interceptor method for the service is called because it now exists.
3. When you use the **MODIFY** command to refresh the z/OS Connect EE server artifacts, the value of the `updateTrigger` attribute on the `zosconnect_apiRequesters`, `zosconnect_policy`, `zosconnect_services`, `zosconnect_zosConnectAPIs`, and `zosconnect_zosConnectDataXform` elements are ignored. Leave these values to default (`updateTrigger="disabled"`) to prevent changes from other update mechanisms.

The command has the following syntax.

```
modify <jobname>.<identifier>,help|zcon,refresh|cleartokencache|clearsafcache
```

For more information about the parameters, type `modify <jobname>.<identifier>, help` or see “[MODIFY command syntax](#)” on page 794 in the *Reference* section.

When you run the command with the `refresh` option, all the updated artifacts in z/OS Connect EE are reloaded. If the command runs successfully, the following message is returned.

Refresh command completed successfully

Specify the `cleartokencache` option to also clear all OAuth 2.0 access tokens and JWTs from the cache.

Specify the `clearsafcache` option to also clear the SAF cache.

If you request the help option, the supported options will be displayed.

To refresh keystores, including SAF key rings, Liberty provides the **MODIFY REFRESH,KEYSTORE** command. For more information, see [Modify commands on z/OS](#) in the *WebSphere Application Server for z/OS Liberty* documentation.

Related reference

Syntax of the modify command

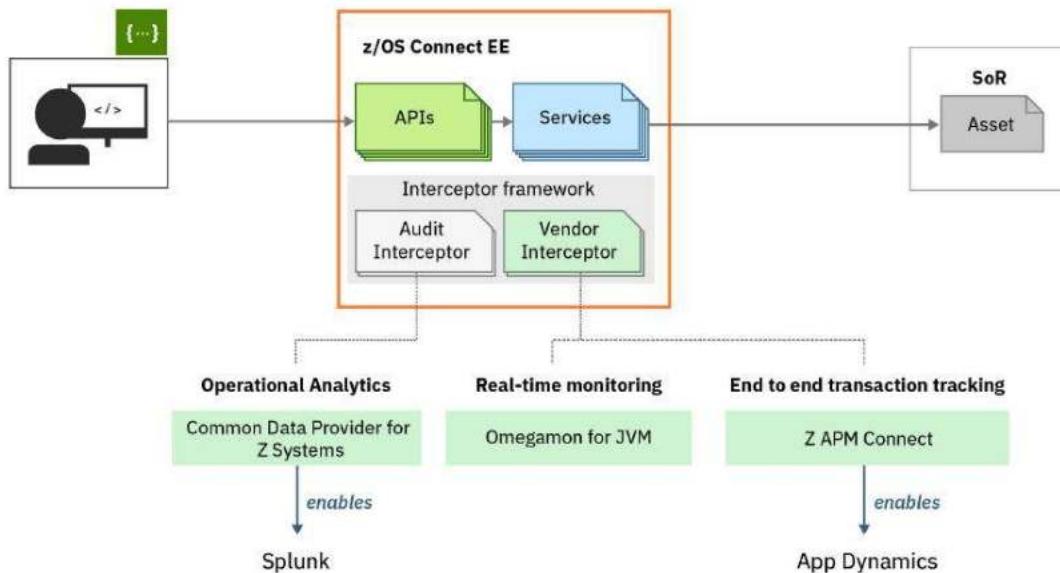
Use the **MODIFY** command to refresh the z/OS Connect EE server artifacts such as Bind files, policies, .aar, .sar, and .ara files, and the `server.xml` configuration file.

Chapter 11. Monitoring

z/OS Connect EE provides monitoring data through the interceptor framework that enables operational analysis, auditing and tracking of requests.

You can use one or more of the following methods to monitor z/OS Connect EE.

1. Record SMF data using the audit interceptor that can be integrated with common analytics platforms.
2. Monitor the health of APIs through real-time monitoring products such as OMEGAMON for JVM. For more information, see [Monitoring APIs with OMEGAMON for JVM](#) and [Using OMEGAMON for JVM to diagnose API failures](#).
3. Deploy end to end tracking of business transactions that flow through z/OS Connect EE to Z assets using tools like IBM Z APM Connect. For more information, see [Using AppDynamics with z/OS Connect EE](#).



Using SMF records to monitor requests

z/OS Connect EE can be configured to write SMF 123 records which capture information about individual requests. This data can be used for audit, to gain insights into your API workloads to aid capacity planning, and for problem determination.

There are two versions of the SMF type 123 subtype 1 records:

- Version 1 contains some basic information about requests.
- Version 2 supersedes version 1 and contains more detailed information about each request, including information about where the request was sent. The version 2 record contains data for multiple requests in one record, making it more efficient as the server information is captured once per record rather than for each request.

z/OS Connect EE provides the audit interceptor to capture SMF 123 records, see [“Configuring the audit interceptor” on page 291](#). By default the audit interceptor captures version 1 records.



CAUTION: SMF 123 subtype 1 version 2 records are not a compatible extension of the version 1 records. Therefore, before you select version 2 of the record, check that the tools you use to format SMF 123 records support the version 2 format, or formatting will fail.

Related information

[Abstract for MVS System Management Facility \(SMF\)](#)

API provider data from SMF type 123 subtype 1 version 2 records

The SMF type 123 subtype 1 version 2 record is used to capture enhanced data for individual API provider requests.

To configure z/OS Connect EE to capture SMF 123 records, see [“Configuring the audit interceptor” on page 291](#).

You should consider using SMF log streams with compression enabled. For more information, see [Setting up and managing SMF recording to logstreams](#) and the **COMPRESS** parameter in [Statements and parameters for SMFPRMxx](#).

SMF type 123 subtype 1 version 2 record structure

The SMF type 123 subtype 1 version 2 record contains the following sections:

1. Standard SMF header.
2. SMF type 123 subtype 1 header extension.
3. Server and Request data triplets section.
4. Server section (one instance).
5. Request data section (multiple instances).



Warning: Version 2 of the SMF 123 subtype 1 record has a different structure to the version 1 record after the 4-byte version field in the SMF type 123 subtype 1 header extension section.

SMF type 123 subtype 1 version 2 section descriptions

Standard SMF header

Table 40. Standard SMF 123 subtype 1 header fields.

Offset	Name	Length	Data type	Data
0	SMF123_LEN	2	BINARY	Record length. Maximum size is 32,756.
2	SMF123_SEG	2	BINARY	Segment descriptor.
4	SMF123_FLAG	1	HEX	System indicator (0x01011110).
5	SMF123_REC_TYPE	1	BINARY	Record type (123 -0x7B).
6	SMF123_TIME	4	TIME (1/100s)	Time when record was moved into the SMF buffer, in hundredths of a second since midnight.
10	SMF123_DATE	4	DATE (01YYDDDF)	Date when record was moved into the SMF buffer, YY is the current year (0 -99), DDD is the current day (1 -366), and F is the sign.

Table 40. Standard SMF 123 subtype 1 header fields. (continued)

Offset	Name	Length	Data type	Data
14	SMF123_SID	4	CHAR	System ID from the SID parameter.
18	SMF123_SSI	4	CHAR	Subsystem ID, set SUBSYS =option specified in the SMF macros. The default is "ZCON".
22	SMF123_SUBTYPE	2	BINARY	Record subtype (0x0001).

Note: The first 4 bytes of the standard SMF header, the record descriptor word (RDW), contains the fields SMFxLEN and SMFxSEG. The SMF offload utility, IFASMFDP, strips the RDW from each record, so offsets for the subsequent header fields must be adjusted.

SMF type 123 subtype 1 version 2 header extension

Table 41. SMF 123 subtype 1 header extension.

Offset	Name	Length	Data type	Data
0	SMF123S1_SUBTYP_E_VERSION	4	BINARY	Record subtype version. Set to 2.
4	SMF123S1_TRIPLET_COUNT	1	BINARY	Number of triplets in this record. Default is x0002.
5	SMF123S1_TRIPLET_OFFSET	1	BINARY	Triplet section offset.
6	SMF123S1_RESERVED_01	2	-	Reserved.
8	SMF123S1_DATETIME_OFFSET	8	BINARY	Local date/time offset (CVTLDTO).

Triplet section

Table 42. Triplet section

Offset	Name	Length	Data type	Data
0	SMF123S1_SERVER_OFFSET	4	BINARY	Server ID section offset.
4	SMF123S1_SERVER_LEN	2	BINARY	Server ID section length.
6	SMF123S1_SERVER_COUNT	2	BINARY	Server ID section count.
8	SMF123S1_REQ_DATA_OFFSET	4	BINARY	Request data section offset.
12	SMF123S1_REQ_DATA_LEN	2	BINARY	Request data section length.
14	SMF123S1_REQ_DATA_COUNT	2	BINARY	Request data section count.

Server section

Table 43. Server section

Offset	Name	Length	Data type	Data
0	SMF123_SERVER_SECTION_VERSION	1	BINARY	Version of the server section.
1	SMF123_RESERVED_02	3	-	Reserved.
4	SMF123_SERVER_SYSTEM	8	CHAR	System name (CVTSNAME).
12	SMF123_SERVER_SYSPLEX	8	CHAR	Sysplex name (ECVTSPLX).
20	SMF123_SERVER_JOBID	8	CHAR	Job ID of the server (JSABJBID).
28	SMF123_SERVER_JOBNAME	8	CHAR	Job name of the server (JSABJBNM).
36	SMF123_SERVER_STOKEN	8	BINARY (HEX)	SToken of the server (ASS-BSTKN).
44	SMF123_SERVER_CONFIG_DIR	128	CHAR	The path to where server.xml for the server is located, this includes the server name.
172	SMF123_SERVER_VERSION	16	CHAR	Version of the server <v.r.m.f>.

Request data section

Table 44. Request data section

Offset	Name	Length	Data type	Data
0	SMF123_REQ_DATA_VERSION	1	BINARY	Version of request data record. Set to 1.
1	SMF123S1_REQ_TYPE	1	BINARY	Request type: 0 = Unknown 1 = API 2 = Service 3 = Admin
2	SMF123S1_HTTP_RESPONSE_CODE	2	BINARY	HTTP response code.
4	SMF123S1_RESP_FLAGS	1	BINARY	Response flags.
	(BIT 0): SMF123S1_REQ_TIMED_OUT		BIT	Request timed out indicator.
	(BITS 1- 7): SMF123S1_RESERVED_03		-	Reserved.

Table 44. Request data section (continued)

Offset	Name	Length	Data type	Data
5	SMF123S1_RESERVED_04	3	-	Reserved.
8	SMF123S1_USER_NAME	64	CHAR	<p>One of the following:</p> <ul style="list-style-type: none"> • If TLS client certificate authentication was used, the authenticated username associated with that certificate. • The distributed ID, if one was sent on the request and is mapped to a SAF username. • The authenticated username.
72	SMF123S1_USER_NAME_MAPPED	8	CHAR	If a distributed ID was sent on the request and is mapped to a SAF username, then this is the authenticated SAF username. Otherwise this value is blank.
80	SMF123S1_CLIENT_IP_ADDR	48	CHAR	Client IP address.
128	SMF123S1_API_NAME	64	CHAR	API name.
192	SMF123S1_API_VERSION	8	CHAR	API version.
200	SMF123S1_SERVICE_NAME	64	CHAR	Service name.
264	SMF123S1_SERVICE_VERSION	8	CHAR	Service version.
272	SMF123S1_REQ_METHOD	8	CHAR	Method GET/POST/PUT/DELETE.
280	SMF123S1_REQ_QUERY_STR	128	CHAR	Query string.
408	SMF123S1_REQ_TARGET_URI	256	CHAR	Target URI.
664	SMF123S1_REQ_PAYLOAD_LEN	4	BINARY	Request payload length in bytes.
668	SMF123S1_RESP_PAYLOAD_LEN	4	BINARY	Response payload length in bytes.
672	SMF123S1_TIME_ZC_ENTRY	16	STCKE	Time the request was received by the z/OS Connect EE server.
688	SMF123S1_TIME_ZC_EXIT	16	STCKE	Time the response was ready to be sent to the HTTP client.
704	SMF123S1_TIME_SOURCE_SENT	16	STCKE	Time the request was sent to the system of record.

Table 44. Request data section (continued)

Offset	Name	Length	Data type	Data
720	SMF123S1_TIME_S OR_RECV	16	STCKE	Time the response was received from the system of record.
736	SMF123S1_SP_NAM E	16	CHAR	Service provider name. Value of com.ibm.zosconnect.spi.Data.SERVICE_PROVIDER_NAME .
752	SMF123S1_SOR_REF ERENCE	32	CHAR	Reference to the element in server.xml that identifies the connection to the system of record. Value of com.ibm.zosconnect.spi.Data.SOR_REFERENCE .
784	SMF123S1_SOR_ID ENTIFIER	64	CHAR	System of record identifier. Value of com.ibm.zosconnect.spi.Data.SOR_IDENTIFIER .
848	SMF123S1_SOR_RE SOURCE	128	CHAR	System of record resource. Value of com.ibm.zosconnect.spi.Data.SOR_RESOURCE .
976	SMF123S1_REQ_ID	8	BINARY	Request identifier that is unique within a z/OS Connect EE server instance.
984	SMF123S1_TRACKI NG_TOKEN	64	BINARY (HEX)	Tracking token. For more information, see “Tracking selected requests” on page 710 .
1048	SMF123S1_REQ_HD R1	64	CHAR	Request header. <code><header1name>:<header1value></code>
1112	SMF123S1_REQ_HD R2	64	CHAR	Request header. <code><header2name>:<header2value></code>
1176	SMF123S1_REQ_HD R3	64	CHAR	Request header. <code><header3name>:<header3value></code>
1240	SMF123S1_REQ_HD R4	64	CHAR	Request header. <code><header4name>:<header4value></code>
1304	SMF123S1_RESP_H DR1	64	CHAR	Mapped response header. <code><header1name>:<header1value></code>

Table 44. Request data section (continued)

Offset	Name	Length	Data type	Data
1368	SMF123S1_RESP_H_DR2	64	CHAR	Mapped response header. <header2name>:<header2value>
1432	SMF123S1_RESP_H_DR3	64	CHAR	Mapped response header. <header3name>:<header3value>
1496	SMF123S1_RESP_H_DR4	64	CHAR	Mapped response header. <header4name>:<header4value>

Sample JCL to format SMF 123 subtype 1 records

To format the SMF 123 subtype 1 records, use the sample JCL provided. For more information, see “[Sample JCL to format SMF records](#)” on page 484.

Request data from SMF type 123 subtype 1 version 1 records

The SMF type 123 subtype 1 version 1 record is used to capture some basic data for individual requests.

You can change your configuration to use the SMF type 123 subtype 1 version 2 records to benefit from the more enhanced data that they provide for API provider requests. See “[API provider data from SMF type 123 subtype 1 version 2 records](#)” on page 476.

To configure z/OS Connect EE to capture SMF 123 records, see “[Configuring the audit interceptor](#)” on page 291.

SMF type 123 subtype 1 version 1 record structure

The SMF type 123 subtype 1 version 1 record contains the following sections:

1. Standard SMF header.
2. SMF type 123 subtype 1 header extension.
3. Server and User data triplets section.
4. Server section (one instance).
5. User data section (one instance).

SMF type 123 subtype 1 version 1 section descriptions

Standard SMF header

Table 45. Standard SMF 123 subtype 1 header fields.

Offset	Name	Length	Data type	Data
0	SMF123_LEN	2	BINARY	Record length. Maximum size is 32,756.
2	SMF123_SEG	2	BINARY	Segment descriptor.
4	SMF123_FLAG	1	HEX	System indicator (0x01011110).
5	SMF123_REC_TYPE	1	BINARY	Record type (123 -0x7B).

Table 45. Standard SMF 123 subtype 1 header fields. (continued)

Offset	Name	Length	Data type	Data
6	SMF123_TIME	4	TIME (1/100s)	Time when record was moved into the SMF buffer, in hundredths of a second since midnight.
10	SMF123_DATE	4	DATE (01YYDDDF)	Date when record was moved into the SMF buffer, YY is the current year (0 -99), DDD is the current day (1 -366), and F is the sign.
14	SMF123_SID	4	CHAR	System ID from the SID parameter.
18	SMF123_SSI	4	CHAR	Subsystem ID, set SUBSYS =option specified in the SMF macros. The default is "ZCON".
22	SMF123_SUBTYPE	2	BINARY	Record subtype (0x0001).

Note: The first 4 bytes of the standard SMF header, the record descriptor word (RDW), contains the fields SMFxLEN and SMFxSEG. The SMF offload utility, IFASMFDP, strips the RDW from each record, so offsets for the subsequent header fields must be adjusted.

SMF type 123 subtype 1 version 1 header extension

Table 46. SMF 123 subtype 1 header extension fields.

Offset	Name	Length	Data type	Data
0	SUBTYPE_VERSION	4	BINARY	Record subtype version. Set to 1.
4	TRIPLET_COUNT	4	BINARY	Number of triplets. Defaults to x0002.
8	RECORD_INDEX	4	BINARY	Index of this record. Always 0.
12	RECORD_COUNT	4	BINARY	Total number of records. Always 1.
16	RECORD_CONT	8	HEX	Record continuation token. Always 8 bytes of nulls.

Server and User data triplets section

Table 47. Server and User Data triplets section fields.

Offset	Name	Length	Data type	Data
0	SERVER_OFFSET	4	BINARY	Server section offset
4	SERVER_LEN	4	CHAR	Server section length.
8	SERVER_COUNT	4	BINARY	Server section count.
12	USERDATA_OFFSET	4	BINARY	User Data section offset.
16	USERDATA_LEN	4	BINARY	User Data section length.
20	USERDATA_COUNT	4	BINARY	User Data section count.

Server section

Table 48. Fields in version 1 of the Server data section.

Offset	Name	Length	Data type	Data
0	SERVER_SECTION_VER	4	BINARY	Version of the server section.
4	SERVER_SYSTEM	8	CHAR	System name.
12	SERVER_SYSPLEX	8	CHAR	Sysplex name.
20	SERVER_JOBID	8	CHAR	Job ID of the server.
28	SERVER_JOBNAME	8	CHAR	Job name of the server.
36	SERVER_STOKEN	8	BINARY	SToken of the server.
44	RESERVED_01	36	CHAR	Reserved.

User data section

Table 49. Fields in version 1 of the User data section header.

Offset	Name	Length	Data type	Data
0	USERDATAHD_VER	4	BINARY	Version of user data header. This table maps version 1 of user data.
4	USERDATAHD_TYPE	4	BINARY	User data type. The value is 0x66.
8	USERDATAHD_DATALEN	4	BINARY	Length of following user data.
12				Data (See Table 50 on page 483).

Table 50. Fields in version 3 of the User data section body..

Character data is EBCDIC right-padded with blanks.

Offset	Name	Length	Data type	Data
0	USERDATA_VER	4	BINARY	Version of user data record. This table maps version 3 of user data.
4	TIME_ZC_ENTRY	8	STCK	Arrival date and time.
12	TIME_ZC_EXIT	8	STCK	Completion date and time.
20	REQ_TARGET_URI	64		Target URI.
84	REQ_PAYLOAD_LEN	4		Request payload length in bytes.
88	API_SERVICE_NAME	64	CHAR	API or service name.
152	REQ_METHOD	8	CHAR	Method (GET/POST/PUT/DELETE)

Table 50. Fields in version 3 of the User data section body..

Character data is EBCDIC right-padded with blanks.

(continued)

Offset	Name	Length	Data type	Data
160	RESP_PAYLOAD_LEN	4	BINARY	Response payload length in bytes
164	USER_NAME	64	CHAR	One of the following: <ul style="list-style-type: none">• If TLS client certificate authentication was used, the authenticated username associated with that certificate.• The distributed ID, if one was sent on the request and is mapped to a SAF username.• The authenticated username.
228	REQ_ID	8	HEX	Request ID (simple sequence number).
236	RESERVED_02	15	HEX	Unused portion of request ID.
251	RESERVED_03	1	BINARY	Reserved for alignment.
252	SERVICE_GROUP	64	CHAR	Service grouping (unused).
316	USER_NAME_MAPPED	8	CHAR	If a distributed ID was sent on the request and is mapped to a SAF username, then this is the authenticated SAF username. Otherwise this value is blank.

Sample JCL to format SMF 123 subtype 1 records

To format the SMF 123 subtype 1 records, use the sample JCL provided. For more information, see “[Sample JCL to format SMF records](#)” on page 484.

Sample JCL to format SMF records

z/OS Connect EE provides sample programs for formatting SMF 123 subtype 1 records. Choose the sample to use based on the version of the records you are formatting.

Note: To use any of these samples you must first edit the JCL to replace the necessary values by following the instructions in the sample.

Formatting SMF 123 subtype 1 version 1 records

To format SMF 123 subtype 1 version 1 records, dump your SMF data set to a raw file and use the sample JCL provided in <hlq>.SBAQSAMP(BAQS123). The JCL uses the ICETOOL utility and formats the user data section.

Formatting SMF 123 subtype 1 version 2 records

To format SMF 123 subtype 1 version 2 records, use either of these samples to dump your SMF data to a file and format the records:

- <hlq>.SBAQSAMP(BAQS123A) uses the ICETOOL utility but only formats the first request data section in the record. The BAQS123A sample is only suitable when all SMF 123 type 1 records in the SMF data sets are at version 2. If any SMF 123 type 1 version 1 records are found, the sample job will fail with the following message:

```
ICE218A 3 1188 BYTE VARIABLE RECORD IS SHORTER THAN 1804 BYTE MINIMUM FOR FIELDS
```

This is a limitation of the z/OS ICETOOL utility that is used by the sample program.

- <hlq>.SBAQSAMP(BAQSMFP) uses a sample C program and formats all the request data sections in an entire version 2 record.

Chapter 12. Exposing z/OS assets as REST APIs

To access and act on a resource on a z/OS subsystem through REST APIs, create a service that defines how the JSON schemas for the request and response messages map to the resource. Then, design a REST API to define how an HTTP action such as GET, PUT, POST, or DELETE would act on the service.

Depending on the service provider and the z/OS subsystem you are accessing, create z/OS Connect EE services and the related service archive (.sar) files by using the z/OS Connect EE API toolkit, the build toolkit, or other required utilities. The z/OS Connect EE API toolkit also provides a user interface for designing and creating REST APIs for your services.

- The API toolkit is an Eclipse-based tool that provides a graphical interface for defining a CICS, IMS or IBM MQ service, with powerful data mapping and service redaction support. Db2 services can also be created using the API toolkit through the REST client service provider.
- The build toolkit provides a command-line interface and also an SDK to build your service archive file.
 - For service providers that are supported by the API toolkit, the build toolkit can build the service archive file from the service project directory created by the API toolkit.
 - For service providers that are not supported by the API toolkit, the build toolkit builds the service archive file from a properties file that was created for the CICS and IBM MQ service providers.

Regardless of how your service is created and defined, the build toolkit enables scripting and an automated process for continuous integration and delivery of services.

In the z/OS Connect EE API toolkit (the **z/OS Connect Enterprise Edition** perspective in the Eclipse environment), you can do the following work:

- Create and export a service project based on CICS channels and containers, CICS COMMAREA, or IMS messages and segments or IBM MQ messages. A service interface editor is provided for you to define the request and response service interfaces, specifying which fields can be exposed, and how they are exposed.
- Create, edit, delete, deploy, and export an API project.
- Create a connection to a z/OS Connect EE server in the **Host Connections** view.
- Browse, start, stop, and remove deployed APIs on connected servers in the **z/OS Connect EE Servers** view.
- Examine and test the operations of an API in Swagger UI that is included in the API toolkit.

Creating services

Use the z/OS Connect EE API toolkit to create your service from CICS, IMS, or IBM MQ through their respective service provider.

Db2 services can be created through the REST client service provider.

For other service providers, prepare a properties file that the build toolkit needs to generate the service archive.

A service archive (.sar) file contains the information that is needed by a z/OS Connect EE service provider to install and provide the service, and to enable the service as a JSON asset. A service archive is also required to build APIs in the z/OS Connect EE API toolkit.

Although you can build a CICS service archive by using the build toolkit, the API toolkit supports payload data structures customization and COBOL REDEFINES handling. These advanced functions are not available if you use the build toolkit. Use the build toolkit command line to create your service in the following scenarios:

- Your program is written in C or C++.
- You are creating a service from an existing JSON schema that defines the external interface.

- You are migrating CICS services that use WOLA to use IPIC and want the JSON format to remain unchanged.

In these cases, see “[Creating a CICS service for C or top-down](#)” on page 798 or “[Creating a WOLA service](#)” on page 556.

Although you can build an IBM MQ service archive from a properties file by using the build toolkit, the API toolkit provides support for payload sizes greater than 32 KB, payload data structure customization and COBOL REDEFINES handling. These advanced functions are not available if you use the build toolkit.

For more information, see “[Creating an IBM MQ service by using a properties file](#)” on page 813.

Creating a CICS, IMS or IBM MQ service

You can create a service for CICS, IMS, or IBM MQ by using the z/OS Connect EE API toolkit.

The z/OS Connect EE API toolkit helps you create a service, define the request and response service interfaces, and specify how the service can access and interact with the z/OS subsystem.

A *service interface* defines how the underlying program, COBOL copybook, or PL/I include can be accessed, which fields can be exposed, and how they are exposed. A service interface can be reused by multiple services. Service interfaces are stored as service interface files (.si files).

Note: If the source of your CICS service is not COBOL or PL/I, see “[Creating a CICS service for C or top-down](#)” on page 798.

After a service is created, you can deploy it directly from the z/OS Connect EE API toolkit. The **zosconnect_services** element must be present in the `server.xml` configuration file for service deployment from the API toolkit.

Note:

Deploying a service directly from within the API toolkit requires server code V3.0.4 or later.

You can also export the service project as a service archive (.sar) file. For more information about deploying the generated service archive file, see “[Automated service archive management](#)” on page 670.

For step-by-step examples of creating services with the API toolkit, see:

- “[Create a CICS service](#)” on page 83
- “[Create an IMS service](#)” on page 89
- “[Create a two-way IBM MQ service](#)” on page 96

Restrictions on the data structure import function

Depending on the z/OS subsystem and types of programs, restrictions on the size and number of data structures apply. The data structure import function is supported on specific platforms.

Supported platforms for the data structure import function

The data structure import function in the API toolkit is supported on the following platforms:

- For COBOL, Windows and Linux platforms are supported.

The COBOL data structure import function requires the 32-bit version of the OS system libraries. These libraries are not included in some Linux distributions. When these 32-bit libraries are missing, you get an error message that states the import wizard has not found any valid structure declarations. The Eclipse error log includes the following entry:

```
Cannot run program "/opt/IBM/zOS_Explorer/../IBMMIMShared/plugins/
com.ibm.etools.cobol.linx_7.1.200.201706022205/...."
```

To resolve this issue, see [Features that use the COBOL importer do not work properly in the IBM Developer for z Systems documentation](#).

- For PL/I, the Windows platform is supported.

Supported data structures

The following restrictions apply to CICS:

- COMMAREAs are limited to 32 KB.
- For channels, you can have any number of containers in a channel and each container can be up to 2 GB.

The following data structure restrictions apply to IMS:

- For IMS programs with data structures whose maximum size exceeds 32 KB (32,767 bytes), the programs must handle the decomposition of a large data structure into multiple segments and assembly of multiple segments back into the large data structure. For a sample on how to handle COBOL data structures larger than 32 KB, see [“Handling large IMS data structures” on page 540](#). When you create a service project in the API toolkit, choose **IMS Large Data Structure Service** as the service type.

Note:

For the maximum size of Enterprise COBOL and PL/I data structures, see the respective documentation:

- [Compiler limits for Enterprise COBOL for z/OS 6.1](#)
- [Limits for Enterprise PL/I for z/OS 5.1](#)

The actual allowable data structure size depends on how your IMS subsystem is configured and tuned.

- The maximum number of data structures is 256.
- Variable count of segment structures is not supported. Each segment must be explicitly added.

The following data structure restrictions apply to IBM MQ:

- – Both request and response data structures must be able to fit inside an IBM MQ message, so they must be less than or equal to 100 MB.

For more information about how COBOL and PL/I fields are converted to JSON fields and syntax restrictions, see the following topics:

- [“COBOL to JSON schema mapping” on page 531](#)
- [“PL/I to JSON schema mapping” on page 535](#)

Resolving COBOL importer issues

Before you import your COBOL file, ensure that all errors reported by the IBM Enterprise COBOL compiler are resolved. Use the **Problems** view or the log file to determine the cause of the issues.

Common issues

If the COBOL source has data names in double-byte character set (DBCS), ensure the compile-time locale is set to one that supports DBCS data. To set the locale, click **Window > Preferences > Importer > COBOL**, and click the **More COBOL options** tab to select a locale such as **jp_JP** for the **Compile time locale name** field.

Line delimiters or encoding might be accidentally modified when files are transferred and opened on different platforms. The importer could report an IGYDS0027-S error when it cannot recognize the carriage return character. An IGYDS0917-E error might indicate that the file name or lines in the file are longer than 72 characters.

For more information about how COBOL fields are converted to JSON fields and syntax restrictions, see [“COBOL to JSON schema mapping” on page 531](#).

The following workspace log message can be ignored: This message is caused by importing a COBOL copybook that does not have a record level (01 level field). The import logic handles this for you and the copybook is imported correctly.

For Ubuntu Linux (64-bit) systems, dependency packages must be installed by using a single **install** command. If you are experiencing issues where the COBOL importer does not work on Ubuntu Linux (64-bit) systems, use the following procedure to ensure dependencies are properly installed:

1. Close z/OS Explorer if it is not already closed.
2. Issue the following commands:

- a. `sudo dpkg --add-architecture i386`
- b. `sudo apt-get update`
- c. `sudo apt-get install lsb-core libc6:i386 libstdc++6:i386`

3. Reopen z/OS Explorer.

Resolving issues

When you import a COBOL file from your workspace, syntax error messages show up in the task list. Click each error message in the **Problems** view to view the source of the error in your code. The error messages are also logged in a generic log file, which is located under the metadata directory at <workspace>\.metadata\.log

If the imported file is not in your workspace, the error messages are logged in the log file.

Related tasks

[“Importing data structures or full programs” on page 494](#)

You can import one or more data structure files (COBOL copybooks or PL/I includes) into your service project. You can also import one or more full COBOL or PL/I programs.

Related reference

[“COBOL to JSON schema mapping” on page 531](#)

The data transformation function in the z/OS Connect EE API toolkit converts COBOL fields to JSON fields.

Related information

[Resolving COBOL Importer Errors \(IBM Rational Application Developer for WebSphere Software\)](#)

Resolving PL/I importer issues

Errors reported by the Enterprise PL/I compiler, if not resolved before you import the PL/I file, would be reported by the PL/I importer. Use the **Problems** view or the log file to determine the cause of the issues.

Common issues

The maximum length for WIDECHAR and CHAR strings supported by PL/I is 32,767. Elements that are longer than 32,767 characters result in an IBM1299IE message, and the PL/I data structure cannot be loaded into the service interface editor.

For more information about how PL/I fields are converted to JSON fields and syntax restrictions, see [“PL/I to JSON schema mapping” on page 535](#).

Resolving issues

When you import a PL/I file from your workspace, syntax error messages show up in the **Problems** view. Click each error message in the **Problems** view to examine the source of the error in your code. Errors are also logged in a generic log file, which is located under the metadata directory at <workspace>\.metadata\.log

If the imported file is not in your workspace, the error messages show up in the error log.

Related tasks

[“Importing data structures or full programs” on page 494](#)

You can import one or more data structure files (COBOL copybooks or PL/I includes) into your service project. You can also import one or more full COBOL or PL/I programs.

Related reference

[“PL/I to JSON schema mapping” on page 535](#)

The data transformation function in the z/OS Connect EE API toolkit converts PL/I fields to JSON fields.

Related information

[Resolving PL/I Importer Errors \(IBM Rational Application Developer for WebSphere Software\)](#)

Creating a service

Create a z/OS Connect EE service project in the **z/OS Connect Enterprise Edition** perspective and define the request and response service interfaces.

Before you begin

Switch to the **z/OS Connect Enterprise Edition** perspective in your Eclipse environment.

1. From the main menu, select **Window > Open Perspective > Other**. The Select Perspective wizard opens.
2. Select **z/OS Connect Enterprise Edition**.

About this task

The project types that are described in this task are supported by the CICS, IMS, or IBM MQ service providers. If you use the WOLA service provider, you must use the z/OS Connect EE build toolkit to create your service. For more information, see [“Creating a WOLA service” on page 556](#).

Procedure

1. Select **File > New > Project**.
The New Project wizard opens.
2. Select **z/OS Connect Enterprise Edition > z/OS Connect EE Service Project**, and click **Next**.
3. Specify a project name, select the project type, and optionally provide a description.
The following project types are supported:
 - **CICS COMMAREA Service**
 - **CICS Channel Service**
 - **IMS Service**
 - **IMS Large Data Structure Service** (for IMS programs with data structures that exceed the maximum size of 32 KB). For large IMS data structure, the imported structure must not begin with an LLZZ or LLZZ<TRANCODE> prefix.
 - **IBM MQ Two-Way Service**
 - **IBM MQ One-Way Service for Sending Messages**
 - **IBM MQ One-Way Service for Receiving Messages**
4. Click **Finish** to create the project.

The service project is created in the **Project Explorer** view. The `service.properties` file opens in the service project editor in a tab that is named after the service project. This service project editor is where you can configure the service and define the service interface. Initially, errors (X) are reported and highlighted for information that is required and must be specified. Depending on the service type, required information varies.

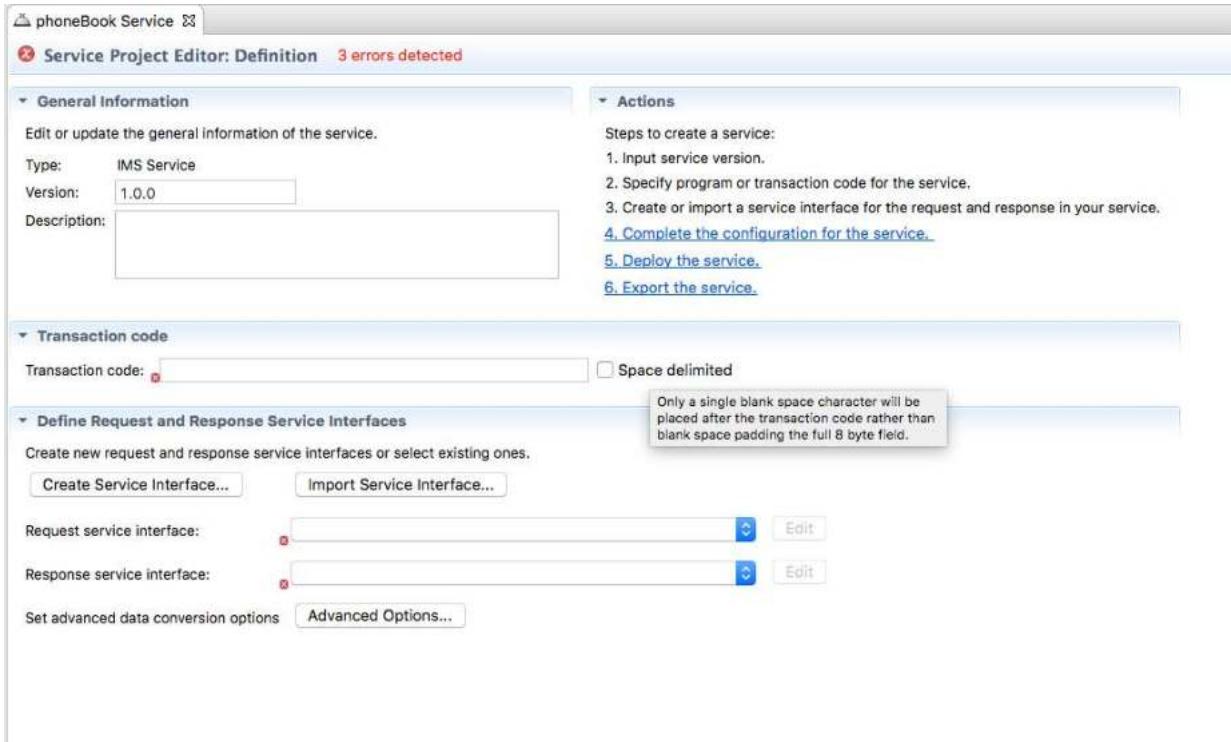


Figure 101. The service project editor

This image is for an IMS service. A CICS or IBM MQ service might have different fields.

The **Actions** pane highlights the steps to create a service.

5. In the service project editor, take the following steps:

- Optionally, change the version number from the default of 1.0.0.
- Specify the program (for CICS) or transaction code (for IMS) to invoke.

The IMS transaction code that is specified here can be overridden when you define the service interface. It can also be overridden at run time. For more information, see “[Transaction code override precedence](#)” on page 256.

Tip:

For IMS services, a portion of the transaction code field can be used to provide more data to the service. If you want to repurpose any unused bytes in the transaction code area of the buffer, toggle the **Space delimited** option. By default, transaction codes are padded to a full 8-byte length.

- Define the request and response service interfaces.

The service interfaces are defined by importing COBOL copybooks, PL/I include files, or full programs, and customizing the interfaces. For more information, see “[Defining the request and response service interfaces](#)” on page 493.

- Select the service interface file to use for the **Request service interface** field.
- Select the service interface file to use for the **Response service interface** field.
- Optionally, specify initialization, conversion, and omission options for your data by clicking **Advanced Options**.
For more information, see “[Service-level data conversion customization](#)” on page 537.
- Click the **Configuration** tab to configure subsystem-specific service properties. For more information, see “[Configuring service properties](#)” on page 510.
- Save your changes.

Note: For IBM MQ one-way services, there is just a single **Service interface** field instead of the **Request service interface** and **Response service interface** fields.

Results

After validation that all required information is specified, the JSON schema files and the service XML file are created in the service project folder. To later edit the service project, open the service project editor by double-clicking the service .properties file in the project folder within the **Project Explorer** view.

What to do next

[“Deploying a service” on page 529.](#)

Defining the request and response service interfaces

Define the service interface from one or more COBOL copybooks, PL/I includes files, or a full program. A *service interface* defines the fields to expose for request and response messages, and how these fields are made available to the API developers, such as with a more descriptive field name.

Procedure

You can create a new service interface by importing a COBOL copybook, PL/I includes file, or full program. You can also import an existing service interface from another service project or the file system. An imported service interface file can be further modified for this service project.

Important: Response service interfaces cannot contain default values in fields. If a default value is specified for a response service interface, an error is issued in the Problems view to assist problem resolution.

1. Create a service interface or reuse an existing service interface in another service project.

- To create a new service interface:
 - a. Click **Create Service Interface**.
 - b. Specify a name for the service interface.
 - c. Click **OK**. The service interface editor opens for you to create the service interface.
 - d. In the service interface editor, import the data structure or full program by clicking **Import data structure** (import icon), or depending on the service type, right-click the COMMAREA, container, segment, or IBM MQ message, and select **Import data structure**.

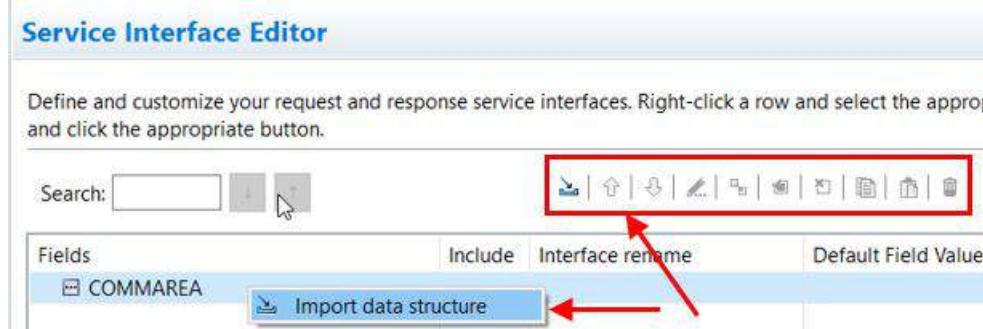


Figure 102. The service interface editor

For more information, see [“Importing data structures or full programs” on page 494.](#)

- To reuse an existing service interface from another project or the file system:
 - a. Click **Import Service Interface**.
 - b. Select an existing service interface file (.si file) from another project in your workspace or from your file system, and click **OK**. Imported service interface files can later be selected as the request or response service interface for the current project.

Note: When you save the service interface, a copy of the service interface file of the same name is saved in this service project. Although the new copy has the same file name as the original one, they are separate copies. The original copy remains intact.

2. Optionally, customize the service interface. You can copy a data structure, specify the order of the data structures, or add multiple containers or segments. You can also specify the fields to include or exclude, rename fields or data structures, change field value type, specify default field values, or add business descriptions.
For more information, see “[Customizing the service interface](#)” on page 496.
3. Save your changes.

Results

The service interface file (.si file) is saved in the service-interfaces/ folder in your service project.

Importing data structures or full programs

You can import one or more data structure files (COBOL copybooks or PL/I includes) into your service project. You can also import one or more full COBOL or PL/I programs.

Before you begin

- Ensure that the program compiles without errors. Errors reported by IBM Enterprise COBOL compiler or Enterprise PL/I compiler must be resolved before you import the file.
- Read the “[Restrictions on the data structure import function](#)” on page 488 topic.

Procedure

To import data structures:

1. In the service interface editor, click **Import data structure**.
 - If you select a COMMAREA, container, segment, IBM MQ message, or any of the rows below it before you click **Import data structure**, the wizard automatically populates the specified COMMAREA, container, segment, or IBM MQ message as the target.
 - If the data structure has changed for an existing request or response service interface, you can re-import the data structure file by clicking **Import data structure**. Existing custom data structure names (aliases) and remarks are preserved if the 01 level structure remains unchanged.
2. Specify the location to import the data structures or a full program from, and specify the file type.
3. Click **Browse** to select the data structure file that contains the COBOL copybooks or PL/I includes to import.

Tip: If you are importing an application source file that references a COPY or INCLUDE file, import from a local file system with both the application source file and the referenced file in the same directory.

- If you import the data structures from z/OS, the data structure file does not have a file extension. It takes the file type that is specified in “[2](#)” on page 494.
- If you import the data structures from a local file system, a file with one of the following extensions takes that file type as default:
 - A COBOL copybook can have any of the following file name extensions: .cbl, .ccp, .cpy, or .cob.
 - A PL/I include can have any of the following file name extensions: .pli, .inc, or .mac.

If the file extension is incorrect for the type of file type that the file is, either change the file extension or click **Window > Preferences > Importer**, and then select either COBOL or PL/I from the list on the left. Then, edit the type of file that is indicated by each file extension.

If the data structure file does not have a file extension, it takes the file type that is specified in “[2](#)” on page 494.

4. Select the data structure name and the associated container or segment to import the data structure into.
5. Click **Add to Import List**.

The selected data structure is added to data structure table.

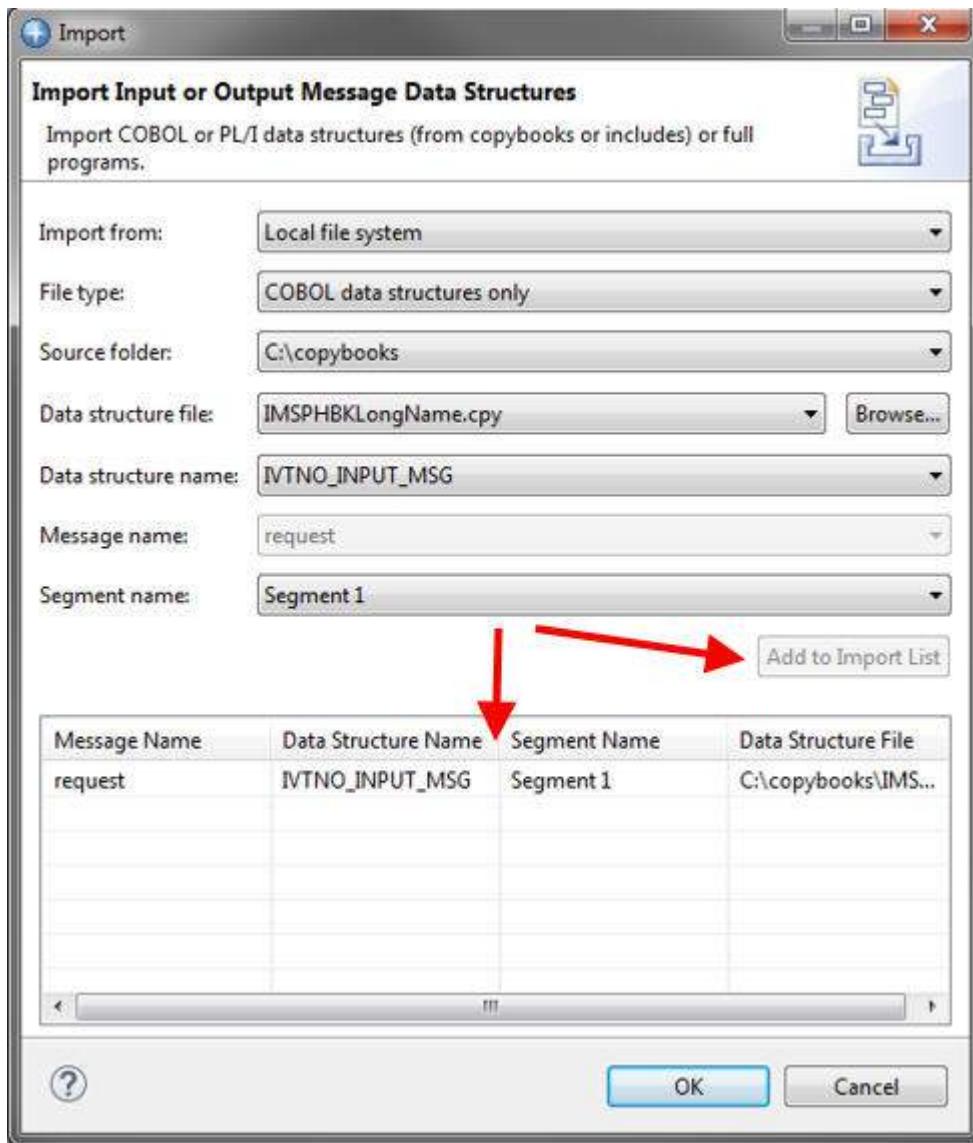


Figure 103. Import dialog for an IMS service

6. Repeat steps “3” on page 494 through “5” on page 494 for each data structure that you want to import.
7. Click **Finish** to import the data structure.

If you are updating an existing data structure, a dialog opens. Click **Yes** to replace the existing data structure and preserve your aliases and remarks. Click **No** to add the data structure as a new, separate data structure.

Tip:

- For each additional container or segment, select a channel or message node and click **Add**. Repeat the steps from step 3 for each data structure that you want to import. For more information and restrictions, see “Customizing the service interface” on page 496.
- When importing multiple copybooks, the start byte for each copybook is relative to the root data structure. In many cases, the value shown is 1. If your data structure utilizes OCCURS DEPENDING ON or REFER clauses, no start positions are indicated for start byte fields.

For more tips about potential issues and how to resolve them, see the following topics:

- “Resolving COBOL importer issues” on page 489
- “Resolving PL/I importer issues” on page 490

Related information

[Setting COBOL Importer Preferences \(IBM Rational Application Developer\)](#)

[Setting PL/I Importer Preferences \(IBM Rational Application Developer\)](#)

Customizing the service interface

You can customize the payload data structures that comprise the request and response service interfaces by copying data structures from one location into another, move data structures up or down, or removing data structures. Service interface fields can be further customized by using the **Edit field** wizard to specify service field names, default values, descriptions, and more.

Before you begin

If the service interface file is not already opened in the service interface editor, in Project Explorer, go to the **service-interfaces/** folder for your project and double-click the service interface file (.si file).

About this task

The following example is a data structure:

```
01 INPUT-MSG.  
  02 IN-LL    PICTURE S9(3) COMP.  
  02 IN-ZZ    PICTURE S9(3) COMP.  
  02 IN-TRCD  PICTURE X(10).  
  02 IN-DATA  PICTURE X(8).
```

You can customize your service interfaces as follows:

- Copy a data structure from a container, COMMAREA, segment, or IBM MQ message by right-clicking the data structure and selecting **Copy data structure**. Then, select either the target data structure folder or the data structure of another that you want to paste the copied data structure after, right-click, and select **Paste data structure**.
 - Import another data structure.
- Tip:** If you create a data structure (by importing or pasting) that has the same name as an existing data structure, it is automatically given a unique name.
- Specify the order of the containers or segments by right-clicking the container or segment, and selecting **Move data structure up** or **Move data structure down**. The order is important for IMS segments and must match the order that is expected by the IMS transaction. For CICS containers, the order is a matter of preference.
 - Add multiple containers to a channel, or multiple segments to an IMS message. Add another container or segment by selecting a channel or message, right-clicking and selecting **Add**.
 - Remove a data structure from a container, COMMAREA, segment, or IBM MQ message by selecting the data structure and clicking **Remove data structure**.
 - Specify the fields to include or exclude, right-click a field, and select **Include field in interface** or **Exclude field from interface**.
 - Use the Edit Field wizard to rename fields or data structures, change field value types, specify default field values, and more. For information on how to edit and customize service fields, see [“Defining service fields” on page 497](#).
 - If one or more COBOL REDEFINES exist, select the redefined field to use. Original field name is selected by default. For more information, see [“Handling COBOL REDEFINES” on page 510](#).

Restriction:

For more information about restrictions on the size of each data structure see [“Restrictions on the data structure import function” on page 488](#).

For more information about how COBOL and PL/I fields are converted to JSON fields and related syntax restrictions, see the following topics:

- “[COBOL to JSON schema mapping](#)” on page 531

- “PL/I to JSON schema mapping” on page 535

Defining service fields

Service fields are used in service interfaces to map, transform, and modify the data used between a z/OS Connect EE API and the underlying z/OS application that the API utilizes. Customizing service fields allows you to provide metadata and optimize payloads for service interfaces.

Before you begin

If the service interface file is not already opened in the service interface editor, in Project Explorer, go to the service-interfaces/ folder for your project and double-click the service interface file (.si file).

From the service interface editor, you can edit service fields by using the **Edit Field** wizard. Access the **Edit Field** wizard for a given service field by right-clicking a service field, and selecting **Edit field**. You can also access the **Edit Field** wizard by selecting a service field, and then clicking the **Edit field** button in the Service interface editor.

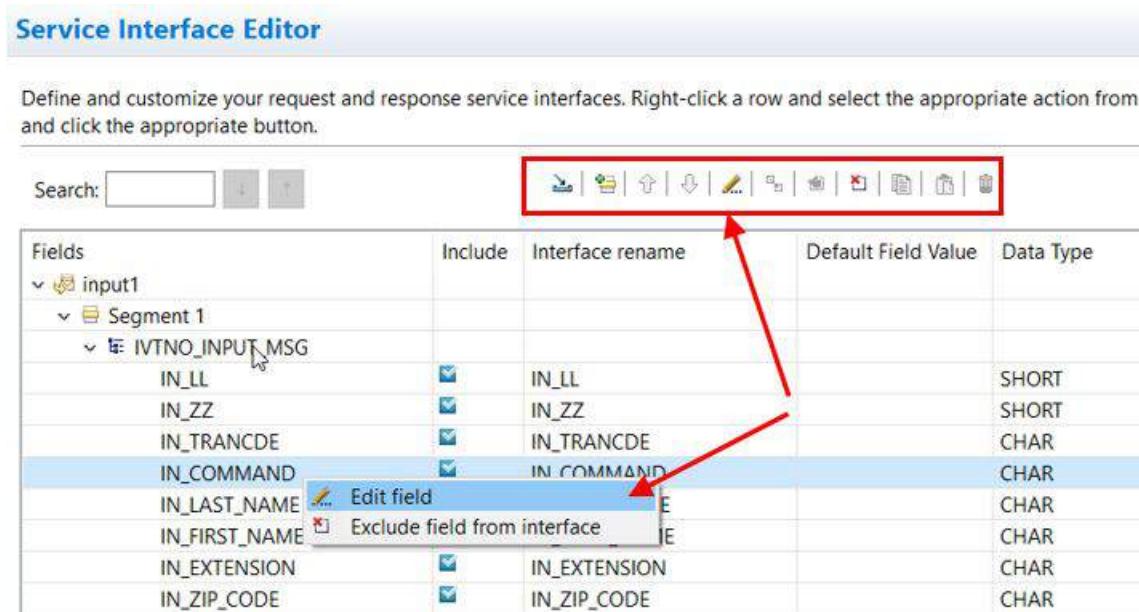


Figure 104. Customizing the interface

Tip:

The options of the **Edit Field** wizard change according to the data type of the service field being edited.

To learn about commonly used **Edit Field** wizard options, see “[Overview of the Edit Field wizard](#)” on page 498. To learn about other capabilities of the **Edit Field** wizard, see the sections following this overview.

Important:

When re-importing data structures, ensure that the definitions for service fields remain preserved. Errors caused by re-importing data structures are indicated by a symbol in the service interface editor, or in the **problems** view of the API toolkit.

The **Edit Field** wizard is capable of adapting service fields in a variety of ways to suit the needs of an API. The following sections describe all of the features provided by the **Edit Field** wizard:

- “[Overview of the Edit Field wizard](#)” on page 498
- “[Defining array counters](#)” on page 499
- “[Defining Boolean fields](#)” on page 501
- “[Defining Date fields](#)” on page 503

Overview of the Edit Field wizard

The **Edit Field** wizard offers the following options for most service field types:

- **Interface rename** allows you to specify a different name to be used for the z/OS Connect EE service.
- **Override data type** allows you to specify a new data type at the service level to perform conditional mediation between application and API data.
- Use **Default value type** to specify the datatype that is used for a default value.
- **Default value** is a datatype-specific value that is used by the service during runtime whenever there is no appropriate service field value specified by an API request or response.
- **Code page conversion** can be toggled on to convert service response code to CHAR data type.
- **Business description** metadata can be provided to clearly indicate the purpose of the service field as it relates to a service.

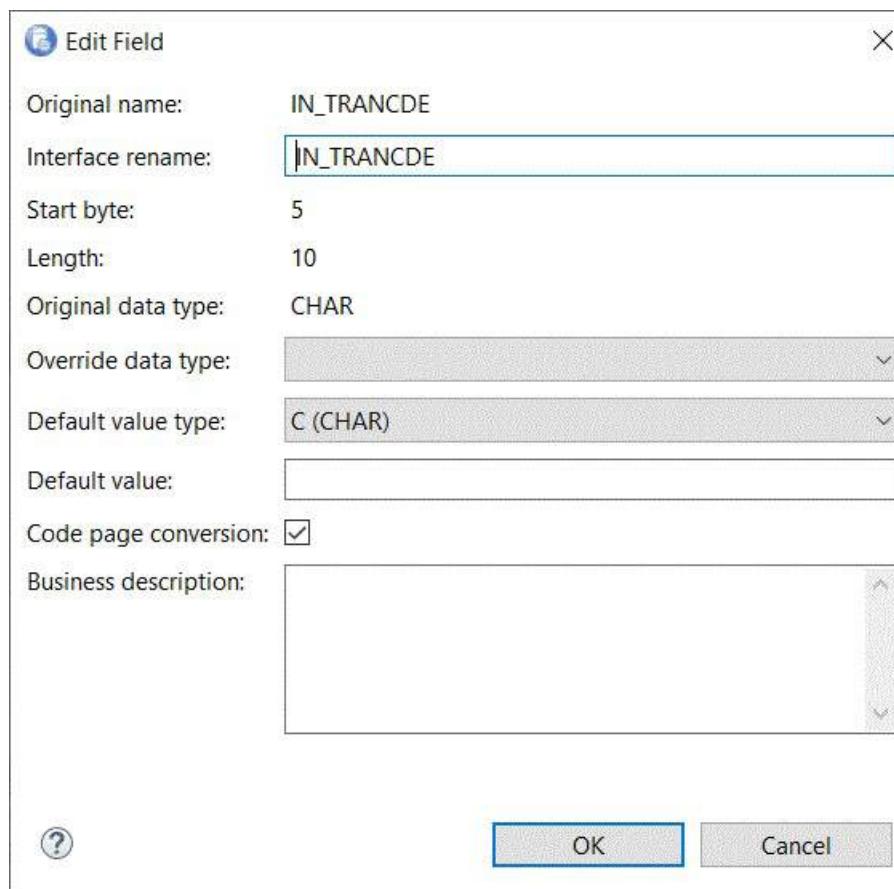


Figure 105. Editing a field with the Edit Field wizard

Tip: For service-level data conversion customization, see “[Service-level data conversion customization](#)” on page 537.

Important:

- Response service interfaces cannot contain default values in fields. If a default value is specified for a response service interface, a message will be issued in the **Problems** view to assist problem resolution.
- The default value can be overridden if a value is assigned in the API editor during API creation. If a value is specified in the payload at run time, the runtime value will always override the value that was specified at development time. To prevent the assigned default value from being overridden at run time, set the field to be omitted from the interface.

- The **Code page conversion** check box is only visible when the **Data type** field has a value of CHAR. To convert the code page for the CHAR type, select the **Code page conversion** check box; otherwise, clear it.

Defining array counters

Array counters can be used to optimize the request and response JSON for services which utilize arrays. For request interfaces, an array counter is automatically set to the number of array entries received in the request JSON for the respective array. For response interfaces, an array counter indicates the number of array entries to include in the response JSON from the respective array.

Array counters are defined by specifying a numeric service field to be used as a counter for fixed-length or variable-length arrays.

Restriction:

The following conditions must be met for a service field to be used as an array counter.

- Defining array counters requires installation of the minimum z/OS Connect EE server and API toolkit versions that support the array counters capability. To learn more about z/OS Connect EE version requirements for capabilities, see [“Capabilities compatibility” on page 751](#).
- Only integer service fields can be used for array counters.
- The array counter service field must reside in the same data structure as the array it counts for.
- The array counter cannot be an OCCURS DEPENDING ON or a REFER object.
- Array counters cannot be used to count more than one array.
- The array counter must be located before the array it counts for in the service data structure.
 - If the array that is using an array counter has a parent array in the service data structure, then the array counter must also have the same immediate parent array.
 - If the array has a parent REDEFINES, the array counter must also have the same immediate parent REDEFINES.

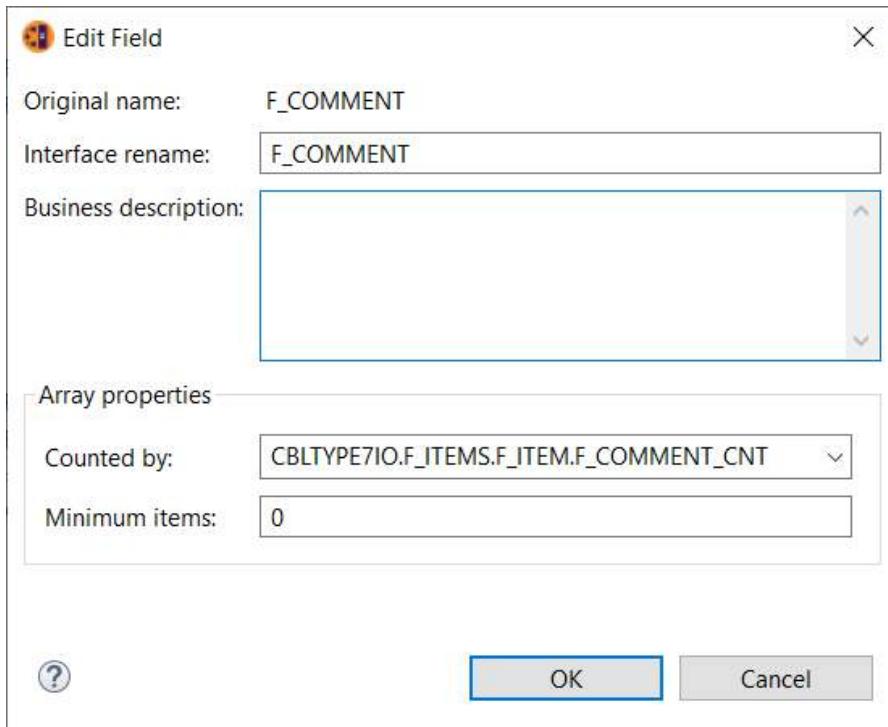
Service fields that do not meet this criteria will not be listed for selection when editing an array in the service interface editor.

For fixed-length or variable-length arrays, specify the following array properties using the **Edit Field** wizard to define and use array counters.

- Counted by** indicates a numeric field that specifies the number of sequential entries used in the array. This count always starts from the first entry of the array.

To change a counted array back to a non-counted array, select the empty line provided at the top of the drop-down list for the **Counted by** field.

- Minimum items** defines the minimum number of expected array entries. This minimum is expressed as minItems in the JSON schema that corresponds to the service interface.



Important:

- Counters specify the number of sequential array entries in use by a service, starting with and including the first entry.
- When setting a counter for an array, the minimum expected number of array entries can be customized. This minimum is reflected in the generated JSON schema (minItems).
 - To enforce the minimum number of array occurrences for API requests and responses, you must enable the **Enforce minimum array occurrence** options in your service project. To learn how to enforce minimum array occurrences, see [“Service-level data conversion customization” on page 537](#).

The following example of a service interface demonstrates where an array counter, F_COMMENT_CNT, can be used to define the number of F_COMMENT array entries within a service interface. Also included in this example is a fixed-length array, F_ROUTINGCODE, which is counted by F_ROUTINGCODE_CNT.

Fields	Include	Interface Rename	Data Type	Field Length	Start Byte
[#] F_ITEM_LIM	<input checked="" type="checkbox"/>	F_ITEM_LIM	INT	4	3055
F_ITEMS	<input checked="" type="checkbox"/>	F_ITEMS	STRUCT	16154	3059
[#] F_COMMENT_LIM	<input checked="" type="checkbox"/>	F_COMMENT_LIM	INT	4	3059
F_ITEM [1..50]	<input checked="" type="checkbox"/>	F_ITEM	ARRAY	16150	3063
F_PARTNUM	<input checked="" type="checkbox"/>	F_PARTNUM	CHAR	32	
F_PRODUCTNAME	<input checked="" type="checkbox"/>	F_PRODUCTNAME	CHAR	32	
F_QUANTITY	<input checked="" type="checkbox"/>	F_QUANTITY	DECIMAL	16	
F_USPRICE	<input checked="" type="checkbox"/>	F_USPRICE	DECIMAL	16	
F_AVAILABLE	<input checked="" type="checkbox"/>	F_AVAILABLE	CHAR	1	
F_DATEAVAIL	<input checked="" type="checkbox"/>	F_DATEAVAIL	CHAR	64	
F_COMMENT_CNT	<input type="checkbox"/>	F_COMMENT_CNT	DECIMAL	2	
F_COMMENT [0..5]	<input checked="" type="checkbox"/>	F_COMMENT	ARRAY	160	
F_ROUTINGCODE_CNT	<input type="checkbox"/>	F_ROUTINGCODE_CNT	USHORT	2	
F_ROUTINGCODE [3..3]	<input checked="" type="checkbox"/>	F_ROUTINGCODE	ARRAY	24	

Defining Boolean fields

Service fields can be exposed in JSON payloads as type Boolean to create more easily consumable APIs. Boolean service fields allow for the definition of specific request and response conditions to process true or false values for a given service field, while leaving the underlying COBOL or PL/I application field values unchanged.

To expose a given service field as Boolean, select the **Boolean** option in the **Override data type** property of the **Edit Field** wizard.

Restriction:

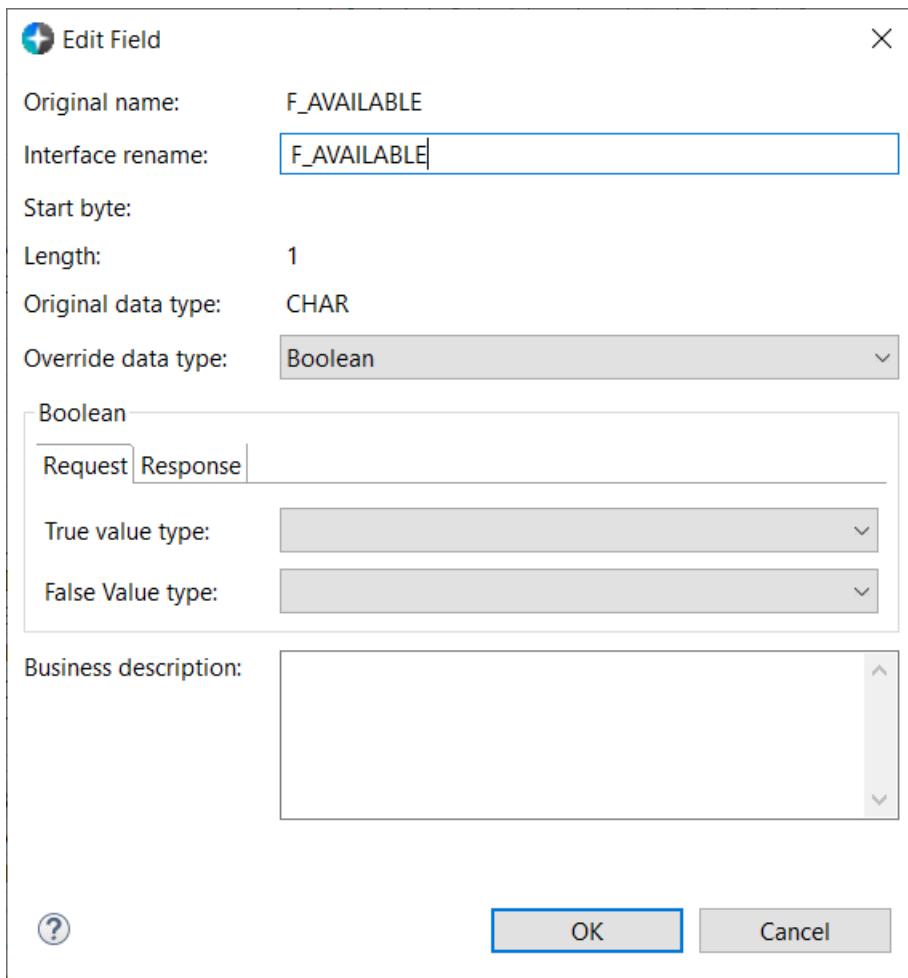
The following conditions must be met for a service field to be overridden as type Boolean:

- Overriding service fields as type Boolean requires installation of the minimum z/OS Connect EE server and API toolkit versions that support the boolean override capability. To learn more about z/OS Connect EE version requirements for capabilities, see [“Capabilities compatibility” on page 751](#).
- The application data type for the service field is Char.
- If the application data type for the service field is not Char:
 - The service field's application data type must not be Float or Double.
 - For decimal service field data types, decimal scale must be set to 0 so that fields can be read as integers.
- PL/I data types that are VARYING cannot be overridden as Boolean service fields.

Service fields that do not meet this criteria will not display the option to be overridden to Boolean in the service interface editor.

Specify the following properties using the **Edit Field** wizard to define and use Boolean service fields.

- Under the **Request** tab:
 - The **True value type** and **False value type** field properties determine how service request JSON values are transformed into application data. True and false request values can be defined with the following options:
 - **Single values** use a precise value to determine if a service field value is true or false. Single values must be of the same datatype as the **Original data type**, unless hex values are used.
 - **High values** are byte values consisting of all 0xFF .
 - **Low values** are byte values consisting of all 0x00.
- Under the **Response** tab:
 - The **True value type** and **False value type** field properties define how service field values are determined as true for an API response. In addition to **single values**, **high values**, and **low values** options, true and false response values can be defined with the following additional options:
 - **Range of values** uses a specified lower and upper bound value to determine if a service field value is true or false. Value ranges are inclusive and must be of the same datatype as the **Original data type**, unless hex values are used.
 - **Multiple values** consist of a delimited set of values which determine if a service field value is true or false. You can choose the delimiter that is used between values.



Once a service field has been overridden as type Boolean, it will be marked in the **Data Type** column of the service interface editor as Boolean, in parenthesis. Also, when looking at service fields using the API field mapping editor, fields that are overridden as type Boolean will be listed as Boolean fields.

Important:

- For the JSON request and response payload, the only valid values are `true` and `false`, without single or double quotes. Any values other than `true` and `false` cannot be used.
- If a service response value does not match either of the specified true or false conditions, an API response of `null` is issued instead.

The following example demonstrates how a service field that is overridden as type Boolean appears in the service interface editor. In this example, the F_AVAILABLE service field is overridden to express a CHAR value as a Boolean data type in the resulting API request and response JSON.

Fields	Include	Interface Rename	Data Type	Field Length	Start Byte
F_ITEMS	<input checked="" type="checkbox"/>	F_ITEMS	STRUCT	16654	3059
[#] F_COMMENT_LIM	<input checked="" type="checkbox"/>	F_COMMENT_LIM	INT	4	3059
▼ [!] F_ITEM [1..50]	<input checked="" type="checkbox"/>	F_ITEM	ARRAY	16650	3063
> F_COUNTERS	<input type="checkbox"/>	F_COUNTERS	STRUCT	12	
F_PARTNUM	<input checked="" type="checkbox"/>	F_PARTNUM	CHAR	32	
F_PRODUCTNAME	<input checked="" type="checkbox"/>	F_PRODUCTNAME	CHAR	32	
F_QUANTITY	<input checked="" type="checkbox"/>	F_QUANTITY	DECIMAL	16	
F_USPRICE	<input checked="" type="checkbox"/>	F_USPRICE	DECIMAL	16	
F_AVAILABLE	<input checked="" type="checkbox"/>	F_AVAILABLE	CHAR (Boolean)	1	
F_DATEAVAIL	<input checked="" type="checkbox"/>	F_DATEAVAIL	CHAR	64	
[!] F_COMMENT [0..5]	<input checked="" type="checkbox"/>	F_COMMENT	ARRAY	160	

Defining Date fields

Service fields can be exposed and consumed in JSON payloads as date string patterns in order to create effective APIs that utilize validated dates. Date service fields allow for the exposing of host dates as *OpenAPI™ 2.0* compliant dates, while leaving the underlying COBOL or PL/I application field values unchanged.

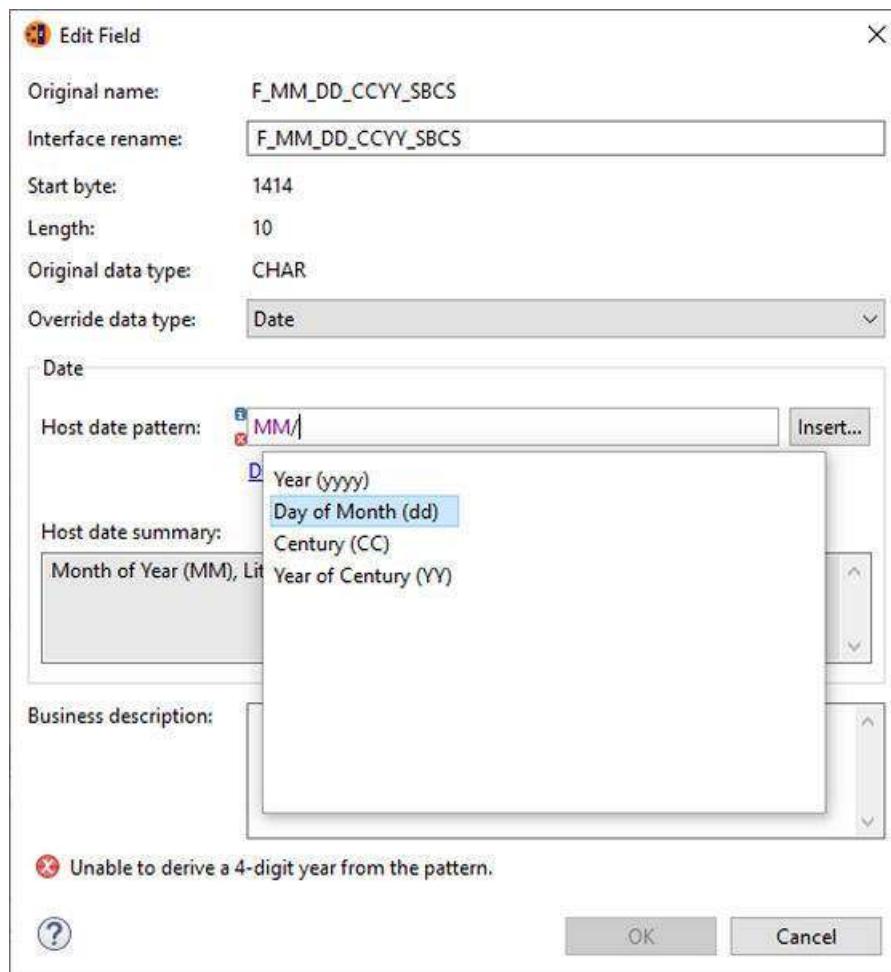
To override a given service field as a Date, select the **Date** option in the **Override data type** property of the **Edit Field** wizard.

Restriction:

- Overriding service fields as dates requires installation of the minimum z/OS Connect EE server and API toolkit versions that support the date override capability. To learn more about z/OS Connect EE version requirements for capabilities, see “[Capabilities compatibility](#)” on page 751.
- CICS CHAR containers cannot be redefined as a Date service field.

Specifying date service fields can be achieved by using one, or a combination, of the following methods from within the **Edit Field** wizard. Use these methods to create both simple, and complex, date service field patterns to suit the needs of your services and APIs.

- In the **Host date pattern** field of the **Edit Field** wizard, you can begin to type the date pattern of an underlying application field. By pressing **Ctrl + Space**, content assist will suggest matching date components that are applicable to your pattern.
- Alternatively, you can click the **insert...** button next to the **Host date pattern** field. A dialogue box will open containing all available date components. Select a component and click **OK** to insert the date component at the current cursor position in the **Host date pattern** field.



In addition to these methods, the **Redefine as CHAR field** function of the **service interface editor** can be used to redefine a STRUCT as a single character field, allowing the content of the STRUCT to be exposed as a date field. To use the **Redefine as CHAR field** function, right-click the STRUCT in the **service interface editor** and select **Redefine as CHAR field**.

Restriction: The following restrictions apply to the **Redefine as CHAR field** function:

- Top-level data structures cannot be overridden as characters by using the **Redefine as CHAR field** function of the **service interface editor**.
- The **Redefine as CHAR field** function only applies to STRUCT type application fields which meet the following conditions:
 - The STRUCT is not already a redefining field.
 - The STRUCT does not contain redefining fields, also known as nested redefines.
 - The STRUCT does not contain ARRAY fields.
 - All numeric fields in the STRUCT are zoned decimal with "sign is separate" is signed.
 - All fields in the STRUCT have the same encoding. For example, all SBCS, all Unicode, or all DBCS.

If the preceding conditions are not met for a STRUCT type application field, messages will be logged when verbose logging is enabled in the API toolkit that indicate why the service field could not be overridden.

- The following additional properties apply to service fields that are overridden as CHAR fields.
 - The redefining field created by the **Redefine as CHAR field** function is considered a "User defined field", and can be later removed by using the **Remove user defined field** context menu action.
 - The redefining field created by the **Redefine as CHAR field** function will be preserved during data structure re-import if the following conditions are met:
 - The redefined STRUCT still exists in the re-imported data structure.
 - The redefined STRUCT still satisfies the requirements for the **Redefine as CHAR field** function.
- Some examples featured in this section use the **Redefine as CHAR field** function. This capability is only required to enable the definition of STRUCTs as character groups, and is not required to override existing numeric and character fields as dates.

Tip:

- For requests, the host date pattern is used to convert a JSON date value to a host date value. For responses, the host date pattern is used to convert a host date value to a JSON date value.
- Default JSON data types can be overridden where necessary by using the data transformation function of the z/OS Connect EE API toolkit. To learn more, see "[COBOL to JSON schema mapping](#)" on page [531](#) and "[PL/I to JSON schema mapping](#)" on page [535](#).
- For requests, an invalid JSON date value will result in an HTTP response code 400 that includes either message GMOMW0024E or GMOMW0025E. For responses, an invalid host date value will cause the field to be omitted from the JSON, and message GMOMW0026E will be issued by the z/OS Connect EE server.
- Services that are updated to utilize date fields must be re-imported into APIs that use those services. When re-importing services that utilize date fields, the **impact analysis** view will specify which API's are affected and how.

The following table contains a list of accepted host date patterns:

Component name	Component syntax	Description
Day of Month	dd	Two digit day of Month.
Day of Year	DDD	Three digit ordinal Day.
Month of Year	MM	Two digit month of Year.
Year	yyyy	Four digit Year.

Component name	Component syntax	Description
Year of Century	YY	Two digit Year. This component must be followed by a Century component.
Century	CC	Two digit century. This component must be preceded by a Year of Century component.
Literal ('')	' value '	Single quotes that contain a literal consisting of one or more characters. Characters in a quoted literal are not parsed as date pattern components. <i>Example:</i> 'abcMM'
Hex literal (%'')	%' hex value '	A literal consisting of one or more hex-encoded bytes. Hex literals are not supported for numeric fields. <i>Example:</i> %'CAFE'
Literal escape ()	\ character	An escaped literal consisting of one character. <i>Example:</i> \'

The following table contains all COBOL fields that are acceptable for a Date override. Only the following COBOL field types may be used for a Date override:

Note: The following tables feature Japanese NLS character equivalents for separator characters, such as a "/" or a "-", that can be used for Host Date patterns.

Field type	Example declarations	Example Host Date patterns
PIC [S]9(n) USAGE COMP	05 YYYYDDD PIC [S]9(7) COMP 05 YYYYMMDD PIC [S]9(8) COMP	yyyyDDD yyyyMMdd
PIC [S]9(n) USAGE COMP-3	05 YYYYDDD PIC [S]9(7) COMP-3 05 DDMMYYYY PIC [S]9(8) COMP-3	yyyyDDD ddMMyyyy
PIC [S]9(n) USAGE DISPLAY	05 YYYYDDD PIC [S]9(7) DISPLAY 05 MMDDYYCC PIC [S]9(8) DISPLAY	yyyyDDD MMddYYCC
PIC X(n) USAGE DISPLAY	05 MMDDYYYY PIC X(8) DISPLAY 05 MM-DD-YYYY PIC X(10) DISPLAY 05 MM-DD-YYYY PIC 99/99/9999 DISPLAY	MMddyyyy MM/dd/yyyy MM/dd/yyyy
PIC N(n) USAGE NATIONAL	05 YYYYMMDD PIC N(8) NATIONAL 05 YYYY-MM-DD PIC N(11) NATIONAL 05 YYYY-MM-DD PIC 99/99/9999 NATIONAL	MMddyyyy yyyy 年 MM 月 dd 日 yyyy/MM/dd
PIC X(n) USAGE DISPLAY-1	05 YYYYMMDD PIC X(10) DISPLAY-1 05 YYYY-MM-DD PIC X(20) DISPLAY-1	MMddyyyy yyyy 年 MM 月 dd 日

Field type	Example declarations	Example Host Date patterns
PIC 9(n) USAGE NATIONAL	05 YYYYDDD PIC 9(7) NATIONAL 05 YYCCMMDD PIC 9(8) NATIONAL	yyyyDDD YYCCmmdd
PIC S9(n) USAGE NATIONAL SIGN [LEADING TRAILING] SEPARATE	05 YYYYDDD PIC S9(7) NATIONAL SIGN LEADING SEPARATE 05 YYCCMMDD PIC S9(8) NATIONAL SIGN TRAILING SEPARATE	yyyyDDD YYCCmmdd

The following table contains all PL/I fields that are acceptable for a Date override. Only the following PL/I field types may be used for a Date override:

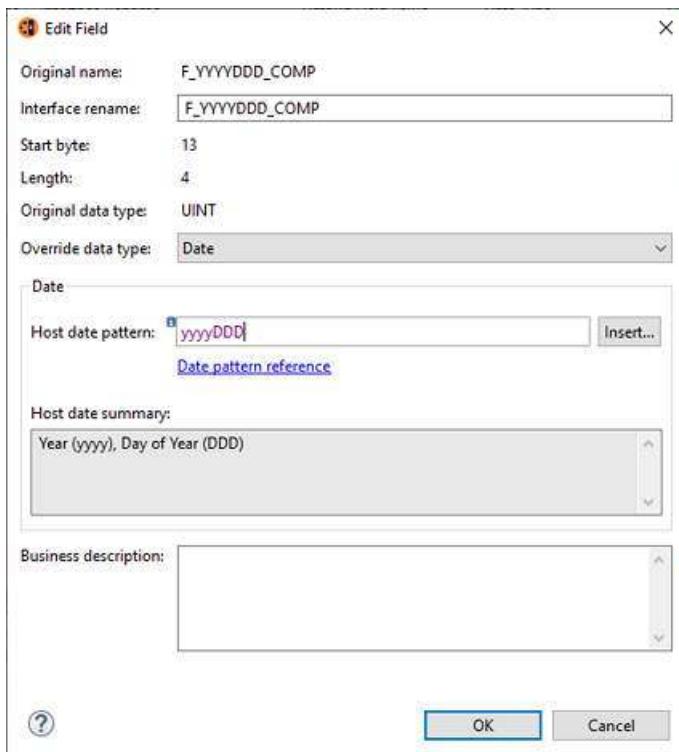
Field type	Example declarations	Example Host Date patterns
FIXED DECIMAL(n,0)	05 YYYYDDD FIXED DECIMAL(7) 05 YYYYMMDD FIXED DECIMAL(8)	yyyyDDD yyyyMMdd
PICTURE '(n)9'	05 YYYYDDD PICTURE '(7)9' 05 DDMMYYYY PICTURE '(8)9'	yyyyDDD ddMMyyyy
PICTURE 'S(n)9' or '(n)9S'	05 YYCCDDD PICTURE 'S(7)9' 05 YYCCMMDD PICTURE '(8)9S'	YYCCDDD YYCCMMdd
CHAR(n) NONVARYING	05 MMDDYYYY CHAR(8) NONVARYING 05 MM_DD_YYYY CHAR(10) NONVARYING 05 YYYY_MM_DD CHAR(20) NONVARYING	MMddyyyy MM/dd/yyyy yyyy 年 MM 月 dd 日
WCHAR(n) NONVARYING	05 YYYYMMDD WCHAR(8) NONVARYING 05 YYYY_MM_DD WCHAR(11) NONVARYING	MMddyyyy yyyy 年 MM 月 dd 日

Numeric date override example

The following example demonstrates the overriding of a numeric field as a date, and the resulting JSON payload. In this example, the following declaration is used by the application to represent a date:

```
04 F-YYYYDDD-COMP PIC 9(07) COMP.
```

The following override is applied to the service field:



During runtime for this service, the following JSON schema is constructed for this field:

```
"F_YYYYDDD_COMP" : {
  "type" : "string",
  "format" : "date"
}
```

The following value is provided in the response JSON at runtime:

```
"F_YYYYDDD_COMP" : "2020-02-29"
```

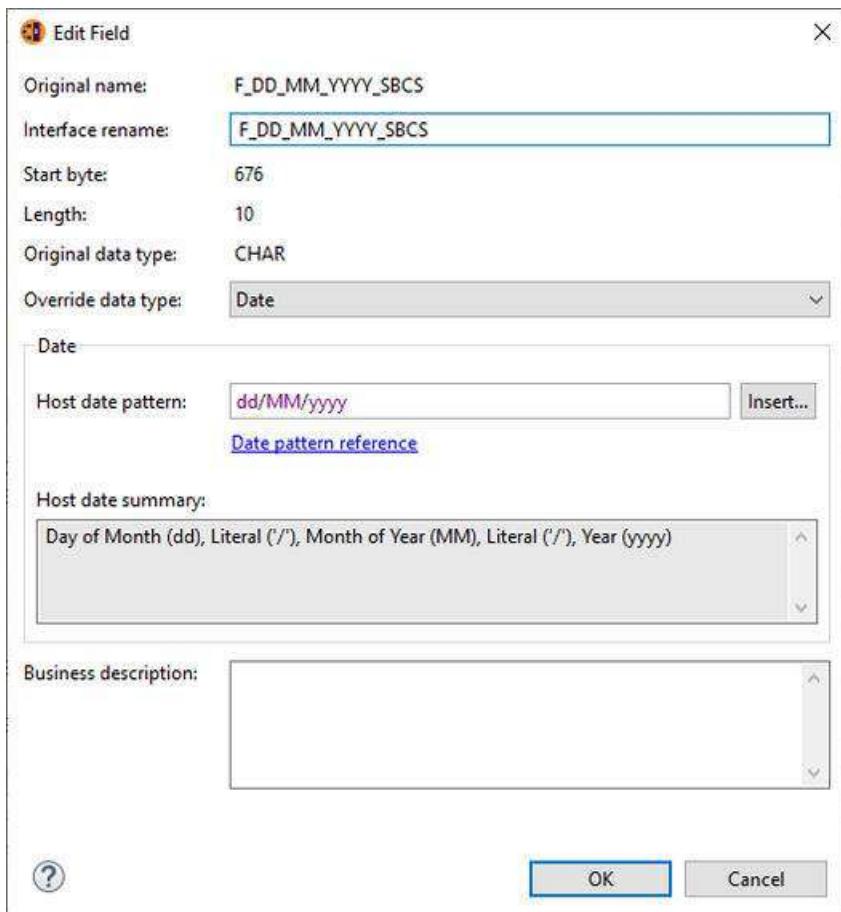
For the request, the service takes the JSON date value of "2020-02-29" and converts it to a host date value of 2020060 using the pattern yyyyDDD. For the response, the host date value of 2020060 is converted to a JSON date value of "2020-02-29" using the same pattern, yyyyDDD.

Character date override example

The following example demonstrates the overriding of a character field as a date, and the resulting JSON payload. In this example, the following declaration is used by the application to represent a date:

```
04 F-DD-MM-YYYY-SBCS PIC X(10) DISPLAY.
```

The following override is applied to the service field:



During runtime for this service, the following JSON schema is constructed for this field:

```
"F_DD_MM_YYYY_SBCS" : {
    "type" : "string",
    "format" : "date"
}
```

The following value is provided in the response JSON at runtime:

```
"F_DD_MM_YYYY_SBCS" : "2020-02-29"
```

For the request, the service takes a JSON date value of "2020-02-29" and converts it to a host date value of 29/02/2020 using the pattern dd/MM/yyyy. For the response, the host date value of 29/02/2020 is converted to a JSON date value of "2020-02-29" using the same pattern, dd/MM/yyyy.

Group date override example

The following example demonstrates the overriding of a STRUCT as characters, then defining those character fields as dates, and the resulting JSON payload. In this example, the following declaration is used by the application to represent a date:

```
04 F-DD-MM-YYYY-DETAIL-SBCS.
 04 F-MM      PIC 9(02) DISPLAY.
 04 FILLER   PIC X(01) VALUE '/'.
 04 F-DD      PIC 9(02) DISPLAY.
 04 FILLER   PIC X(01) VALUE '/'.
 04 F-YYYY    PIC 0(04) DISPLAY.
```

The STRUCT is overridden as a character field group in the **service interface editor** by right-clicking the STRUCT and selecting the **Redefine as CHAR field** function.



The following override is applied to the service field group:

Original name:	detailSBCS_MM/dd/yyyy
Interface rename:	detailSBCS_MM/dd/yyyy
Start byte:	1549
Length:	10
Original data type:	CHAR
Override data type:	Date
Date	
Host date pattern:	MM/dd/yyyy
Date pattern reference	
Host date summary:	
Month of Year (MM), Literal ('/'), Day of Month (dd), Literal ('/'), Year (yyyy)	
Business description:	

During runtime for this service, the following JSON schema is constructed for this field:

```
"detailSBCS_MM/dd/yyyy" : {
    "type" : "string",
    "format" : "date"
}
```

The following value is provided in the response JSON at runtime:

```
"detailSBCS_MM/dd/yyyy" : "2020-02-29"
```

For the request, the service takes the JSON date value of "2020-02-29" and converts it to a host date value of 02/29/2020 using the pattern MM/dd/yyyy. For the response, the host date value of 02/29/2020 is converted to a JSON payload of "2020-02-29" using the same pattern, MM/dd/yyyy.

Related tasks

["Customizing the service interface" on page 496](#)

You can customize the payload data structures that comprise the request and response service interfaces by copying data structures from one location into another, move data structures up or down, or removing data structures. Service interface fields can be further customized by using the **Edit field** wizard to specify service field names, default values, descriptions, and more.

Related reference

[“Service-level data conversion customization” on page 537](#)

For each service, you can specify to exclude special, non-printable control characters or suppress low values or empty tags from the JSON response messages. You can also specify whether and how to initialize fields in the input message.

[“COBOL to JSON schema mapping” on page 531](#)

The data transformation function in the z/OS Connect EE API toolkit converts COBOL fields to JSON fields.

[“PL/I to JSON schema mapping” on page 535](#)

The data transformation function in the z/OS Connect EE API toolkit converts PL/I fields to JSON fields.

Handling COBOL REDEFINES

If a COBOL field is redefined by one or more REDEFINES statements, by default, the original field name is selected. You can select a group of REDEFINES to be exposed in your service.

If a field is redefined by the REDEFINES clause multiple times for different data description entries, expose only one group of the REDEFINES in your service. Because the last definition overrides previous definitions, exposing multiple definitions for a field could result in unexpected behaviors and errors. Use different services, one for each REDEFINES group, to ensure the correct values in correct data types are passed through.

In the following example, Field-1 is redefined by Group-1 and again by Group-2. The two groups contain different data description entries. For request messages, the last definition, Group-2, is used at run time. To ensure both types of input are preserved, expose Group-1 definition in one service and Group-2 in another. For output (response) messages, multiple views of the data would be returned. If the values are not valid, empty tags are returned.

```
05 Field-1 PICTURE X(100).  
05 Group-1 REDEFINES Field-1.  
    10 Field-2 PICTURE 9(4) comp-3.  
    10 Field-3 PICTURE X(96).  
05 Group-2 REDEFINES Field-1.  
    10 Field-4 PICTURE 9(4) comp-5.  
    10 Field-5 PICTURE X(50).  
    10 Field-6 PICTURE X(46).
```

Related reference

[“COBOL to JSON schema mapping” on page 531](#)

The data transformation function in the z/OS Connect EE API toolkit converts COBOL fields to JSON fields.

Configuring service properties

Configure the service properties for connection and interaction with the target z/OS subsystem by clicking the **Configuration** tab.

For CICS services (COMMAREA, or channel), specify the following properties:

Table 51. Properties for CICS services

Property	Importanc e	Description
Coded character set identifier (CCSID)	Required	Specifies the CCSID that is used at run time to encode character data in COMMAREA and BIT container application data structures. Default is IBM-037. For more information, see “ Coded character set identifiers ” on page 515 and “ Considerations for services using the API or build toolkits ” on page 275.
Connection reference	Required	Specifies the ID of the CICS connection.
Transaction ID	Optional	Specifies the CICS transaction name. The transaction ID usage value specifies how the transaction ID value is used. If this parameter is not specified, the transaction ID value in the connection is used.

Table 51. Properties for CICS services (continued)

Property	Importanc e	Description
Transaction ID usage	Optional	Specifies how the transaction ID value is used. Valid values are EIB_ONLY and EIB_AND_MIRROR. If this value is not specified, the transaction ID usage value in the connection is used.
Use context containers	Optional	Channel services only When enabled, the following context containers are sent to CICS: DFHWS-URI, DFHWS-URI-QUERY, and DFHHTTPMETHOD. The BAQHTTPHEADERS container is sent only when headers are configured in the Context containers HTTP headers property. For more information, see “ Context containers ” on page 230.
Context containers HTTP headers	Optional	Channel services only The HTTP headers to include in the BAQHTTPHEADERS context container if they are present in the HTTP request. This property is enabled only if Use context containers is enabled.

For IMS services, specify the following properties:

Table 52. Properties for IMS services

Property	Importanc e	Description
Connection profile	Required	Specifies the name of the connection profile to use at service invocation.
Interaction profile	Required	Specifies the name of the interaction profile to use at service invocation.
Program name	Optional	Specifies the program name to identify the IMS program that is associated by the service. An IMS program can be triggered by multiple transactions, which define scheduling rules, output limits, and more. The program itself determines how a transaction is processed and how it looks to the user. This optional value is for informational purposes and can be used to identify all services that are associated with a specific program.
IMS destination override	Optional	Specifies a different IMS destination (IMS data store) to use, overriding the one that is defined in the interaction profile. With this option, you can reuse the profiles across multiple z/OS Connect EE services regardless of the IMS destination that the profile was originally defined for.

For IBM MQ services, specify the following properties. Select the table that matches your configuration.

Table 53. Properties for IBM MQ two-way services

Property	Importanc e	Description
Connection factory JNDI name	Required	Defines a JNDI name that is used to locate a connection factory that connects to an IBM MQ for z/OS queue manager on the same LPAR as the z/OS Connect EE server or on a different LPAR. For more information, see JMS connection factory in the <i>WebSphere® Application Server for z/OS Liberty</i> documentation.
Request destination JNDI name	Required	Defines a JNDI name that is used to locate an IBM MQ queue where request messages can be sent. For more information, see JMS Queue (jmsQueue) in the <i>WebSphere® Application Server for z/OS Liberty</i> documentation.
Reply destination JNDI name	Required	Defines a JNDI name that is used to locate an IBM MQ queue that contains reply messages. For more information, see JMS Queue (jmsQueue) in the <i>WebSphere® Application Server for z/OS Liberty</i> documentation.
Wait interval	Required	Specifies how long, in milliseconds the IBM MQ service provider waits for messages to arrive on a queue. Tip: You should set the wait interval to a value less than the <code>asyncRequestTimeout</code> value to prevent the HTTP request from timing out in the z/OS Connect EE server.
MQMD format	Required	Used in the format field of the MQMD header in messages that are sent by the IBM MQ service provider. For more information, see “Considerations for services using the API or build toolkits” on page 275
Coded character set identifier (CCSID)	Required	Specifies the CCSID that is used at run time to encode character data in application data structures. Default is 37, which is IBM®-037. For the list of CCSSIDs supported in z/OS Connect EE, see Coded character set identifiers . For more information, see “Considerations for services using the API or build toolkits” on page 275
Is message persistent	Required	Describes the persistence of messages that are sent to the IBM MQ queue. The default value is false, indicating that messages are non-persistent. If set to true, messages are persistent. Persistent messages survive queue manager restart; non-persistent messages do not.

Table 53. Properties for IBM MQ two-way services (continued)

Property	Importanc e	Description
Reply selection	Required	<p>Defines how the service locates reply messages on the reply destination. The following values are valid:</p> <ul style="list-style-type: none"> • <code>msgIDToCorrelID</code> Reply messages are assumed to be generated with the correlation ID set to the value of the message ID from the request message. The service generates a suitable message selector based on this information. This option is the default value. • <code>none</code> No mechanism is used to correlate reply messages with request messages. The service gets the first available message on the reply queue. • <code>correlIDToCorrelID</code> Reply messages are assumed to be generated with the correlation ID set to the value of the correlation ID from the request message. The service generates a suitable message selector based on this information. If the request message does not specify a correlation ID, the service generates a random correlation ID for the request message. For more information, see ibm-mq-md-correlID.
Expiry	Required	<p>Specifies the expiry time of messages that are sent by the IBM MQ service provider. If set, the value is an integer that describes how long the message is available in milliseconds before it expires. By default, messages do not expire.</p> <p>Setting <code>expiry</code> is equivalent to setting the MQMD Expiry field.</p> <p>Negative values mean that messages never expire. The default value is -1.</p> <p>REST clients can override <code>expiry</code> by specifying an <code>ibm-mq-md-expiry</code> HTTP header with a valid 64-bit integer.</p> <p>Tip: Change <code>expiry</code> from the default to a value that is both appropriate for the back-end service and is a relatively small number of seconds. Otherwise there is the potential for reply messages to build up on the reply destination and not processed by the IBM MQ service provider.</p>

Table 54. Properties for IBM MQ one-way services for sending

Property	Importanc e	Description
Connection factory JNDI name	Required	<p>Defines a JNDI name that is used to locate a connection factory that connects to an IBM MQ for z/OS queue manager on the same LPAR as the z/OS Connect EE server or on a different LPAR.</p> <p>For more information, see JMS connection factory in the WebSphere® Application Server for z/OS Liberty documentation.</p>
Destination JNDI name	Required	<p>Defines a JNDI name that is used to locate an IBM MQ queue or topic where request messages can be sent.</p> <p>For more information, see JMS Queue (jmsQueue) in the WebSphere® Application Server for z/OS Liberty documentation.</p>

Table 54. Properties for IBM MQ one-way services for sending (continued)

Property	Importanc e	Description
Coded character set identifier (CCSID)	Required	Specifies the CCSID that is used at run time to encode character data in application data structures. The default is 37, which is IBM-037. For the list of CCSIDs supported in z/OS Connect EE, see “Coded character set identifiers” on page 515 and “Considerations for services using the API or build toolkits” on page 275.
MQMD format	Required	Used in the format field of the MQMD header in messages that are sent by the IBM MQ service provider. For more information, see “Considerations for services using the API or build toolkits” on page 275.
Is message persistent	Required	Describes the persistence of messages that are sent to the IBM MQ queue. The default value is false, indicating that messages are non-persistent. If set to true, messages are persistent. Persistent messages survive queue manager restart; non-persistent messages do not.
Expiry	Required	<p>Specifies the expiry time of messages that are sent by the IBM MQ service provider. If set, the value is an integer that describes how long the message is available in milliseconds before it expires. By default, messages do not expire.</p> <p>Setting <code>expiry</code> is equivalent to setting the MQMD <code>Expiry</code> field.</p> <p>Negative values mean that messages never expire. The default value is -1.</p> <p>REST clients can override <code>expiry</code> by specifying an <code>ibm-mq-md-expiry</code> HTTP header with a valid 64-bit integer.</p>

Table 55. Properties for IBM MQ one-way services for receiving

Property	Importanc e	Description
Connection factory JNDI name	Required	<p>Defines a JNDI name that is used to locate a connection factory that connects to an IBM MQ for z/OS queue manager on the same LPAR as the z/OS Connect EE server or on a different LPAR.</p> <p>For more information, see JMS connection factory in the WebSphere® Application Server for z/OS Liberty documentation.</p>
Destination JNDI name	Required	<p>Defines a JNDI name that is used to locate an IBM MQ queue or topic from which the service receives messages.</p> <p>For more information, see JMS Queue (jmsQueue) in the WebSphere® Application Server for z/OS Liberty documentation.</p>
Coded character set identifier (CCSID)	Required	Specifies the CCSID that is used at run time to encode character data in application data structures. The default is 37, which is IBM-037. For the list of CCSIDs supported in z/OS Connect EE, see “Coded character set identifiers” on page 515.

Table 55. Properties for IBM MQ one-way services for receiving (continued)

Property	Importanc e	Description
Wait interval	Optional	<p>Specifies how long, in milliseconds, the IBM MQ service provider waits for messages to arrive on the queue. If the value is blank or 0, the service provider does not wait for messages. A negative value means that the service provider waits until a message is available.</p> <p>Tip: You should set the wait interval to a value less than 30000 to prevent the HTTP request from timing out in the z/OS Connect EE server. The wait interval should be set to a value that is just under, or the same as the value of the expiry property.</p>
Message selector	Optional	<p>Defines a valid JMS message selector that is used to locate messages from the queue.</p> <p>For more information, see Message selectors in JMS in the <i>IBM MQ</i> documentation.</p> <p>REST clients can override <code>expiry</code> by specifying an <code>ibm-mq-md-expiry</code> HTTP header with a valid 64-bit integer.</p>

Coded character set identifiers

You must specify the coded character set identifier (CCSID) field when configuring a CICS service in the service project editor. Coded character set identifiers are used at run time to encode character data in COMMAREA and BIT container application data structures.

The following table shows the CCSIDs that are supported in z/OS Connect EE.

Table 56. Coded character set identifiers

CCSID	Encoding	Additional information
37	IBM037 IBM-037 Cp037	Host: USA, Canada (ESA), Netherlands, Portugal, Brazil, Australia, New Zealand
273	IBM273 IBM-273 Cp273	Host: Austria, Germany
277	IBM277 IBM-277 Cp277	Host: Denmark, Norway
278	IBM278 IBM-278 Cp278	Host: Finland, Sweden
280	IBM280 IBM-280 Cp280	Host: Italy

Table 56. Coded character set identifiers (continued)

CCSID	Encoding	Additional information
284	IBM284 IBM-284 Cp284	Host: Spain, Latin America (Spanish)
285	IBM285 IBM-285 Cp285	Host: United Kingdom
290	IBM-290	JAPANESE EBCDIC
297	IBM297 IBM-297 Cp297	Host: France
300	IBM-300	JAPAN DB EBCDIC
301	IBM-301	PC data: Japan DB
420	IBM420 IBM-420 IBM-420S Cp420	Host: Arabic
424	IBM424 IBM-424 Cp424	Host: Hebrew
437	IBM437 IBM-437 US-ASCII ASCII Cp437 US-ASCII	PC data: PC Base; USA, many other countries
500	IBM500 IBM-500 Cp500	Host: Belgium, Canada (AS/400), Switzerland, International Latin-1
720	IBM-720	MSDOS ARABIC
737	IBM-737 x-IBM737	MSDOS GREEK

Table 56. Coded character set identifiers (continued)

CCSID	Encoding	Additional information
775	IBM775 IBM-775	MSDOS BALTIC
808	IBM-808	PC data: Cyrillic, Russia; with euro
813	ISO-8859-7 ISO8859_7	ISO 8859-7: Greece
819	ISO-8859-1 ISO8859_1	ISO 8859-1: Latin-1 countries
833	IBM-833	KOREAN EBCDIC
834	IBM-834 x-IBM834	KOREAN DB EBCDIC
835	IBM-835	T-CHINESE DB EBCD
836	IBM-836	S-CHINESE EBCDIC
837	IBM-837	S-CHINESE EBCDIC
850	IBM850 IBM-850 Cp850	PC data: Latin-1 countries
852	IBM852 IBM-852 Cp852	PC data: Latin-2 multilingual
855	IBM855 IBM-855 Cp855	PC data: Cyrillic
856	IBM-856 x-IBM856 Cp856	PC data: Hebrew
857	IBM857 IBM-857 Cp857	PC data: Latin-5 (Turkey)

Table 56. Coded character set identifiers (continued)

CCSID	Encoding	Additional information
858	IBM00858 IBM-858 Cp858	PC data: Latin-1 countries; with euro
859	IBM-859	PC data: LATIN-9
860	IBM860 IBM-860	PC data: Portugese
861	IBM861 IBM-861	PC data: Iceland
862	IBM862 IBM-862 Cp862	PC data: Hebrew (migration)
863	IBM863 IBM-863	PC data: Canada
864	IBM864 IBM-864 IBM-864S Cp864	PC data: Arabic
865	IBM865 IBM-865 Cp865	PC data: Den/Norway
866	IBM866 IBM-866 Cp866	PC data: Cyrillic, Russia
867	IBM-867	PC data: Hebrew with euro
868	IBM868 IBM-868 Cp868	PC data: Urdu
869	IBM869 IBM-869 Cp869	PC data: Greece

Table 56. Coded character set identifiers (continued)

CCSID	Encoding	Additional information
870	IBM870 IBM-870 Cp870	Host: Latin-2 multilingual
871	IBM871 IBM-871 Cp871	Host: Iceland
874	x-IBM874	PC data: Thai
875	IBM-875 x-IBM875 Cp875	Host: Greece
897	IBM-897	PC data: Japan SB
912	ISO-8859-2 ISO8859_2	ISO 8859-2: Latin-2 multilingual
915	ISO-8859-5 ISO8859_5	ISO 8859-5: Cyrillic
916	ISO-8859-8 ISO8859_8	ISO 8859-8: Hebrew
918	IBM918 IBM-918 Cp918	Host: Urdu
920	ISO-8859-9 ISO8859_9	ISO 8859-9: Latin-5 (ECMA-128, Turkey TS-5881)
921	IBM-921 x-IBM921 Cp921	PC data: Latvia, Lithuania
922	IBM-922 x-IBM922 Cp922	PC data: Estonia
923	ISO-8859-15 Cp923 ISO8859_15_FDIS	ISO 8859-15: Latin-9

Table 56. Coded character set identifiers (continued)

CCSID	Encoding	Additional information
924	IBM-924	ISO 8859-15: Latin-9
927	IBM-927	PC data: T-Chinese
930	IBM-930 x-IBM930 Cp930	Katakana Host: extended SBCS Kanji Host: DBCS including 4370 user-defined characters Katakana Host: extended SBCS Kanji Host: DBCS including 1880 user-defined characters
932	IBM-932	PC data: Japan Mix
933	IBM-933 x-IBM933 Cp933	Host: Extended SBCS Host: DBCS including 1880 user-defined characters and 11172 full Hangul characters
935	IBM-935 x-IBM935 Cp935	Host: Extended SBCS Host: DBCS including 1880 user-defined characters
937	IBM-937 x-IBM937 Cp937	Host: Extended SBCS Host: DBCS including 6204 user-defined characters
939	IBM-939 x-IBM939 Cp939	Latin Host: extended SBCS Kanji Host: DBCS including 4370 user-defined characters Latin Host: extended SBCS Kanji Host: DBCS including 1880 user-defined characters
942	IBM-942 IBM-942C x-IBM942 x-IBM942C Cp942 Cp942C	PC data: Extended SBCS PC data: DBCS including 1880 user-defined characters

Table 56. Coded character set identifiers (continued)

CCSID	Encoding	Additional information
943	IBM-943 IBM-943C Shift_JIS windows-31j windows-932 x-IBM943 x-IBM943C Cp943 Cp943C MS932	PC data: SBCS PC data: DBCS for Open environment including 1880 IBM user-defined characters
947	IBM-947	T-CHINESE BIG-5
948	IBM-948 x-IBM948 Cp948	PC data: Extended SBCS PC data: DBCS including 6204 user-defined characters
949	IBM-949 IBM-949C x-IBM949 x-IBM949C Cp949 Cp949C	IBM KS Code - PC data: SBCS IBM KS code - PC data: DBCS including 1880 user-defined characters
950	Big5 IBM-950 x-IBM950 Cp950	PC data: SBCS (IBM BIG5) PC data: DBCS including 13493 CNS, 566 IBM selected, 6204 user-defined characters
951	IBM-951	PC data: IBM KS
954	EUC-JP IBM-954 IBM-954C	G0: JIS X201 Roman G1: JIS X208-1990 G1: JIS X201 Katakana G1: JIS X212
964	EUC-TW IBM-964 x-IBM964 Cp964	G0: ASCII G1: CNS 11643 plane 1 G1: CNS 11643 plane 2

Table 56. Coded character set identifiers (continued)

CCSID	Encoding	Additional information
970	EUC-KR x-IBM970 Cp970	G0: ASCII G1: KSC X5601-1989 including 1880 user-defined characters
971	IBM-971	KOREAN EUC
1006	IBM-1006 x-IBM1006 Cp1006	ISO-8: Urdu
1025	IBM-1025 x-IBM1025 Cp1025	Host: Cyrillic multilingual
1026	IBM1026 IBM-1026 Cp1026	Host: Latin-5 (Turkey)
1027	IBM-1027	JAPAN LATIN EBCD
1041	IBM-1041	PC data: Japan
1043	IBM-1043	PC data: T-Chinese
1046	IBM-1046 IBM-1046S x-IBM1046	ARABIC - PC
1047	IBM1047 IBM-1047	Host: Latin-1
1051	hp-roman8	HP EMULATION
1088	IBM-1088	PC data: Korea KS
1089	ISO-8859-6 ISO8859_6	ISO 8859-6: Arabic
1097	IBM-1097 x-IBM1097 Cp1097	Host: Farsi
1098	IBM-1098 x-IBM1098 Cp1098	PC data: Farsi

Table 56. Coded character set identifiers (continued)

CCSID	Encoding	Additional information
1112	IBM-1112 x-IBM1112 Cp1112	Host: Latvia, Lithuania
1114	IBM-1114	PC data: T-CH SB
1115	IBM-1115	PC data: S-CH GB
1122	IBM-1122 x-IBM1122 Cp1122	Host: Estonia
1123	IBM-1123 x-IBM1123 Cp1123	Host: Cyrillic Ukraine
1124	IBM-1124 x-IBM1124 Cp1124	8-bit: Cyrillic, Belarus
1140	IBM01140 IBM-1140 Cp1140	Host: USA, Canada (ESA), Netherlands, Portugal, Brazil, Australia, New Zealand; with euro
1141	IBM01141 IBM-1141 Cp1141	Host: Austria, Germany; with euro
1142	IBM01142 IBM-1142 Cp1142	Host: Denmark, Norway; with euro
1143	IBM01143 IBM-1143 Cp1143	Host: Finland, Sweden; with euro
1144	IBM01144 IBM-1144 Cp1144	Host: Italy; with euro
1145	IBM01145 IBM-1145 Cp1145	Host: Spain, Latin America (Spanish); with euro

Table 56. Coded character set identifiers (continued)

CCSID	Encoding	Additional information
1146	IBM01146 IBM-1146 Cp1146	Host: United Kingdom; with euro
1147	IBM01147 IBM-1147 Cp1147	Host: France; with euro
1148	IBM01148 IBM-1148 Cp1148	Host: Belgium, Canada (AS/400), Switzerland, International Latin-1; with euro
1149	IBM01149 IBM-1149 Cp1149	Host: Iceland; with euro
1200	UTF-16BE	Unicode with character set 65535. In the absence of a byte-order mark (BOM), assumed to be UTF-16 BE (big-endian).
1202	UTF-16LE	UTF-16 LE with IBM PUA
1204	UTF-16	UTF-16 with IBM PUA
1208	UTF-8 UTF-8J UTF8	Unicode with character set 65535. UTF-8.
1232	UTF-32BE	UTF-32 BE with IBM PUA
1234	UTF-32LE	UTF-32 LE with IBM PUA
1236	UTF-32	UTF-32 with IBM PUA
1351	IBM-1351	JAPAN OPEN
1362	IBM-1362	KOREAN MS-WIN
1363	IBM-1363 IBM-1363C windows-949 MS949	PC data: MS Windows Korean SBCS PC data: MS Windows Koran DBCS including 11172 full Hangul

Table 56. Coded character set identifiers (continued)

CCSID	Encoding	Additional information
1364	IBM-1364	Host: Extended SBCS Host: DBCS including 1880 user-defined characters and 11172 full Hangul characters
1370	IBM-1370	PC data: Extended SBCS; with euro PC data: DBCS including 6204 user-defined characters; with euro
1371	IBM-1371	Host: Extended SBCS; with euro Host: DBCS including 6204 user-defined characters; with euro
1375	Big5-HKSCS	Mixed Big-5 Ext for HKSCS
1380	IBM-1380	PC data: S-CH GB
1381	IBM-1381 x-IBM1381 Cp1381	PC data: Extended SBCS (IBM GB) PC data: DBCS (IBM GB) including 31 IBM-selected, 1880 user-defined characters
1382	IBM-1382	S-CHINESE EUC
1383	EUC-CN GB2312 IBM-1383 x-IBM1383 Cp1383	G0: ASCII G1: GB 2312-80 set
1385	IBM-1385	PC data: S-CH GBK
1386	GBK IBM-1386 windows-936 MS936	PC data: S-Chinese GBK and T-Chinese IBM BIG-5 PC data: S-Chinese GBK
1388	IBM-1388	Host: Extended SBCS Host: DBCS including 1880 user-defined characters

Table 56. Coded character set identifiers (continued)

CCSID	Encoding	Additional information
1390	IBM-1390	Katakana Host: extended SBCS; with euro Kanji Host: DBCS including 6205 user-defined characters
1399	IBM-1399	Latin Host: extended SBCS; with euro Kanji Host: DBCS including 4370 user-defined characters; with euro
5050	JIS0201 JIS0208 JIS0212 JIS0201 JIS0208 JIS0212	G0: JIS X201 Roman G1: JIS X208-1990 G1: JIS X201 Katakana G1: JIS X212
5054	ISO-2022-JP	JAPANESE TCP
5346	windows-1250 Cp1250	MS Windows: Latin-2, version 2 with euro
5347	windows-1251 Cp1251	MS Windows: Cyrillic, version 2 with euro
5348	windows-1252 Cp1252	MS Windows: Latin-1 countries, version 2 with euro
5349	windows-1253 Cp1253	MS Windows: Greece, version 2 with euro
5350	windows-1254 Cp1254	MS Windows: Turkey, version 2 with euro
5351	windows-1255 Cp1255	MS Windows: Hebrew, version 2 with euro
5352	windows-1256 windows-1256S Cp1256	MS Windows: Arabic, version 2 with euro
5353	windows-1257 Cp1257	MS Windows: Baltic Rim, version 2 with euro

Table 56. Coded character set identifiers (continued)

CCSID	Encoding	Additional information
5354	windows-1258 Cp1258	MS Windows: Vietnamese, version 2 with euro
5488	GB18030	GB18030, 1-byte data GB18030, 2-byte data GB18030, 4-byte data
9030	IBM-838 Cp838	Host: Thai extended SBCS
9066	IBM-874 Cp874	PC data: Thai extended SBCS
9400	CESU-8	CESU-8 with IBM PUA
25546	ISO-2022-KR	KOREAN TCP
33722	IBM-33722 IBM-33722C	IBMeucJP

How to specify a custom CCSID for a CICS service

You can specify a custom CCSID when you create a CICS service.

Before you begin

Refer to “[Coded character set identifiers](#)” on page 515 to check whether the code page that you want to use is supported in z/OS Connect EE by default. Specify a custom CCSID only when you have a specific requirement for it.

About this task

To specify a custom CCSID in z/OS Connect EE, you must extend the Java Runtime Environment (JRE) to support your custom character encoding with your own `CharsetProvider` and `Charset` classes.

Procedure

1. Build a custom `CharsetProvider` JAR file that contains your own `CharsetProvider` and `Charset` class implementations.

When you implement the `CharsetProvider` class, you must set a canonical name or an alias name for your custom character encoding. This canonical name or alias name is composed of (from left to right) a specific string "Cp" and a numeric string. The numeric string is the CCSID that you need to specify when you generate a service archive (.sar).

The following two code examples show how to initialize a new character set with two kinds of names.

Example 1: using a canonical name

```
import java.nio.charset.spi.CharsetProvider;
public class CustomCharsetProvider extends CharsetProvider {
    // The canonical name of the custom charset
    private static final String CUSTOM_CHARSET_NAME = "Cp99999";
```

```

// A handler to the Charset object
private Charset custom_charset = null;

public CustomCharsetProvider() {
    String[] alias = new String[] { CUSTOM_CHARSET_NAME };
    this.custom_charset = new CustomCharset(CUSTOM_CHARSET_NAME, alias);
}
...
}

```

In this example, "Cp99999" is a canonical encoding and the numeric part "99999" is used to specify the CCSID later.

Example 2: using an alias name

```

import java.nio.charset.Charset;
import java.nio.charset.spi.CharsetProvider;

public class CustomCharsetProvider extends CharsetProvider {
    // The canonical name of the custom charset
    private static final String CUSTOM_CHARSET_NAME="Unicode-at-on";
    // The specific numeric alias name of the custom charset
    private static final String CUSTOM_CHARSET_NAME_ALIAS = "Cp99999";
    // A handler to the Charset object
    private Charset custom_charset = null;

    public CustomCharsetProvider() {
        String[] alias = new String[] { CUSTOM_CHARSET_NAME_ALIAS };
        this.custom_charset = new CustomCharset(CUSTOM_CHARSET_NAME, alias);
    }
    ...
}

```

In this example, "Unicode-at-on" is a canonical encoding that is not supported in z/OS Connect EE by default. "Cp99999" is the alias name that is set for "Unicode-at-on" and the numeric part "99999" is used to specify the CCSID later.

For more information about how to build a custom `CharsetProvider` JAR file, see the Java API documentation for `CharsetProvider`.

2. Deploy the custom `CharsetProvider` JAR file to JRE.

- Copy the JAR file to the z/OS Connect EE runtime JRE extensions directory, for example, `$ZCEERUNTIME_JRE_Location/jre/lib/ext`.
- Depending on which toolkit that you use to generate the service archive (`.sar`), copy the JAR file to the corresponding JRE extensions directory.
 - If you use the API toolkit, ensure the JAR file is deployed to the API toolkit JRE extensions directory, for example, `$ZOSExplorer_Install_Location/jdk/jre/lib/ext`.
 - If you use the build toolkit, ensure the JAR file is deployed to the build toolkit JRE extensions directory, for example, `$BUILDTOOLKIT_JRE_Location/jre/lib/ext`.

Results

The JRE is now extended to support your custom character encoding.

What to do next

Use the custom code page to generate a `.sar` file by setting the numeric string (for example, "99999") specified in Step 1 to the CCSID property in the service project editor or in the build toolkit properties file as required. For more information about how to generate a `.sar` file and deploy it in z/OS Connect EE, see ["Create a CICS service" on page 83](#).

Note: The specified numeric string, (for example, "99999") can also be set to the `cicsCCSID` property when a policy is applied. For more information about the `cicsCCSID` property, see ["Valid properties for use in policy rules" on page 687](#).

When the .sar file is generated and deployed, invoke the service and check the response to verify that the custom code page is used in z/OS Connect EE.

Deploying a service

Deploy your service to the server directly from the API toolkit if the server connection is already properly configured.

Before you begin

Deploying a service directly from within the API toolkit requires server code V3.0.4 or later.

You must create a host connection to the z/OS Connect EE server. For more information, see “[Connecting to a z/OS Connect EE server](#)” on page 603.

About this task

If the service project is open, you can deploy a service directly by clicking the **Deploy Service to z/OS Connect EE Server** button  in the corner.

You can also deploy one or more services from the **Project Explorer** view.

Procedure

To deploy a service from the **Project Explorer** view:

1. In the **Project Explorer** view, select one or more service projects and right-click to select **z/OS Connect EE > Deploy Service to z/OS Connect EE Server**.
2. In the **Deploy Service** window, select the server to which to deploy the services.
3. If a service of the same name exists and you want to overwrite it, select the **Update existing services** check box.

Results

Deployment result is reported. Errors that occur during the deployment are recorded in the **Problems** view. A BAQR7070E error indicates that the **zosconnect_services** element is not configured in the **server.xml** configuration file.

A deployed service is automatically started unless specified otherwise in the z/OS Connect EE preferences window.

Tips:

- To get a refreshed view of all deployed services and their status on the server, in the **z/OS Connect EE Servers** view, right-click the **Services** folder, and select **Refresh**.
- To get a refreshed view of all deployed services and APIs and their status on a server, right-click the server itself and select **Refresh Server**.

What to do next

[Export your service project as a service archive \(a .sar file\)](#). This file is needed to create an API to invoke the service.

Related concepts

[“Setting preferences for the API toolkit” on page 606](#)

Starting, stopping, or removing a service

You can start, stop, or remove a service in the **z/OS Connect EE Servers** view.

Before you begin

Switch to the **z/OS Connect Enterprise Edition** perspective.

Ensure that you are connected to the server. For more information, see “[Connecting to a z/OS Connect EE server](#)” on page 603.

About this task

The **z/OS Connect EE Servers** view in the **z/OS Connect Enterprise Edition** perspective provides a list of defined host connections and the services and APIs that are deployed on the servers.

Procedure

1. In the **z/OS Connect EE Servers** view, ensure that the server is connected. If the server is not connected, right-click the server and select **Connect**.
2. Expand the server and the **Services** folder to see all services that are deployed to the server.
3. For each service, you can examine its status, service URL, invoke URL, related service provider in the **Properties** view by right-clicking the service and select **Show Properties View**.
4. To start, stop, or remove a service, right-click the service and select **Start Service**, **Stop Service**, or **Remove Service**.

Related concepts

[“Setting preferences for the API toolkit” on page 606](#)

Exporting the service project

To create an API or deploy a service through an automated DevOps process, you need a service archive (a .sar file). You can export your service project as a service archive to a location that you specify.

Procedure

1. In the **Project Explorer** view, right-click the service project and select **z/OS Connect EE > Export z/OS Connect EE Service Archive**.
2. Specify the location where you want to save this service archive file.

Results

The service is saved as a .sar file.

What to do next

- For API creation, see [“Creating a REST API” on page 589](#).
- In a DevOps environment, the service archive is generated by using the build toolkit with the service project directory as input. See [“Generating service archives for DevOps” on page 560](#).
- For automated service deployment, see [“Automated service archive management” on page 670](#).

High-level languages and JSON schema mapping

A mapping is the set of rules that specifies how information is converted between high-level language structures and JSON schemas. The z/OS Connect EE API toolkit provides the user interface for specifying and customizing the input and output data format.

When you create or edit a service, you can specify that a field is included or excluded, and specify a default value. You can also specify whether to escape special, non-printable characters or suppress low values or empty tags in the JSON response messages.

Note: If a JSON property is passed with the value “ ” (the empty string), then the underlying field is initialized to 0x40s (spaces). If a JSON property is not passed, the underlying field is not initialized and remains as 0x00s (null). Passing an empty string is not semantically the same as omitting a property.

The flowchart in [Figure 106 on page 531](#) shows how values are assigned to input fields for a request.

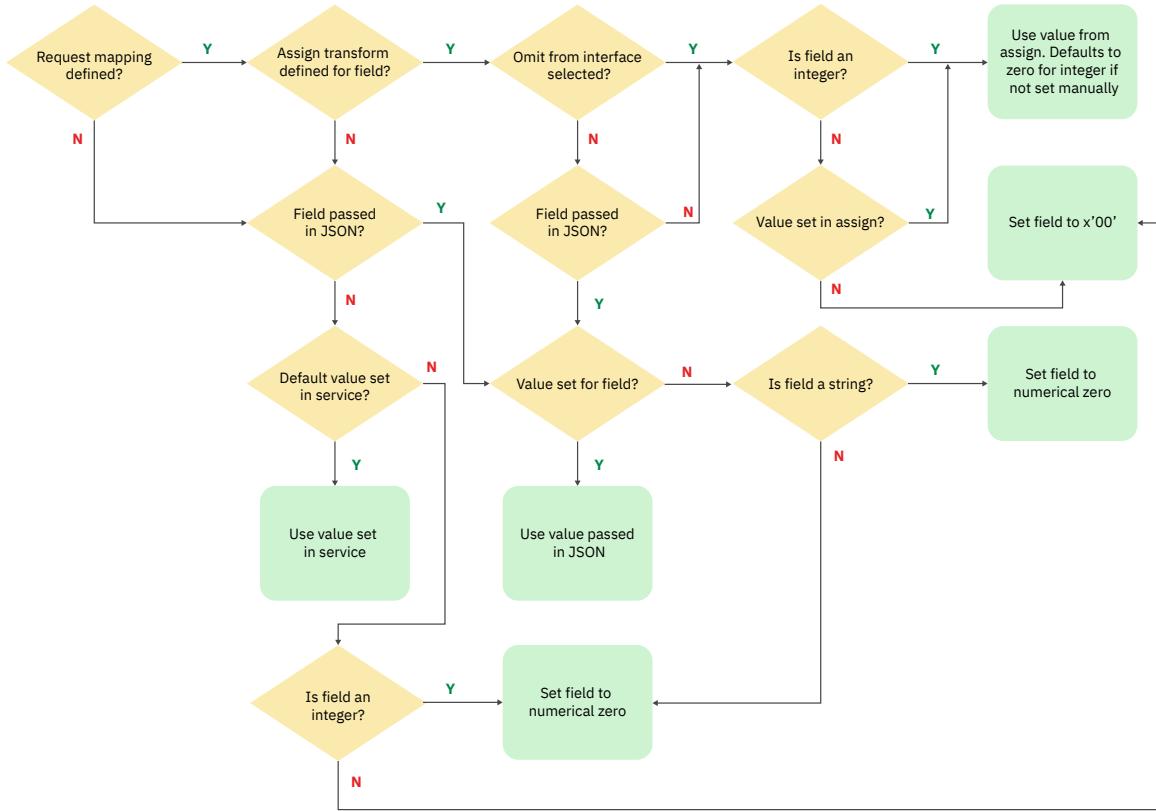


Figure 106. Assignment of values to request input fields

COBOL to JSON schema mapping

The data transformation function in the z/OS Connect EE API toolkit converts COBOL fields to JSON fields.

OCCURS DEPENDING ON (ODO) clauses could involve array lists or variable-length arrays and require computation. For input messages, the values are computed rather than from the message itself. On the response, only the ODO subjects are returned. ODO support requires that an ODO object exist in the same structure as the referencing ODO subject. ODO objects are not included as they are of no value or interest to API developers.

COBOL data description elements are mapped to schema elements according to the following table. COBOL data description elements that are not shown in the table are not supported. For example, a real decimal such as PIC +9(9).9(2) is not supported. While the implied decimal type such as PIC S9(9)V9(2) or just PIC 9(9)V9(2) can be used to attempt to represent a real decimal, these values are represented differently in memory and result in a different file record format on disk. Real decimal and implied decimal types cannot be used interchangeably.

Table 57. COBOL to JSON schema mapping

COBOL type	JSON type
USAGE DISPLAY: PIC X(n) PIC A(n)	"type": "string", "maxLength": n
USAGE DISPLAY-1: PIC G(n)	
USAGE NATIONAL: PIC N(n)	

Table 57. COBOL to JSON schema mapping (continued)

COBOL type	JSON type
USAGE DISPLAY: PIC S9(n)	<pre>"type": "integer", "minimum": -n, "maximum": n</pre>
USAGE BINARY,COMP, COMP-4: PIC S9(1<=n<=4) PIC S9(5<=n<=9) PIC S9(10<=n<=18)	<p>where n is the maximum value that can be represented by the pattern of '9' characters.</p>
USAGE COMP-5: PIC 9(n)	<pre>"type": "integer", "minimum": -(x + 1), "maximum": x</pre> <p>where x is the maximum value that can be represented by $2^{(m*8)-1}$.</p> <p>m is the number of storage representation for n.</p> <p>For example, $m = 2$ for $1 \leq n \leq 4$ and $m = 4$ for $5 \leq n \leq 9$ and $m = 8$ for $10 \leq n \leq 18$</p>
USAGE COMP-5: PIC 9(n)	<pre>"type": "integer", "minimum": -0, "maximum": x</pre> <p>where x is the maximum value that can be represented by $2^{(m*8)}$.</p> <p>m is the number of storage representation for n.</p> <p>For example, $m = 2$ for $1 \leq n \leq 4$ and $m = 4$ for $5 \leq n \leq 9$ and $m = 8$ for $10 \leq n \leq 18$</p>
USAGE DISPLAY: PIC 9(n)	<pre>"type": "integer", "minimum": 0, "maximum": n</pre>
USAGE BINARY,COMP, COMP-4: PIC 9(1<=n<=4) PIC 9(5<=n<=9) PIC 9(10<=n<=18)	<p>where n is the maximum value that can be represented by the pattern of '9' characters.</p>
USAGE COMP-3: PIC S9(n)	<pre>"type": "number", "description": "decimal", "minimum": -x, "maximum": y, "multipleOf": z</pre> <p>where:</p> <ul style="list-style-type: none"> • x is the minimum value that can be represented by the pattern of '9' characters • y is the maximum value that can be represented by the pattern of '9' characters • z is the smallest unit available = $1 / 10^n$

Table 57. COBOL to JSON schema mapping (continued)

COBOL type	JSON type
USAGE DISPLAY: PIC 9(m)V9(n)	<pre>"type": "number", "description": "decimal", "minimum": 0, "maximum": y, "multipleOf": z</pre>
	<p>where:</p> <ul style="list-style-type: none"> • y is the maximum value that can be represented by the pattern of '9' characters • z is the smallest unit available = $1 / 10^n$
USAGE COMP-5: PIC S9(m)V9(n)	<pre>"type": "number", "description": "decimal", "minimum": -(x+1), "maximum": x, "multipleOf": z</pre>
	<p>where:</p> <ul style="list-style-type: none"> • x is the maximum value that can be represented by $2^{((k*8)-1)}$ with scaling factor (that is the decimal position of n) • k is the number of storage representation for m, For example, $k = 2$ for $1 \leq m \leq 4$ and $k = 4$ for $5 \leq m \leq 9$ and $k = 8$ for $10 \leq m \leq 18$ • z is the smallest unit available = $1 / 10^n$
USAGE COMP-5: PIC 9(m)V9(n)	<pre>"type": "number", "description": "decimal", "minimum": 0, "maximum": x, "multipleOf": z</pre>
	<p>where:</p> <ul style="list-style-type: none"> • x is the maximum value that can be represented by $2^{(k*8)}$ with scaling factor (that is the decimal position of n) • k is the number of storage representation for m, For example, $k = 2$ for $1 \leq m \leq 4$ and $k = 4$ for $5 \leq m \leq 9$ and $k = 8$ for $10 \leq m \leq 18$ • z is the smallest unit available = $1 / 10^n$
USAGE DISPLAY: PIC S9(m)V9(n)	<pre>"type": "number", "description": "decimal", "minimum": -x, "maximum": y, "multipleOf": z</pre>
USAGE BINARY,COMP, COMP-4: PIC S9(m)V9(n)	<p>where:</p> <ul style="list-style-type: none"> • x is the minimum value that can be represented by the pattern of '9' characters • y is the maximum value that can be represented by the pattern of '9' characters • z is the smallest unit available = $1 / 10^n$

Table 57. COBOL to JSON schema mapping (continued)

COBOL type	JSON type
USAGE COMP-3: PIC 9(m)V9(n)	<pre>"type": "number", "description": "decimal", "minimum": 0, "maximum": y, "multipleOf": z</pre>
SAGE BINARY,COMP, COMP-4: PIC 9(m)V9(n)	<p>where:</p> <ul style="list-style-type: none"> • y is the maximum value that can be represented by the pattern of '9' characters • z is the smallest unit available = 1 / 10ⁿ
USAGE COMP-1	<pre>"type": "number", "description": "float"</pre>
USAGE COMP-2	<pre>"type": "number", "description": "double"</pre>
OCCURS n	<pre>"type": "array", "items": { "type": "object", "properties": { // nested properties or objects } }, "minItems": n, "maxItems": n</pre>
OCCURS n TIMES PIC X OCCURS n TIMES PIC A OCCURS n TIMES PIC G DISPLAY-1 OCCURS n TIMES PIC N	<pre>"type": "array", "items": { "maxLength": 1, "type": "string" }, "minItems": n, "maxItems": n</pre>
OCCURS n TO m TIMES DEPENDING ON t PIC X OCCURS n TO m TIMES DEPENDING ON t PIC A OCCURS n TO m TIMES DEPENDING ON t PIC G DISPLAY-1 OCCURS n TO m TIMES DEPENDING ON t PIC N	<pre>"type": "array", "items": { "maxLength": 1, "type": "string" }, "minItems": n, "maxItems": m</pre>
OCCURS n TO m TIMES DEPENDING ON t PIC X(k) OCCURS n TO m TIMES DEPENDING ON t PIC A(k) OCCURS n TO m TIMES DEPENDING ON t PIC G(k) DISPLAY-1 OCCURS n TO m TIMES DEPENDING ON t PIC N(k)	<pre>"type": "array", "items": { "maxLength": k, "type": "string" }, "minItems": n, "maxItems": m</pre>

Table 57. COBOL to JSON schema mapping (continued)

COBOL type	JSON type
OCCURS <i>n</i> TO <i>m</i> TIMES DEPENDING ON <i>t</i>	<pre> "items": { "type": "object", "properties": { // nested properties or objects } }, "type": "array", "minItems": <i>n</i>, "maxItems": <i>m</i> </pre>

Related information

[“Defining Date fields” on page 503](#)

PL/I to JSON schema mapping

The data transformation function in the z/OS Connect EE API toolkit converts PL/I fields to JSON fields.

PL/I data description elements are mapped to schema elements according to the following table. PL/I data description elements that are not shown in the table are not supported.

For REFER statements, only simple REFER usage is supported. A REFER usage is simple if only one structure element uses REFER, and that element has no later siblings and no parents with siblings, and that element is one of the following types:

- A scalar string or AREA
- A one-dimensional array of CHAR with constant LBOUND
- An array of elements of constant size and with only the upper bound in the first dimension non-constant

Table 58. PL/I to JSON schema mapping

PL/I type	JSON type
CHARACTER:	
CHAR(<i>n</i>)	"type": "string" "maxLength": <i>n</i>
GRAPHIC(<i>n</i>)	
WIDECHAR(<i>n</i>)	
CHARACTER:	Not supported
CHAR(<i>n</i>) VARYING	
ORDINAL	Not supported
FIXED DECIMAL(<i>n</i> , <i>m</i>)	<pre> "type": "number", "description": "decimal", "minimum": -<i>x</i>, "maximum": <i>y</i>, "multipleOf": <i>z</i> </pre>
	where: <ul style="list-style-type: none"> • <i>x</i> is the minimum value that can be represented by the pattern of '9' characters. • <i>y</i> is the maximum value that can be represented by the pattern of '9' characters. • <i>z</i> is the smallest unit available = 1 / 10^{<i>n</i>}

Table 58. PL/I to JSON schema mapping (continued)

PL/I type	JSON type
PICTURE '999V99'	<pre>"type": "number", "description": "decimal", "minimum": 0, "maximum": y, "multipleOf": z</pre>
	<p>where:</p> <ul style="list-style-type: none"> • y is the maximum value that can be represented by the pattern of '9' characters. • z is the smallest unit available = $1 / 10^n$.
PICTURE '\$999.99'	<pre>"type": "string" "maxLength": n</pre>
FIXED BINARY (n)	<pre>"type": "integer", "minimum": -(k + 1), "maximum": k</pre>
	<p>where k is the maximum value that can be represented by the primitive.</p>
UNSIGNED FIXED BINARY (n)	<pre>"type": "integer", "minimum": 0, "maximum": k</pre>
	<p>where k is the maximum value that can be represented by the primitive.</p>
BINARY FLOAT($01 \leq n \leq 21$)	<pre>"type": "number", "description": "float"</pre>
BINARY FLOAT($21 \leq n \leq 53$)	<pre>"type": "number", "description": "double"</pre>
DECIMAL FLOAT(n)	<pre>"type": "number", "description": "float"</pre>
where $n \leq 6$	
DECIMAL FLOAT(n)	<pre>"type": "number", "description": "double"</pre>
where $6 < n \leq 16$	
Values greater than 16 are not supported.	
BIT(n)	<pre>"type": "string" "maxLength": m where m = n/8</pre>
where n is a multiple of 8. Other values are not supported.	

Table 58. PL/I to JSON schema mapping (continued)

PL/I type	JSON type
name (n) <i>data description</i>	<p>For data declarations:</p> <pre>"type": "array" "maxOccurs": n "minOccurs": n "items": { <i>data description</i> JSON }</pre>

The following additional restrictions also apply:

- If the COMPLEX attribute is specified for a data item, it is ignored.
- If the PRECISION attribute is specified for a data item, it is ignored.
- Enterprise PL/I FLOAT IEEE is not supported.
- Enterprise PL/I algorithms that pad data items in order to align them on byte, fullword, or doubleword boundaries are not implemented. One of the following two techniques can be used to address this restriction:
 1. Modify the Enterprise PL/I application and specify the UNALIGNED keyword on each 01-level structure that represents the interface to the application. Ensure that you remove any ALIGNED keywords from members of the 01-level structures. Recompile and link the application.
 2. Insert padding items into structure in order to cause data items that require byte, fullword, or doubleword alignment to be aligned correctly. Refer to the Enterprise PL/I for z/OS documentation for more information on PL/I data item alignment characteristics.

Related information

[“Defining Date fields” on page 503](#)

[Data alignment for Enterprise PL/I for z/OS, Version 5.3](#)

Service-level data conversion customization

For each service, you can specify to exclude special, non-printable control characters or suppress low values or empty tags from the JSON response messages. You can also specify whether and how to initialize fields in the input message.

These options can be configured per service by clicking **Advanced Options** in the service project editor.

Note:

Service-level data conversion occurs after input message fields are read. During runtime, input message field data is determined by using following order of precedence:

1. The JSON response value.
2. The default value from API mapping, which is specified in the API toolkit API editor.
3. The default value from service definition, which is specified in the API toolkit service interface editor.
4. Values that are specified through Service-level data conversion customization options.

Table 59. Input options

Option	Description
Initialize fields	<p>Select this option to initialize fields in the input data structure if a default value is not specified and either the field is omitted from the input interface, or the field is included but the respective JSON tag is not received at run time. When this option is selected, the fields are initialized based on the data type:</p> <ul style="list-style-type: none">Non-numeric data fields are initialized with spaces.Numeric data fields are initialized with zeros. Numeric fields with USAGE DISPLAY are initialized as EBCDIC representation of zeros.Composite fields are initialized with low values. <p>By default, fields are not initialized.</p>
Enforce minimum array occurrence	<p>Enforces the minimum number of array occurrence in the input data structure as defined in the copybook. By default, minimum array occurrences are enforced.</p>
Imported IMS large data structure starts with LLZZTRANCODE fields	<p>When disabled, the imported request IMS large data structure does not declare leading LL, ZZ, TRANCODE fields and therefore generated LL, ZZ, TRANCODE fields are used when creating the first message segment to send to IMS. This is the default option.</p> <p>When enabled, the imported request IMS large data structure declares leading LL, ZZ, TRANCODE fields and therefore the existing LL, ZZ, TRANCODE fields are used when creating the first message segment to send to IMS.</p> <p>This option applies only to request service interfaces in IMS Service and IMS LDS projects. Whether enabled or disabled, the values of LL and TRANCODE are set by z/OS Connect EE.</p> <p>Note:</p> <p>If the structure does declare these leading fields, and this option is disabled, request data will appear shifted by up to 12 bytes to the right.</p> <p>If the structure does not declare these leading fields, and this option is enabled, request data will appear shifted by up to 12 bytes to the left.</p> <p>This option is provided by the IMS large data structure support capability. To learn more, see “Capabilities compatibility” on page 751.</p>

Table 60. Output options

Option	Description
Trim leading whitespace	Select this option to trim leading whitespace from JSON property values in the output messages. By default, leading whitespace is not trimmed.
Trim trailing whitespace	Select this option to trim trailing whitespace from JSON property values in output messages. By default, trailing whitespace is trimmed.

Table 60. Output options (continued)

Option	Description
Escape control characters	Select this option to escape non-printable control characters, such as tokens or control blocks, in JSON property values as \uNNNN for necessary internal processing, instead of removing them. By default, control characters are omitted, not escaped. For the list of machine-dependent control characters, see “Special characters removed in JSON responses by default” on page 540 .
Omit fields with uniform byte values	Select this option to omit the JSON name-value pair for a non-numeric field from the JSON output message when the data for the field is composed of the same byte value repeated throughout, such as all 0x00 or all 0xFF. By default, this option is not selected. Specify the hexadecimal value that all bytes in a non-numeric field must contain to be omitted in the Value to use for field omission field. The default value is 00.
Omit empty fields	Select this option to omit JSON tags that contain an empty string ("tag": "") from JSON output messages after whitespace and control characters are processed. By default, empty tags are not omitted.
Enforce minimum array occurrence	Enforces the minimum number of array occurrence in the output data structure as defined in the copybook. By default, minimum array occurrences are enforced.
Imported IMS large data structure starts with LLZZ fields	When disabled , the imported response IMS large data structure does not declare leading LL, ZZ fields and therefore the response data is assumed to start at the first byte of the DATA portion of the first message segment received from IMS. This is the default option. When enabled , the imported response IMS large data structure declares leading LL, ZZ fields and therefore the response data is assumed to start at the fifth byte of the DATA portion of the first message segment received from IMS. This option applies only to response service interfaces in IMS Service and IMS LDS projects.
Note:	
If the structure does declare these leading fields, and this option is disabled, response data will appear shifted 4 bytes to the right.	
If the structure does not declare these leading fields, and this option enabled, response data may appear to be shifted 4 bytes to the left. The LL field in the response data structure is not interpreted as the length of the segment.	
This option is provided by the IMS large data structure support capability. To learn more, see “Capabilities compatibility” on page 751 .	

Special characters removed in JSON responses by default

Special, non-printable control characters are removed from the JSON response tag values unless the Escape control characters option is selected.

Table 61. Non-printable control characters that are removed by default			
Special characters removed by default			
0x0000 NUL	0x0001 SOH	0x0002 STX	0x0003 ETX
0x0004 EOT	0x0005 ENQ	0x0006 ACK	0x0007 BEL
0x0008 BS	0x000B VT	0x000C FF	0x000E SO
0x000F SI	0x0010 DLE	0x0011 DC1	0x0012 DC2
0x0013 DC3	0x0014 DC4	0x0015 NAK	0x0016 SYN
0x0017 ETB	0x0018 CAN	0x0019 EM	0x001A SUB
0x001B ESC	0x001C FS	0x001D GS	0x001E RS
0x001F US	0x007F DEL	0x0080 XXX	0x0081 XXX
0x0082 BPH	0x0083 NBH	0x0084 IND	0x0086 SSA
0x0087 ESA	0x0088 HTS	0x0088 HTS	0x0089 HTJ
0x008A VTS	0x008B PLD	0x008C PLU	0x008D RI
0x008E SS2	0x008F SS3	0x0090 DCS	0x0091 PU1
0x0092 PU2	0x0093 STS	0x0094 CCH	0x0095 MW
0x0096 SPA	0x0097 EPA	0x0098 SOS	0x0099 XXX
0x009A SCI	0x009B CSI	0x009C ST	0x009D OSC
0x009E PM	0x009F APC		

You can specify to escape rather than remove special characters as you create or edit a service:

1. In the service project editor, click **Advanced Options**
2. Select the Escape control characters check box.
3. Click **Apply** and then **OK**.

Special characters are now escaped as \uNNNN.

Handling large IMS data structures

IMS message segments have a size limit of 32 KB (32,767 bytes). If your application programs contain large data structures, the transaction must handle the decomposition of the large data structure into multiple segments, each within the 32 KB limit and then assemble multiple segments on the response.

IMS transactions that are defined as MSGTYPE=(MULTSEG) are eligible to handle large IMS data structures. In existing IMS transactions, *segmentation* is most likely already implemented by GU and GN calls to process request messages that are larger than 32 KB. *Segmentation* refers to the process of decomposing a large data structure into multiple segments so each IMS message segment is within the 32 KB restriction. For response messages, *de-segmentation* is most likely implemented by multiple ISRT calls. The term *de-segmentation* refers to the process of assembling multiple segments for an IMS transaction message that contains a large data structure.

If your transaction already handles *segmentation* and *de-segmentation* of large data structures, no change is needed. If it does not, sample utilities are provided that your application can invoke instead of interacting with the IMS message queue directly. The sample utilities, BAQGETS and BAQSETS for COBOL,

and BAQPGETS and BAQPSETS for PL/I, handle the *segmentation* and *de-segmentation* of large data structures by using a temporary storage in the message processing region.

To create a service from an IMS program with large data structures, in z/OS Connect EE API toolkit, select **IMS Large Data Structure Service** as the project type. If you select **IMS Service** as the service type, the entire data structure would be imported into the first segment.

Important: By default, imported IMS large data structures are not expected to begin with an LLZZ or LLZZ<TRANCODE> prefix fields.

- If large data structures do begin with LLZZ or LLZZ<TRANCODE> prefix fields, you must enable options in the **Advanced Service Data Conversion Options** dialog menu to support these fields. To open this dialog, go to the **Definition** tab of the Service Project Editor and click **Advanced Options**. To learn more about these options, see “[Service-level data conversion customization](#)” on page 537.
- The advanced data conversion options allowing IMS large data structure services to support LLZZTRANCODE fields requires installation of the minimum z/OS Connect EE server and API toolkit versions that support this capability. To learn more about z/OS Connect EE version requirements for the IMS large data structure support capabilities, see “[Capabilities compatibility](#)” on page 751.

The following diagram demonstrates how a large IMS data structure in COBOL programs is processed. For PL/I, the process is the same except for the utility and sample names are different.

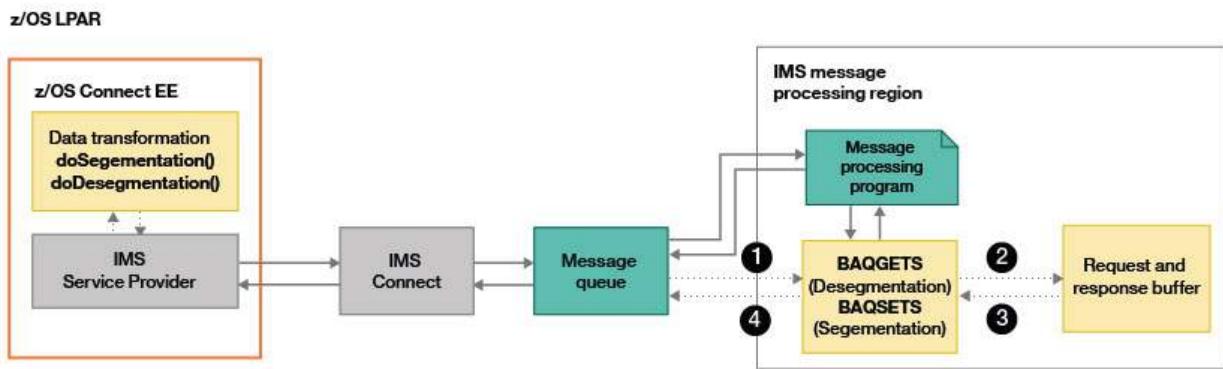


Figure 107. Handling large IMS data structure with the COBOL BAQGETS and BAQSETS utilities

To leverage these sample utilities, your IMS COBOL applications would invoke the supplied sample utilities BAQGETS and BAQSETS instead of interacting with the IMS message queue directly. IMS PL/I applications would invoke the supplied sample utilities BAQPGETS and BAQPSETS instead of interacting with the IMS message queue directly.

1. BAQGETS/BAQPGETS performs de-segmentation by reading the large data structure from the IMS message queue.
2. BAQGETS/BAQPGETS writes the de-segmented request structure to the temporary storage.
3. BAQSETS/BAQPSETS performs segmentation by reading from the storage.
4. BAQSETS/BAQPSETS writes the segmented response structure to the IMS message queue

To demonstrate detailed usage of these utilities, a sample COBOL IMS transaction BAQLDS and a sample PL/I IMS transaction BAQPLDS are provided. These utilities and the samples are provided in the SBAQSAMP data set. Two JSON documents for use with the samples are available in the <installation_path>/samples/imslds directory. Use a text editor to view the sample JSON documents.

- `requestBody.json` is the request JSON to use to invoke the service that you create from the supplied BAQLDSC and BAQPLDSI copybooks. The copybooks contain comments that describe the data structure to select for input and output when you create the transaction metadata for your service in the z/OS Connect EE API toolkit. Increase or decrease the number of `IO_RECORD` array objects to change the size of the data structure sent to and received from IMS.

- `responseBody.json` is the expected response JSON from your service that invokes the BAQLDS and BAQPLDS sample programs. The property `I_TEXT_OK` in the response indicates whether the JSON request body was converted correctly ("Y") or not ("N") based on what BAQLDS or BAQPLDS expects.

Important: When the structure's maximum length does not exceed 32 KB, do not use the BAQGETS, BAQSETS, BAQPGETS, or BAQPSETS sample utility. Use CBLTDLI or CEETDLI instead.

The following table lists the COBOL samples and utilities that are provided in the SBAQSAMP data set.

<i>Table 62. COBOL samples and utilities for large IMS data structure support in the SBAQSAMP data set</i>		
Member name	Source type	Description
BAQLDS	COBOL	Sample IMS transaction that demonstrates usage of the BAQGETS and BAQSETS utilities. This transaction must be defined to IMS with MSGTYPE(MULTSEG).
BAQLDSC	COBOL	Copybook for the BAQLDS sample transaction. Import this copybook into the service interface editor in z/OS Connect EE API toolkit.
BAQGETS	COBOL	Sample utility that reads a large data structure from the IMS message queue and performs de-segmentation by getting each segment using CBLTDLI.
BAQSETS	COBOL	Sample utility that writes a large data structure to the IMS message queue by performing segmentation and inserting each segment using CBLTDLI.
BAQGETSJ	JCL	Sample JCL for building BAQGETS.
BAQSETSJ	JCL	Sample JCL for building BAQSETS.
BAQLDSJ	JCL	Sample JCL for building BAQLDS. BAQGETS and BAQSETS must be built before running this JCL to build the BAQLDS sample.

The following table lists the PL/I samples and utilities that are provided in the SBAQSAMP data set.

<i>Table 63. PL/I samples and utilities for large IMS data structure support in the SBAQSAMP data set</i>		
Member name	Source type	Description
BAQPLDS	PL/I	Sample IMS transaction that demonstrates usage of the BAQPGETS and BAQPSETS utilities. This transaction must be defined to IMS with MSGTYPE(MULTSEG).
BAQPLDSI	PL/I	Include for the BAQPLDS sample transaction. Import this copybook into the service interface editor in the z/OS Connect EE API toolkit.
BAQPGETS	PL/I	Sample utility that reads a large data structure from the IMS message queue and performs de-segmentation by getting each segment using CBLTDLI.
BAQPSETS	PL/I	Sample utility that writes a large data structure to the IMS message queue by performing segmentation and inserting each segment using CBLTDLI.
BAQPGETJ	JCL	Sample JCL for building BAQPGETS.
BAQPSETJ	JCL	Sample JCL for building BAQPSETS.

Table 63. PL/I samples and utilities for large IMS data structure support in the SBAQSAMP data set (continued)

Member name	Source type	Description
BAQPLDSJ	JCL	Sample JCL for building BAQPLDS. BAQPGETS and BAQPSETS must be built before running this JCL to build the BAQPLDS sample.

The following table lists the content of the <installation_path>/samples/imslds directory.

Table 64. Samples for large IMS data structure support in the <installation_path>/samples/imslds directory

Member name	Source type	Description
requestBody.json	JSON	Sample request JSON body for services based on the BAQLDS and BAQPLDS sample IMS transaction programs. Increase or decrease the number of IO_RECORD array objects to change the size of the data structure sent to and received from IMS.
responseBody.json	JSON	Sample response JSON body for services based on the BAQLDS and BAQPLDS sample IMS transaction programs.

Creating an IMS database service

Using the API toolkit, you can create an IMS database service by specifying IMS database connection information and a prepared SQL query.

Before you begin

- An IMS database service connection profile must be configured in the `server.xml` file. See “Configuring for the IMS database service provider” on page 263.
- A host connection to an IMS database must be configured, if you do not have host connections configured, see “Creating an IMS database host connection” on page 265. This connection is used during service creation to connect to IMS and collect metadata from the IMS catalog.

Procedure

To create an IMS database service:

- Open z/OS Explorer and switch to the **z/OS Connect Enterprise Edition** perspective:
 - From the main menu, select **Window > Open Perspective > Other**. The Select Perspective wizard opens.
 - Select **z/OS Connect Enterprise Edition**.
- Select **File > New > Project**.
The New Project wizard opens.
- Select **z/OS Connect Enterprise Edition > z/OS Connect EE Service Project**, and click **Next**.
- Specify a project name, and select **IMS database Service** as the project type. You may optionally provide a description for the service.
- Click **Finish** to create the project.

The service project is created in the **Project Explorer** view. The `service.properties` file opens in the service project editor in a tab that is named after the service project. The service project editor is where you can configure the service and define the service interface. Initially, errors (X) are reported and highlighted for information that is required and must be specified.

The **Actions** pane highlights the steps to create a service.

6. In the service project editor, take the following steps:
 - a) Optionally, change the version number from the default of 1.0.0.
 - b) In the **SQL Command** field, copy and paste a valid SQL query. Alternatively, you can import a file with a .sql extension that contains an SQL query. Only a single select query for each service is currently supported.

Note: SQL queries should be prepared and tested against your IMS database before they are used with the IMS database service provider. This can be done within IMS Explorer, see [Creating and running SQL queries against an IMS database](#) to learn more.
 - c) For the **Database Connection** field, click the drill down menu and select an IMS database connection you have configured in your host connections. If your IMS database connection is not listed, ensure that you have [connected to an IMS database](#).
 - d) For the **Database Name** field, specify the IMS PSB you want to use for your service.
 - e) Click the **Generate service interface** button. The API toolkit establishes a connection with IMS using the connection profile specified in step c, and collects database metadata from IMS catalog.

Note: If you do not click **Generate service interface**, or if any changes are made to your IMS database service definition, an error marker (X) is placed next to the button. You must generate the service interface before you save your IMS database service.
 - f) Save your changes using **File > Save**, or **Ctrl+S**.

The IMS database service is defined. We will now configure the connection profile that is used to connect to the IMS database when an IMS database service is invoked.

7. Click the **Configuration** tab for your service.
8. Specify the ID of the connection profile name in the **Connection profile** field.

The connection profile name is the ID of the connection factory created in step 6 of “[Configuring for the IMS database service provider](#)” on page 263.

Note: The connection profile name is used by an IMS database service at runtime to connect to IMS database.
9. Save your changes by pressing **Ctrl+S**.

Results

Your IMS database service is complete, and ready to [deploy to a z/OS Connect EE server](#).

Note: After validation that all required information is specified, JSON schema files and the service XML file are created in the service project folder. To edit the service project at a later time, open the service project editor by double-clicking the `service.properties` file in the project folder within the **Project Explorer** view.

To perform a test of the service, from a REST client, issue an HTTP GET request that uses the following nomenclature: `http://<serverName>:<port>/zosConnect/services/<serviceName>`

where `<serverName>` and `<port>` is server name and port combination of your z/OS Connect EE server, and `<serviceName>` is the service name you defined in step 4.

If your service is working, you will see a response body similar to the following:

```
{
  "zosConnect": {
    "serviceName": "<yourService>",
    "serviceDescription": "",
    "serviceProvider": "DBSP-1.0",
    "serviceURL": "https://host:port/zosConnect/services/<yourService>",
    "serviceInvokeURL": "https://host:port/zosConnect/services/<yourService>?action=invoke",
    "dataXformProvider": "jsonByte-1.0",
    "serviceStatus": "Started"
  },
  ...
}
```

To execute the service, from a REST client, issue an HTTP POST request that uses the following nomenclature: `http://<serverName>:<port>/zosConnect/services/<serviceName>?action=invoke`

Provide an input payload containing appropriate values for all input parameters in your SQL query:

```
{  
  "request":{  
    "parameter1": "<value1>",  
    "parameter2": "<value2>",  
    ...  
    "parameterN": "<valueN>"  
  }  
}
```

Example

The following examples demonstrate how various input payloads for an IMS database service is defined by the query that is used to create the service.

Note: All of the following examples assume that you have already deployed your IMS database service to a z/OS Connect EE server.

In this example, an IMS database service was created using the following SQL query:

```
SELECT PATNAME FROM PCB01.PATIENT, PCB01.WARD WHERE WARDNO=? AND PATLL=?
```

Note that WARDNO and PATLL constitute the input parameters of the input payload used for the service. Consequentially, the following example JSON input payload will be used in a REST client to execute the service:

```
{  
  "request":{  
    "WARDNO": "0004",  
    "PATLL": 900  
  }  
}
```

In this example, the IMS database service issues the following response to the REST client:

```
{  
  "response": {  
    "result": [  
      {  
        "PATLL": 900,  
        "PATNUM": "0001",  
        "PATNAME": "BECCA KERRY"  
      },  
      {  
        "PATLL": 900,  
        "PATNUM": "0002",  
        "PATNAME": "ROB MILLER"  
      },  
      {  
        "PATLL": 900,  
        "PATNUM": "0003",  
        "PATNAME": "MARCUS QUERALES"  
      }  
    ]  
  }  
}
```

The following example demonstrates a different input payload and SQL query for an IMS database. In this example, the `server.xml` for this service uses the **flattentables** attribute, which produces flattened ARRAY field in the service response.

In this example, an IMS database service was created using the following SQL query:

```
SELECT ADDRESS_1_CITY, ADDRESS_1_STATE, ADDRESS_1_ZIP FROM PCB01.ARRAY WHERE ADDRESS_2_LAST_NAME=?
```

The following JSON input payload is used to execute the service:

```
{  
    "request": {  
        "ADDRESS_2_LAST_NAME": "TRAN"  
    }  
}
```

In this example, the IMS database service issues the following response to the REST client:

```
{  
    "response": {  
        "result": [  
            {  
                "ADDRESS_1_STATE": "CA",  
                "ADDRESS_1_ZIP": "95141",  
                "ADDRESS_1_CITY": "San Jose"  
            }  
        ]  
    }  
}
```

In this final example, the `server.xml` for this service also uses the **flattentables** attribute, which produces flattened STRUCT field in the service response.

In this example, an IMS database service was created using the following SQL query:

```
SELECT ACCOUNTS_1_TYPE, ACCOUNTS_2_TYPE, ACCOUNTS_3_TYPE, ACCOUNTS_4_TYPE,  
ACCOUNTS_5_TYPE FROM PCB01.STRUCT WHERE FIRST_NAME=?
```

The following JSON input payload is used to execute the service:

```
{  
    "request": {  
        "FIRST_NAME": "KEVIN"  
    }  
}
```

In this example, the IMS database service issues the following response to the REST client:

```
{  
    "response": {  
        "result": [  
            {  
                "ACCOUNTS_1_TYPE": "VISA",  
                "ACCOUNTS_3_TYPE": "VISA",  
                "ACCOUNTS_2_TYPE": "VISA",  
                "ACCOUNTS_5_TYPE": "VISA",  
                "ACCOUNTS_4_TYPE": "VISA"  
            }  
        ]  
    }  
}
```

For a more thorough example of creating an IMS database service, see the Quick Start Scenario “[Prepare the sample IMS database](#)” on page 62.

What to do next

Instructions for starting, stopping, and removing IMS database services are the same as for IMS and CICS. To read more, see “[Starting, stopping, or removing a service](#)” on page 529.

If you need to troubleshoot issues caused by an IMS database service, see “[Enabling trace in z/OS Connect EE server](#)” on page 715.

Related tasks

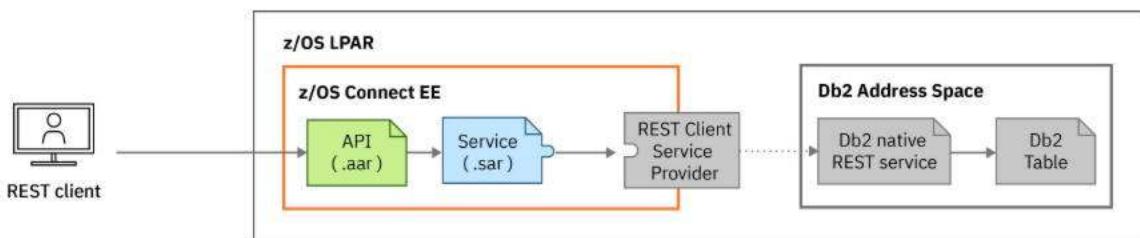
[“Deploying a service” on page 529](#)

Deploy your service to the server directly from the API toolkit if the server connection is already properly configured.

Db2 services

Use the z/OS Connect EE API toolkit to create a z/OS Connect EE service.

The z/OS Connect EE API toolkit helps you create a Db2 service project where you define the request and response JSON schemas, and the Db2 native REST service which z/OS Connect EE interacts with. The Db2 native REST service is created in Db2 using the Db2 REST services function. The z/OS Connect Db2 service uses the REST client service provider to process requests to Db2.



Pre-requisites: You must apply the PTF for Db2 APAR PI98649: New function update of Db2 native RESTful services support.

After a service is created, you can deploy it directly from the z/OS Connect EE API toolkit. The **zosconnect_services** element must be present in the `server.xml` configuration file for service deployment using .sar files.

Note:

Deploying a service directly from within the API toolkit requires server code V3.0.4 or later.

You can also export the service project as a service archive (.sar) file. For more information about deploying the generated service archive file, see [“Automated service archive management” on page 670](#).

Related information

[Db2 REST services](#)

Creating a Db2 service project

Create a Db2 service project with the z/OS Connect EE API toolkit.

Before you begin

Switch to the **z/OS Connect Enterprise Edition** perspective in your Eclipse environment.

1. From the main menu, select **Window > Open Perspective > Other**. The Select Perspective wizard opens.
2. Select **z/OS Connect Enterprise Edition**.

About this task

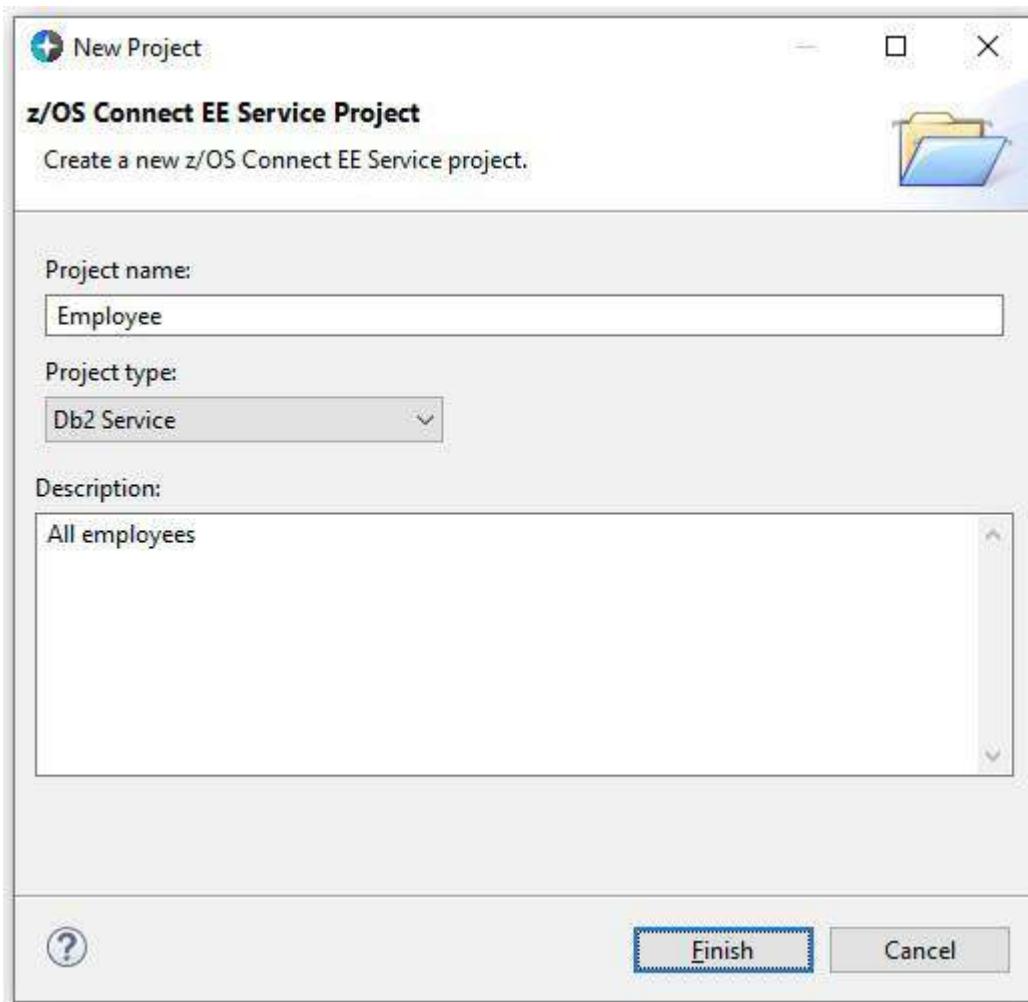
Create a Db2 service project.

Procedure

1. Select **File > New > Project**.

The New Project wizard opens.

2. Select **z/OS Connect Enterprise Edition** > **z/OS Connect EE Service Project**, and click **Next**.
3. Enter a name of your choice for the project. This name is also used as the service name.
4. Select **Db2 Service** from the **Project Type** list.



5. Enter a description and click **Finish**.
6. Optionally, change the version number from the default of 1.0.0.

Results

The service project is created in the **Project Explorer** view and the `service.properties` file opens in the service project editor in a tab that is named after the service project. This service project editor is where you configure the service and define the Db2 native REST service details. Initially, errors (✖) are reported and highlighted for information that is required and must be specified.

What to do next

Import a Db2 native REST service from Db2 service manager. See “[Importing a Db2 native REST service](#)” on page 549.

Alternatively, you can define a service by importing the schemas from your local machine and manually specifying the Db2 native REST service specifications. See “[Importing JSON schemas from your local machine](#)” on page 551

Importing a Db2 native REST service

Import Db2 native REST service details from the Db2 service manager.

Before you begin

Complete the steps in “[Creating a Db2 service project](#)” on page 547.

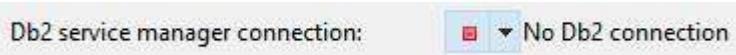
1. From the main menu, select **Window > Open Perspective > Other**. The Select Perspective wizard opens.
2. Select **z/OS Connect Enterprise Edition**.
3. Open the `service.properties` file in the service project editor and optionally, change the version number from the default of 1.0.0..

About this task

Define the Db2 service by importing a Db2 native REST service from the Db2 service manager.

Procedure

1. Click **Import from Db2 service manager...** to open the import wizard.
2. Click the **connection widget** to select which Db2 service manager connection to use for importing.



- a) If the connection is not already defined, select **New Db2 Service Manager Connection...**
- b) Specify the following information:

Table 65. Adding a Db2 server connection

Field	Description
Name	A descriptive name for the Db2 server connection.
Host name	The name or the IP address of the Db2 server.
Port number	The port number for the Db2 service manager. Select the Secure connection (TLS/SSL) checkbox for secure connections.
Connection timeout	The amount of time in milliseconds the API toolkit waits for a successful connection to be established with the Db2 server before timing out. The default is 30 seconds unless specified otherwise in the z/OS Connect EE preferences window. A value of 0 indicates to wait forever.
Read timeout	The amount of time in milliseconds the API editor reads response data from the Db2 server before timing out. The default is 30 seconds unless specified otherwise in the z/OS Connect EE preferences window. A value of 0 indicates to wait forever.

Tip: You can change the default connection timeout and read timeout values. From the main menu bar, click **Window > Preferences... > z/OS Connect EE**, and specify your default timeout values.

- c) Click **Save and Connect** to save the definition and connect to the server.

You are prompted to either specify an existing credential, or create a new credential. To create a credential, click **Add** in the **Credentials** section. For more information, see [Defining connection credentials](#) in the *IBM Explorer for z/OS* documentation.

Note: If client authentication is enabled on the server, this user credential is used for authorization. A trusted client certificate must be configured for user authentication. For more information, see “[Configuring client certificates for server connections](#)” on page 604.

- d) After you select an existing credential or create a new one, click **OK** to connect with that credential. The table shows all the Db2 native REST services that are returned by Db2.

Service Name	Collection ID	Description	URI
callSPsqlTables	SYSIBMSERVICE	Call common SQLTABLES SP, which takes inputs for...	/services/SYSIBMSERVICE/callSPsqlTables
firstNameJames	SYSIBMSERVICE	Gets users with the given first name of James. SELE...	/services/SYSIBMSERVICE/firstNameJames
firstNameJames1	SYSIBMSERVICE	Gets users with the given first name of James. SELE...	/services/SYSIBMSERVICE/firstNameJames1
firstNameJames10	SYSIBMSERVICE	Gets users with the given first name of James. SELE...	/services/SYSIBMSERVICE/firstNameJames10
firstNameJames11	SYSIBMSERVICE	Gets users with the given first name of James. SELE...	/services/SYSIBMSERVICE/firstNameJames11
firstNameJames12	SYSIBMSERVICE	Gets users with the given first name of James. SELE...	/services/SYSIBMSERVICE/firstNameJames12
firstNameJames13	SYSIBMSERVICE	Gets users with the given first name of James. SELE...	/services/SYSIBMSERVICE/firstNameJames13
firstNameJames14	SYSIBMSERVICE	Gets users with the given first name of James. SELE...	/services/SYSIBMSERVICE/firstNameJames14
firstNameJames2	SYSIBMSERVICE	Gets users with the given first name of James. SELE...	/services/SYSIBMSERVICE/firstNameJames2
firstNameJames3	SYSIBMSERVICE	Gets users with the given first name of James. SELE...	/services/SYSIBMSERVICE/firstNameJames3
firstNameJames4	SYSIBMSERVICE	Gets users with the given first name of James. SELE...	/services/SYSIBMSERVICE/firstNameJames4
firstNameJames5	SYSIBMSERVICE	Gets users with the given first name of James. SELE...	/services/SYSIBMSERVICE/firstNameJames5
firstNameJames6	SYSIBMSERVICE	Gets users with the given first name of James. SELE...	/services/SYSIBMSERVICE/firstNameJames6
firstNameJames7	SYSIBMSERVICE	Gets users with the given first name of James. SELE...	/services/SYSIBMSERVICE/firstNameJames7
firstNameJames8	SYSIBMSERVICE	Gets users with the given first name of James. SELE...	/services/SYSIBMSERVICE/firstNameJames8

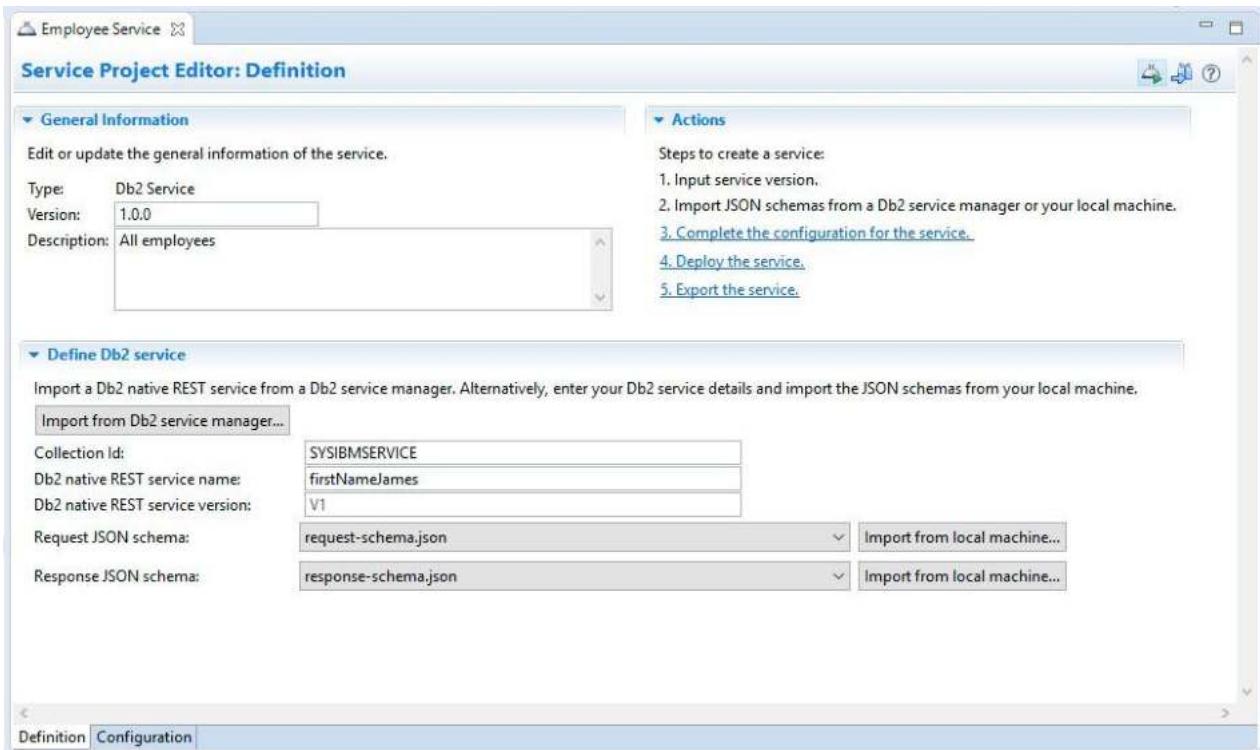
3. Select the Db2 native REST service which will be exposed by z/OS Connect EE. If needed, the search box can be used to filter the listed services.
4. Click **Import**.
5. Save your changes.

Results

The Db2 native REST service name, version, and collection ID are set to the value from Db2 and the request and response JSON schemas are specified for the service and files that are added to the project.

Note:

- The collection ID value can be overridden by the policy setting or the `zosconnect_services` setting in `server.xml`. For more information, see “[Overriding Db2 service properties in the server configuration](#)” on page 286.
- A collection ID with a value that contains the # character is not supported.



What to do next

Follow the steps in “Configuring the connection reference to Db2” on page 552.

Related information

[Db2 REST services](#)

Importing JSON schemas from your local machine

Create a Db2 service from a JSON schemas that are stored on your local machine.

Before you begin

Complete the steps in “Creating a Db2 service project” on page 547.

1. From the main menu, select **Window > Open Perspective > Other**. The Select Perspective wizard opens.
2. Select **z/OS Connect Enterprise Edition**.
3. Open the service.properties file in the service project editor and optionally, change the version number from the default of 1.0.0..

About this task

Import the JSON schema to create a service that connects z/OS Connect EE to a Db2 system.

Procedure

1. Enter the collection ID, service name, and version for the Db2 native REST service.

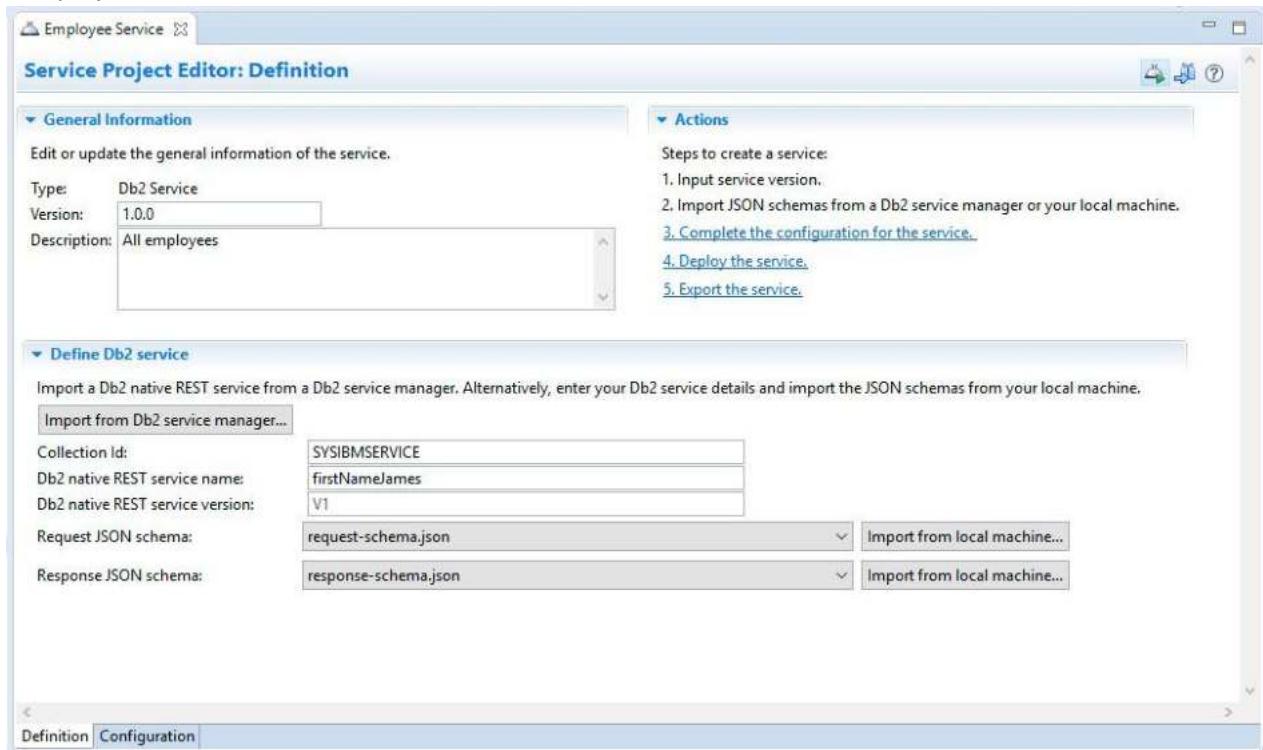
Note:

- The collection ID value can be overridden by the policy setting or the `zosconnect_services` setting in `server.xml`. For more information, see “[Overriding Db2 service properties in the server configuration](#)” on page 286.
- A collection ID with a value that contains the # character is not supported.

- For the Request JSON schema, click **Import from local machine** and select the schema from your local machine.
- For the Response JSON schema, click **Import from local machine** and select the schema from your local machine.

Results

The project now contains the details and schemas for the Db2 native REST service.



What to do next

[“Configuring the connection reference to Db2” on page 552.](#)

Configuring the connection reference to Db2

Specify the connection reference used by the z/OS Connect EE server.

Before you begin

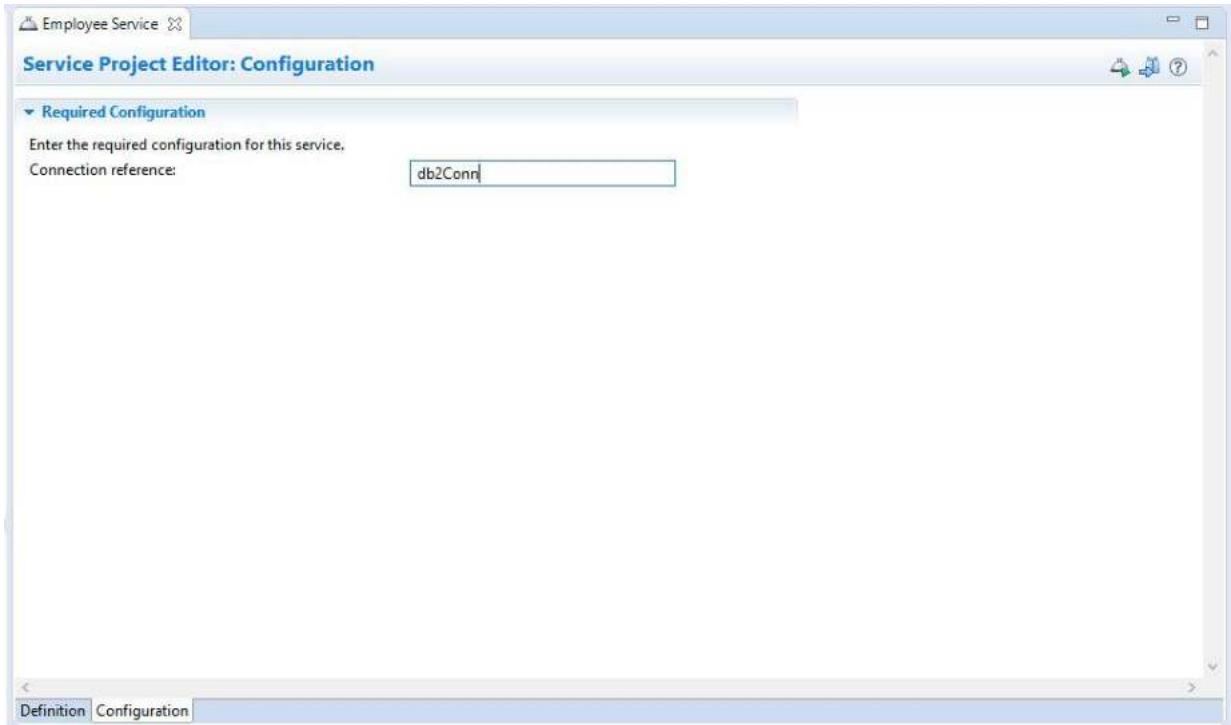
Complete the steps in either [“Importing a Db2 native REST service” on page 549](#) or [“Importing JSON schemas from your local machine” on page 551](#)

About this task

Complete the configuration by specifying the connection reference used by the z/OS Connect EE server.

Procedure

- In the project dialog, click the **Configuration** tab.



2. Enter the **Connection reference**. This is the id of the `zosconnect_zosConnectServiceRestClientConnection` element in `server.xml`, which defines the connection to Db2 that is used at run time.

Note: The value of `connectionRef` defined in the `server.xml` file overrides the value that you defined for **Connection reference**. For example:

```
<zosconnect_services>
  <service name="exampleDb2" >
    <property name="connectionRef" value="newValue" />
  </service>
</zosconnect_services>
```

Results

The connection reference is defined and the project is now complete.

What to do next

[“Deploying a Db2 service” on page 553.](#)

Related tasks

[Configuring a REST client connection in z/OS Connect EE](#)

Follow these steps to configure a REST client connection to an HTTP endpoint.

Deploying a Db2 service

Deploy your service to the server directly from the API toolkit if the server connection is already properly configured.

Before you begin

Deploying a service directly from within the API toolkit requires server code V3.0.4 or later.

You must create a host connection to the z/OS Connect EE server. For more information, see [“Connecting to a z/OS Connect EE server” on page 603.](#)

About this task

If the service project is open, you can deploy a service directly by clicking the **Deploy Service to z/OS Connect EE Server** button () in the corner.

You can also deploy one or more services from the **Project Explorer** view.

Procedure

To deploy a service from the **Project Explorer** view:

1. In the **Project Explorer** view, select one or more service projects and right-click to select **z/OS Connect EE > Deploy Service to z/OS Connect EE Server**.
2. In the **Deploy Service** window, select the server to which to deploy the services.
3. If a service of the same name exists and you want to overwrite it, select the **Update existing services** check box.

Results

Deployment result is reported. Errors that occur during the deployment are recorded in the **Problems** view. A BAQR7070E error indicates that the **zosconnect_services** element is not configured in the **server.xml** configuration file.

A deployed service is automatically started unless specified otherwise in the z/OS Connect EE preferences window.

Tips:

- To get a refreshed view of all deployed services and their status on the server, in the **z/OS Connect EE Servers** view, right-click the **Services** folder, and select **Refresh**.
- To get a refreshed view of all deployed services and APIs and their status on a server, right-click the server itself and select **Refresh Server**.

What to do next

[Export your service project as a service archive \(a .sar file\)](#). This file is needed to create an API to invoke the service.

Related concepts

[“Setting preferences for the API toolkit” on page 606](#)

Starting, stopping, or removing a Db2 service

You can start, stop, or remove a service in the **z/OS Connect EE Servers** view.

Before you begin

Switch to the **z/OS Connect Enterprise Edition** perspective.

Ensure that you are connected to the server. For more information, see [“Connecting to a z/OS Connect EE server” on page 603](#).

About this task

The **z/OS Connect EE Servers** view in the **z/OS Connect Enterprise Edition** perspective provides a list of defined host connections and the services and APIs that are deployed on the servers.

Procedure

1. In the **z/OS Connect EE Servers** view, ensure that the server is connected. If the server is not connected, right-click the server and select **Connect**.
2. Expand the server and the **Services** folder to see all services that are deployed to the server.

3. For each service, you can examine its status, service URL, invoke URL, related service provider in the **Properties** view by right-clicking the service and select **Show Properties View**.
4. To start, stop, or remove a service, right-click the service and select **Start Service**, **Stop Service**, or **Remove Service**.

Related concepts

[“Setting preferences for the API toolkit” on page 606](#)

Exporting a Db2 service

To create an API or deploy a service through an automated DevOps process, you need a service archive (a .sar file). You can export your service project as a service archive to a location that you specify.

Procedure

1. In the **Project Explorer** view, right-click the service project and select **z/OS Connect EE > Export z/OS Connect EE Service Archive**.
2. Specify the location where you want to save this service archive file.

Results

The service is saved as a .sar file.

What to do next

- For API creation, see [“Creating a REST API” on page 589](#).
- In a DevOps environment, the service archive is generated by using the build toolkit with the service project directory as input. See [“Generating service archives for DevOps” on page 560](#).
- For automated service deployment, see [“Automated service archive management” on page 670](#).

Creating a REST client service

Create a service for the REST client by configuring the properties file that the build toolkit needs to generate the service archive.

To use the build toolkit to build your service archive for a REST client, the following properties must be defined.

Note: All JSON schema files must use UTF-8 encoding.

Table 66. Mandatory properties		
Property	Importance	Description
provider	Required	Must be set to <code>rest</code> to build a service archive file for the REST client service provider.
name	Required	The name of the service.
version	Required	The version of the service.

The following properties are available for the REST client service provider:

Table 67. Properties for the REST client service provider		
Property	Importance	Description
connectionRef	Required	The ID of the <code>zosconnect_zosConnectServiceRestClient</code> that defines the connection to the remote service.
description	Optional	The description of the service.

Table 67. Properties for the REST client service provider (continued)

Property	Importance	Description
requestSchemaFile	Required	The JSON schema file that defines the request object for this service. Note: If you do not specify an absolute path, the build toolkit uses a relative path from the directory in which the zconbt command is run.
responseSchemaFile	Required	The JSON schema file that defines the response object for this service. Note: If you do not specify an absolute path, the build toolkit uses a relative path from the directory in which the zconbt command is run.
verb	Optional	The HTTP verb that is used to call the service. If this verb is omitted, the verb that is used to invoke the service is used.
uri	Required	The resource that this service exposes. It must not contain the host name or port.

Sample properties file for the REST client

```
provider=rest
name=example
version=1.0
description>An example REST client service
requestSchemaFile=request.json
responseSchemaFile=response.json
verb=POST
uri=/resource/item
connectionRef=restConn
```

You can now use the build toolkit to generate the service archive. See “[Generating service archives for DevOps](#)” on page 560 for sample commands.

Related reference

[“zconbt command syntax” on page 795](#)

The **zconbt** command starts the build toolkit tool. You can use the build toolkit to generate archive files for services, APIs or API requesters.

Creating a WOLA service

Create a service for the WOLA service provider by configuring the properties file that the build toolkit needs to generate the service archive.

A service archive file built with the build toolkit contains equivalent function to a service archive file built using the **BAQLS2JS** utility and can also be deployed by copying the file to the services directory. For more information about the **BAQLS2JS** utility, see “[Conversion for z/OS Connect Enterprise Edition data transformation” on page 606](#). The WSBind file is included in the service archive file.

Define the following properties to build your service archive file with the build toolkit.

Note: All JSON schema files must use UTF-8 encoding. Language structure files must use the default code page for the system where the build toolkit is run.

Table 68. Common properties

Property	Importance	Description
provider	Required	Must be set to wola.
name	Required	The name of the service. For example, inquireSingle.
version	Required	The version of the service.
connectionRef	Required	The name of the configured connection factory element to be used. This element contains the JNDI name of the WOLA resource adapter connection factory.
connectionWaitTimout	Optional	The number of seconds to wait for an external address space application that matches the registration name to issue a WOLA Receive Request or Host Service API and become active. Set this parameter to 0 to disable timeout.
program	Required	The name of the program to invoke in the remote system.
register	Required	The name of the WOLA target register.
tranId	Optional	The name of the WOLA CICS Link Server link invocation task transaction ID.
useChannel	Optional	A boolean to specify if CICS channel and containers should be used. Additional properties are available when set to true, see Table 5 . The default is false.

The following properties are available for the WOLA service provider when used with DataXForm.

Table 69. Properties for the WOLA service provider using DataXForm

Property	Importance	Description
ccsid	Optional	The CCSID that is used at run time to encode character data. For example, 037.
charVarying	Optional	Defines how variable-length character data is mapped. Valid values are no null collapse binary. The default is null.
charOccurs	Optional	A CICS parameter that specifies how character arrays in the language structure are mapped. Valid values are string array. The default is string. Error messages with the prefix DFH might log issues with this property using the name CHAR-OCCURS.
charUsage	Optional	A CICS parameter that specifies how character arrays are encoded. Valid values are national dbcS. The default is national. Error messages with the prefix DFH might log issues with this property using the name CHAR-USAGE.

Table 69. Properties for the WOLA service provider using DataXForm (continued)

Property	Importance	Description
dataScreening	Optional	A CICS parameter that specifies whether application supplied data is screened for errors. Valid values are true false. The default is true.
dataTruncation	Optional	Controls if variable length data is tolerated in a fixed-length field structure. Valid values are false true. The default is false.
dateTime	Optional	Specifies if potential ABSTIME fields in the high-level language structure are mapped as timestamps. Valid values are unused packed15. The default is unused.
language	Required	The language of the target program. Valid values are COBOL C PLI-ENTERPRISE PLI-OTHER.
requestStructure	Required	The relative or absolute path to the file that contains the language structure for the request.
responseStructure	Required	The relative or absolute path to the file containing the language structure for the response.
truncateNullArrays	Optional	Specifies how structured arrays are processed. Valid values are false true. The default is false.
truncateNullArraysValue	Optional	Specifies which values are treated as empty for truncateNullArrays processing. Valid values are null space zero. The default is null.
structure	Optional	A CICS parameter that specifies the names of the high-level language structures that are contained in the partitioned data set members that are specified in the requestStructure and responseStructure parameters. This parameter is valid only for C and C++ languages, and COMMAREA.

The following properties are available for the WOLA service provider when used without DataXForm.

Table 70. Properties for the WOLA service provider without DataXForm

Property	Importance	Description
requestSchema	Required	The relative or absolute path to the JSON schema for the request.
responseSchema	Required	The relative or absolute path to the JSON schema for the response.

The following properties are available for the WOLA service provider when used for migration.

Table 71. Properties for the WOLA service provider when used for migration

Property	Importance	Description
bindFile	Required	The relative or absolute path to the bind file.
requestSchema	Required	The relative or absolute path to the JSON schema for the request.
responseSchema	Required	The relative or absolute path to the JSON schema for the response.

The following properties are available for the WOLA service provider when using channel and containers.

Table 72. Properties for the WOLA service provider when using channel and containers

Property	Importance	Description
channelName	Optional	The CICS channel name to use for delivering messages and receiving payloads using CICS containers. The default is IBM-WAS-ADAPTER.
requestContainerName	Optional	The name of the request container. The default is ZCONReqData.
requestContainerType	Required	The type of the request container, CHAR or BIT.
responseContainerName	Optional	The name of the response container. The default is ZCONRespData.
responseContainerType	Required	The type of the response container, CHAR or BIT,
useContextContainers	Optional	A boolean to specify if context containers should be used. The default is false.
contextEncoding	Optional	The encoding of the data in all context containers that are sent to the destination program. This property is only used when useContextContainers is set to true. The default is cp819.

Table 72. Properties for the WOLA service provider when using channel and containers (continued)

Property	Importance	Description
httpHeaders	Optional	The HTTP header name or list of comma-separated and case-sensitive HTTP header names that are passed to the destination program. This property is only used when useContextContainers is set to true.

Sample properties file for the WOLA service provider

```

provider=wola
name=service3
version=1.0
description>An example WOLA service
language=COBOL
program=MYPROG
register=ZAPPREG
requestStructure=request.cpy
responseStructure=response.cpy
connectionRef=wolaCf

```

You can now use the build toolkit to generate the service archive. See “[Generating service archives for DevOps](#)” on page 560 for sample commands.

Related reference

“[zconbt command syntax](#)” on page 795

The **zconbt** command starts the build toolkit tool. You can use the build toolkit to generate archive files for services, APIs or API requesters.

Generating service archives for DevOps

Use the build toolkit **zconbt** command to generate the service archive from service projects that are created in the API toolkit or from a properties file that defines the service.

Before you begin

Ensure that the z/OS Connect EE build toolkit is installed. For more information, see “[Installing the z/OS Connect EE build toolkit](#)” on page 199.

About this task

A service archive (.sar) file contains the information that is needed by a service provider to install and provide the service.

For CICS, IMS, and IBM MQ services that are created in the API toolkit, you can use the built-in service deployment and service archive export functions to test the service and create the API directly within the API toolkit. The service project directories can also be checked in to the source control management system so that they are automatically built and deployed as part of a DevOps pipeline. You can use the build toolkit to build the service projects into deployable archive files and use scripts to copy the files to the services directory.

For WOLA and REST client services, you must use the build toolkit to generate your service archives from properties files that define the services.

Db2 service projects can be built by the build toolkit only if it is V1.2 or later. If you run with an older, incompatible version of the build toolkit, one of the following error messages is displayed:

BAQB0007E: Error with property responseSchemaFile. Reason: Unable to read file schemas/response/response-schema.json

BAQB0007E: Error with property requestSchemaFile. Reason: Unable to read file schemas/request/request-schema.json

Procedure

1. Obtain the latest service source files from your source control management system and put them in a location that the **zconbt** command can access.
 - For CICS, IMS, and IBM MQ, service source files are the service project directories from the API toolkit.
 - For WOLA and REST client services, each service is defined in a properties file.
Note: The properties file and all JSON schema files must use UTF-8 encoding. The properties file must also use key value pairs with the format of key=value. For example,
requestSchemaFile=C:/Users/example/request schemas/request.json.
2. Use the **zconbt** command to generate the service archive.
 - For services that are created in the API toolkit, use the **-pd** parameter to specify the service project directory, and the **-f** parameter to specify the name for the service archive. For example,
zconbt --projectDirectory=./u/serviceProjects/catalog --file=./u/SARs/catalog.sar
 - **zconbt --projectDirectory=C:/serviceProjects/phonebook --file=C:/serviceProjects/SARs/phonebook.sar**
 - For WOLA or REST client services that are defined in a properties file, specify the properties file as the input, by using the **--properties** parameter.
zconbt --properties=service1.properties --file=./service1.sar

Note: If you are running on Microsoft Windows, use either a double backslash \\ or a forward slash character / in the path name.

Note: If you are running on z/OS, you can use the sample JCL provided in <hlq>.SBAQSAMP(BAQZCBT) to run the zconbt .zos command. You must customize the sample JCL following the instructions in BAQZCBT.

3. If any errors occur, make the necessary corrections and repeat the procedure.
4. You can now deploy the service archive file by copying it to the services directory.

Results

A service is automatically deployed when the service archive is copied into the services directory if the server is configured to monitor the services directory for changes.

What to do next

For more information about service archive management, see [“Automated service archive management” on page 670](#).

Related concepts

[“DevOps with z/OS Connect EE” on page 14](#)

Identify a DevOps process before you start your development to automate the development and deployment of services, APIs, and API requesters for continuous integration and delivery.

Related reference

[“zconbt command syntax” on page 795](#)

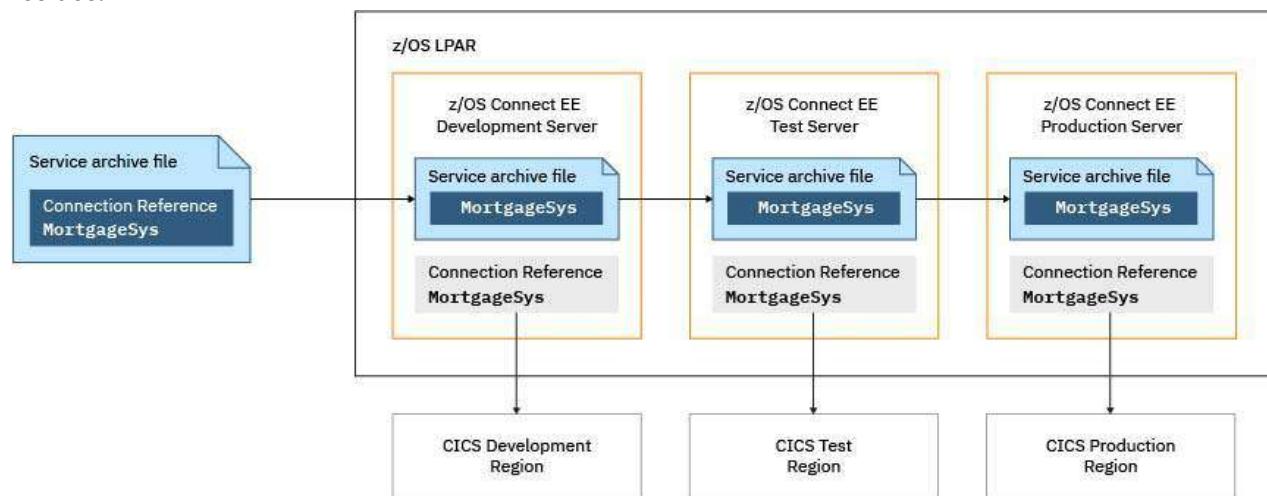
The **zconbt** command starts the build toolkit tool. You can use the build toolkit to generate archive files for services, APIs or API requesters.

How to move services between environments for DevOps.

You can move your service artifacts between environments such as Development, Test, and Production without changing the service archive (.sar) configuration files. This is an important consideration when you choose connection names for your Systems of Record (SoRs).

The attributes of a service are stored in the service archive file. Environment attributes are specified in the `server.xml` configuration file. This means that service artifacts can be moved from one environment to another without the need to regenerate the service. The service uses *connection by reference*, so if you have three environments, development, test, and production, you define a connection reference with the same ID in each environment. The connection reference defines the connection to the system of record for that environment. The service artifact is configured to use the ID of the connection reference. Because the ID is the same in each environment, the service artifact can be moved without regeneration from one environment to another even though each environment uses a different system of record.

For example, you have a set of mortgage services that targets various programs in a mortgage application. When you create a service, you specify the connection reference as `MortgageSys`. This defines a connection reference for the service to the connection element with ID `MortgageSys` in the `server.xml` configuration file of the z/OS Connect EE server in that environment. The `MortgageSys` connection element defines the connection to the system of record where the Mortgages application resides.



```
Development: "MortgageSys" defines a connection to DEV_SYS  
Test: "MortgageSys" defines a connection to RETAIL_TEST_SYS  
Production: "MortgageSys" defines a connection to MORTGAGE_PROD_SYS
```

If you add two more sets of services, one for Loans programs and another for Investments, you might use the connection references `LoansSys` and `InvestSys` respectively. The configuration in each server is then expanded to:

```
Development:  
  "MortgageSys" -> DEV_SYS  
  "LoansSys" -> DEV_SYS  
  "InvestSys" -> DEV_SYS  
Test:  
  "MortgageSys" -> RETAIL_TEST_SYS  
  "LoansSys" -> RETAIL_TEST_SYS  
  "InvestSys" -> INVESTMENT_TEST_SYS  
Production:  
  "MortgageSys" -> MORTGAGE_PROD_SYS  
  "LoansSys" -> LOANS_PROD_SYS  
  "InvestSys" -> INVESTMENT_PROD_SYS
```

In this configuration, all the services can move unchanged through the Development, Test, and Production environments because the same connection IDs are used in each environment to represent the logical endpoints. You might also connect all the services to the same system of record in Development, and in separate systems of record in Production.

Invoking a service

Use the HTTP method POST or PUT with the **action=invoke** query string to invoke a service.

The **action=invoke** query string runs the `invoke()` method of the service provider SPI implementation.

HTTP method

POST or PUT

URI

`/zosConnect/services/{serviceName}`

If the service name contains forward slashes, they must be escaped by using %2F. For example, if the service name is MyService/v1, it must be provided as MyService%2Fv1.

Description

Invoke a service.

Security

Users with Admin or Invoke authority can invoke a service. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Sample

The following sample runs the `invoke` method using the HTTP POST or PUT method for the service that is named `recordOpsCreate` and passes a JSON object payload in the request body.

```
https://host:port/zosConnect/services/recordOpsCreate?action=invoke
{
<JSON object passed in for the service invocation>
}
```

The following sample shows the JSON payload that is returned:

```
{
<JSON object returned from the service invocation>
}
```

The z/OS Connect EE `invoke` method supports an input payload in JSON object form for this request. In the code example, it is assumed that z/OS Connect EE looked up the service provider and identified a service reference for the service called SAMPLE-1.0. A data transformation reference is contained in the z/OS Connect EE service definition with a provider called `jsonByte-1.0`. When the `invoke` method of the service provider gets control and calls the `getBytes()` method, the data transformation implementation gets control and converts the request payload from JSON to a byte array and returns it to the service provider.

If the `invoke` action request has a URL that contains query parameters, these parameters, and other HTTP request information, are passed to the service provider by using the `com.ibm.zosconnect.spi.HttpZosConnectRequest` SPI interface that is provided by z/OS Connect EE. Interceptors that are processed for actions or operations also receive HTTP request information through the same object.

Another invocation style that is supported by z/OS Connect EE provides a means to define a user-defined URI as the `invokeURI` in the z/OS Connect service definition. With this style, the HTTP request does not have to contain `zosConnect/services` and can instead be a user-defined string.

When this style is employed, z/OS Connect EE supports use of other HTTP methods, such as GET, PUT, POST, and DELETE. Requests arriving with this URI, regardless of the HTTP method that is employed, pass through z/OS Connect EE interceptors and are passed to the `invoke()` method of the service provider.

Errors

When a service is stopped, the service will not invoke interceptors for new requests and the following message is returned:

412 Precondition failed

Creating APIs

z/OS Connect EE provides the tools you need to create APIs for your z/OS Connect EE services.

In the Eclipse environment, use the API Editor to create and edit APIs. For more information about using the API Editor, see

When you are in the **z/OS Connect Enterprise Edition** perspective in the Eclipse environment, you can do the following work:

- Create, edit, delete, deploy, and export an API project in the **Project Explorer** view.
- Browse, start, stop, and remove deployed APIs on connected servers in the **z/OS Connect EE Servers** view.
- Examine and test the operations of an API in Swagger UI that is included in the API toolkit.

For step-by-step examples of creating and deploying APIs, see

- [“Create an API to invoke the CICS catalog manager services ” on page 115](#)
- [“Create an API to invoke the IMS phone book service” on page 131](#)
- [“Create an API to invoke the IBM MQ stock query service” on page 149](#)
- [“Create a Db2 API to invoke the Db2 employee services” on page 154](#)

How to use the API editor

z/OS Connect Enterprise Edition provides an Eclipse-based API editor for you to design and create REST APIs for JSON services that are deployed on the z/OS Connect Enterprise Edition server.

The editor is GUI-based, so you can use the common keyboard and mouse-click actions for your API design needs, such as:

- **Drag-and-drop** to map incoming data from the HTTP headers, path parameter, or query parameters to fields in the service.
- **Right-click to access menus** for actions applicable to the selected elements, such as adding or editing a query parameter, assigning a value to a field, or undo a transform action.
- **Control or Shift keys to select multiple fields.**

For tutorials about how to create a REST/JSON service from various applications, and how to create a REST API to act on the service, see [Chapter 4, “Quick Start Scenarios,” on page 55](#)

Tip: To open the API editor for an existing API project, double-click the package.xml file in your project folder in the Project Explorer view.

Assume you have the following API project to allow your users to add a work contact.

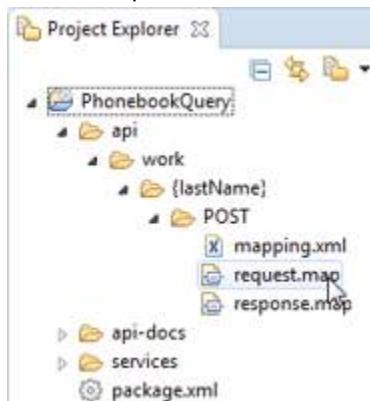
The screenshot shows the 'z/OS Connect EE API Editor' interface. In the 'Describe your API' section, the 'Name' field is set to 'contacts', 'Base path' to '/contacts', and 'Version' to '1.0.0'. The 'Path' field contains the URL '/work/{lastName}?firstName&extension&zipcode'. A callout box points to '{lastName}' with the text '{lastName} – path parameter enclosed in curly brackets ("{}")'. Another callout box points to '?firstName&extension&zipcode' with the text '?firstName&extension&zipcode – query parameters, separated by an ampersand ("&")'. Below the path, there are three method definitions: 'POST' (Phonebook), 'GET', and 'PUT', each with 'Service...' and 'Mapping...' buttons.

In this example, *Last name* is the key that uniquely identifies the contact, and users fill in the given name, extension, and postal code for the given surname.

You have the API name, base path, and path defined. Click **Mapping** to start mapping the incoming data from the HTTP POST request to fields in the service.

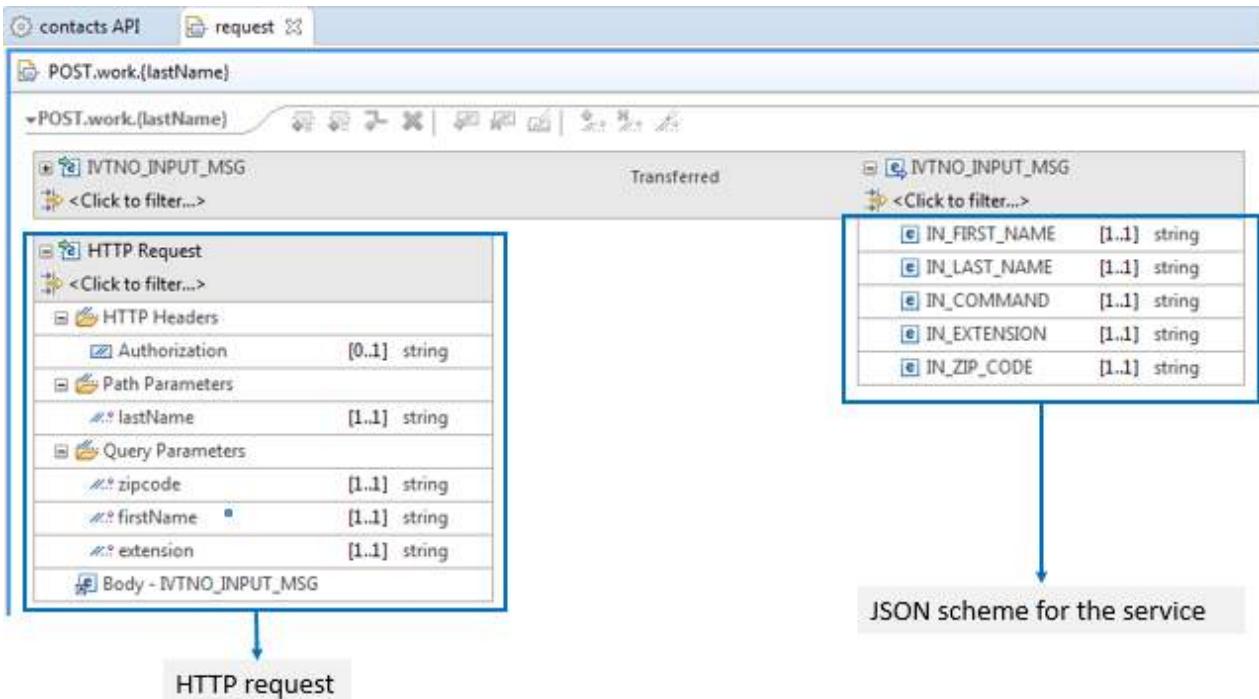
The API name, base path, and path with the path parameter and query parameters are defined and ready to do the mapping for the POST method.

You have the API name, base path, and path defined. You are now ready to click **Mapping** to start mapping incoming data from the HTTP POST request to fields in the service.

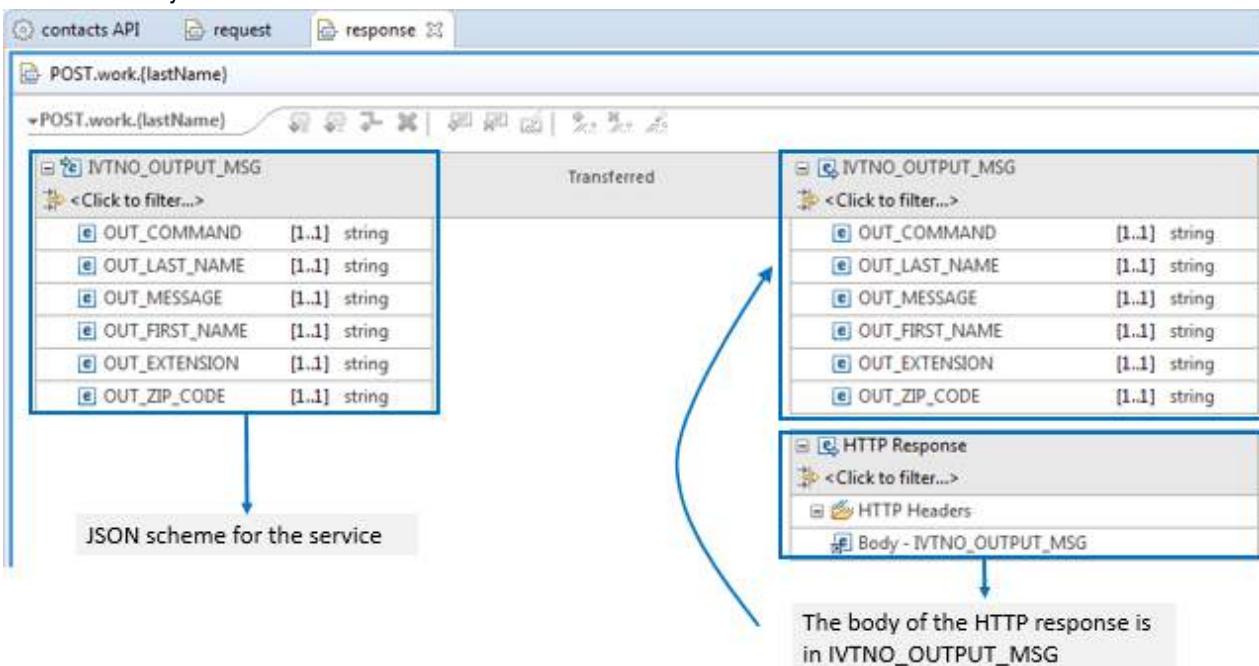


Tip: You can open the request or response-mapping editor for an existing API project directly by double-clicking the `request.map` or `response.map` file in the `/api` folder of your project in the **Project Explorer** view.

As you open the request mapping, you see the HTTP request on one side and the fields in the service on the other. As an API developer, your primary interest is to map the incoming information from the HTTP request, typically in the header or in a parameter from the left, to appropriate fields in the service on the right.



If you open the response mapping, you see fields in the service on the left and the HTTP response on the right. As an API developer, your primary interest is to determine what information from the service to send back to the body in the HTTP response. You can also remove some fields from the response and send back only needed information.



Transform actions

The goal of the mapping editor is to let you more easily design and test a light-weight REST API in a graphical user interface. The goal is achieved mainly through the data transform actions, including:

- **Move:** Move data coming from the HTTP header, path parameter, or query parameters to appropriate fields in the service.
- **Remove:** Hide a field in the service. This action is often used in the response mapping so unnecessary fields are not available to the API for the HTTP response.

- **Assign:** Assigns a static value to a field in the service.

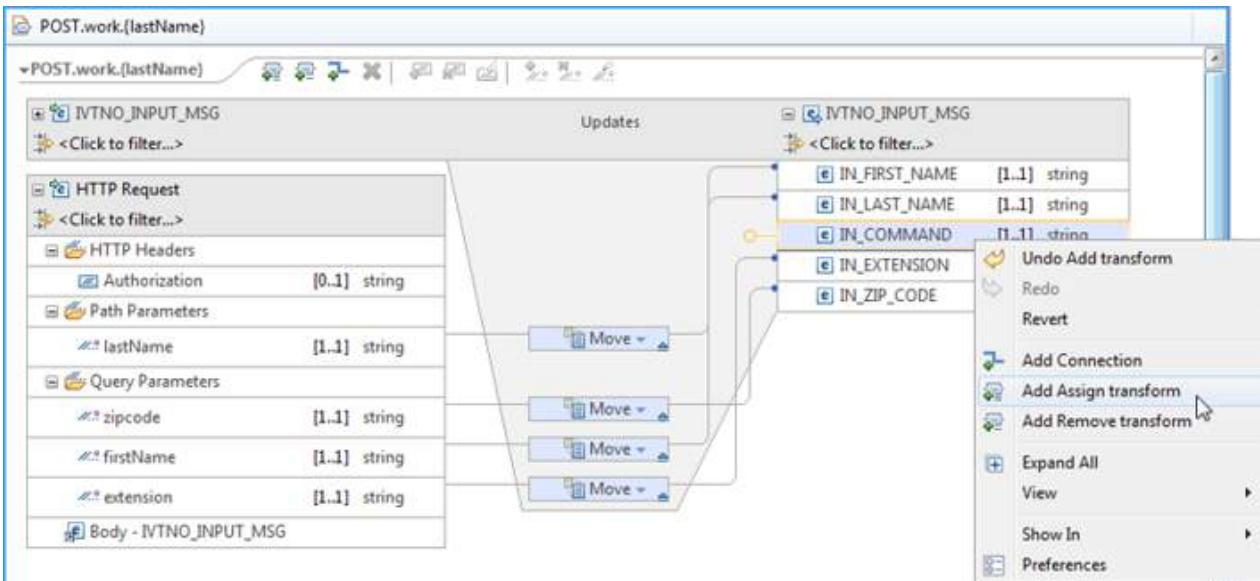
To move data from the HTTP request to appropriate fields in the service, move your mouse over the element in the HTTP request and simply drag and drop. Here, we're getting the last name, first name, extension, and zipcode from the HTTP header, and moving them into the appropriate fields in the service.

The screenshot shows the IBM Worklight Studio interface with two main panes. The left pane displays the 'POST.work.(lastName)' configuration, which includes an 'HTTP Request' section and a 'Body - IVTNO_INPUT_MSG' section. The right pane shows the 'Transferred' section of the 'IVTNO_INPUT_MSG' service, which contains fields like IN_FIRST_NAME, IN_LAST_NAME, IN_COMMAND, IN_EXTENSION, and IN_ZIP_CODE. A mouse cursor is positioned over the 'IN_COMMAND' field in the right pane.

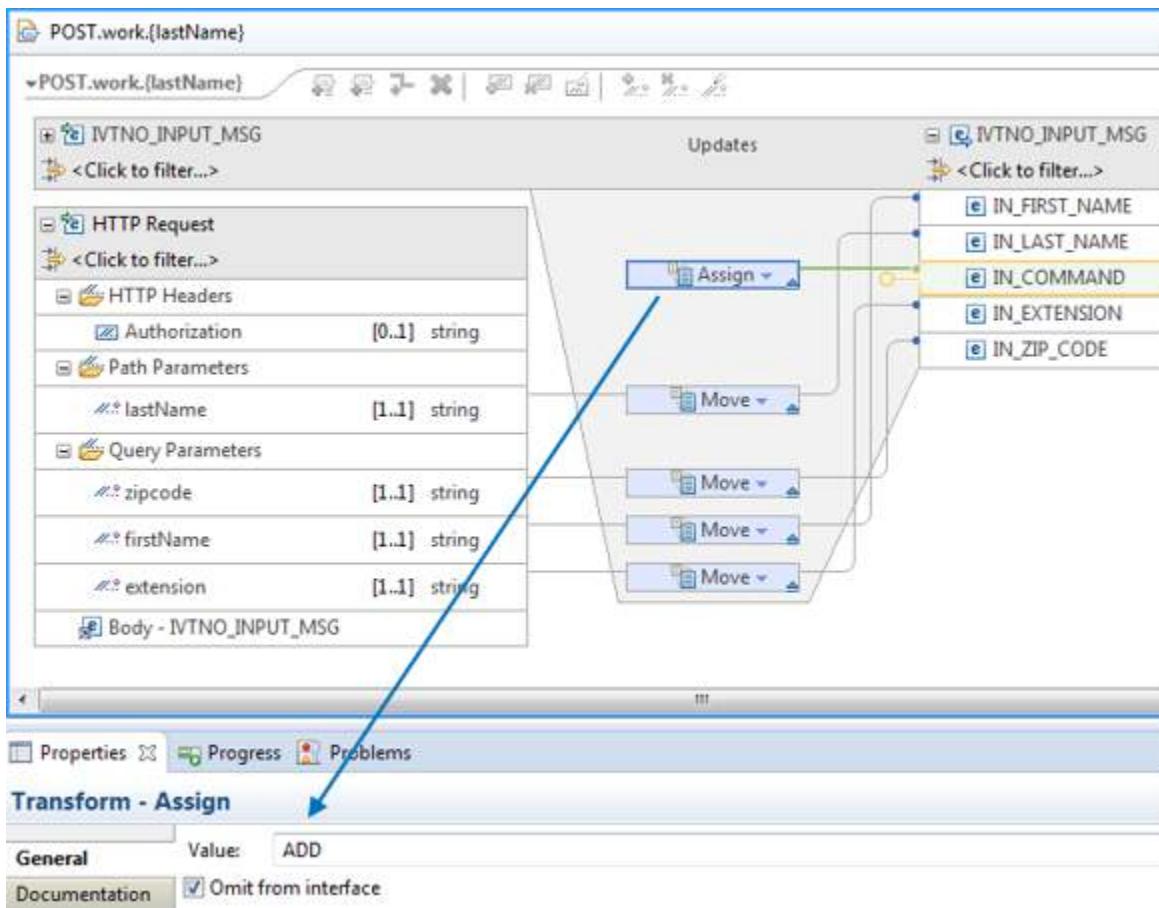
You can also easily add an HTTP header or a query parameter through the right-click context menu:

The screenshot shows the 'HTTP Headers' section of the 'IVTNO_INPUT_MSG' service. A right-click context menu is open over the 'Authorization' item, with the 'Add HTTP Header' option highlighted. Other options in the menu include Undo, Redo, Revert, Add Connection, Expand All, Collapse, View, Show In, and Preferences.

For the work contact to be added, the IBM Z application has defined ADD as the command for adding a record, so we right-click the **IN_COMMAND** field in the service and select **Add Assign transform**.



You can then specify the value to assign for this field in the **Properties** tab. For this example, specify the value of ADD.



Assigning a value of ADD to a field

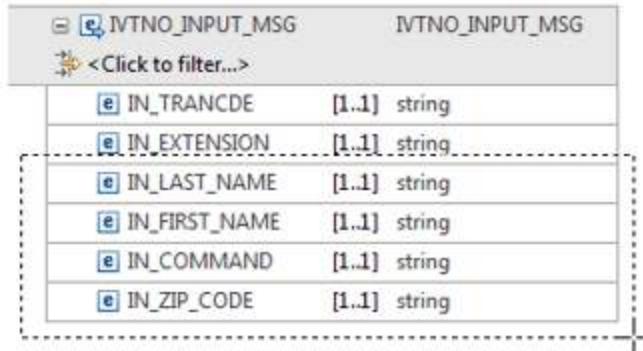
If you make a mistake, you can undo or delete your transform action by right-clicking the action and select **Delete** from the menu.

Multi-select

To select multiple fields for the same action, you can:

- Select a block of fields by selecting the first field, holding down the Shift key, and using the Up or Down arrow to extend the selection.
- Select a block of fields by drawing a box around the fields with your mouse.
- Select multiple fields that are not adjacent to one another by selecting the first field, holding down the Ctrl key, and clicking the other fields you want to select.

The following example shows the use of the mouse to draw a box around the block of fields. As soon as you release the mouse button, the fields in the box will be selected for your transform action.

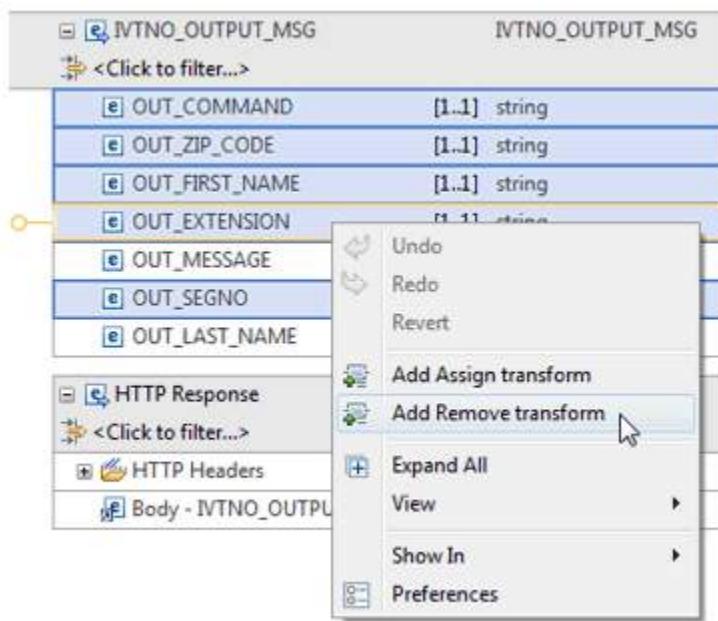


A screenshot of the Eclipse IDE interface showing a table of fields. The table has two columns: 'Name' and 'Type'. The rows contain the following data:

Name	Type
IN_TRANCODE	[1..1] string
IN_EXTENSION	[1..1] string
IN_LAST_NAME	[1..1] string
IN_FIRST_NAME	[1..1] string
IN_COMMAND	[1..1] string
IN_ZIP_CODE	[1..1] string

A dashed rectangular selection box is drawn around the first five rows (IN_TRANCODE through IN_COMMAND). The last row (IN_ZIP_CODE) is not selected.

The following example shows the use of the Ctrl key to select multiple fields to remove from the HTTP response that are not adjacent to each other. In this example, the only information returned is the output message (OUT_MESSAGE) that indicates whether the operation was successful or not for the specific last name (OUT_LAST_NAME). We don't want to echo back the other information.



Re-importing changed services

If a service is modified after your API is created, don't worry! You can re-import the changed service definitions into the project, and the impact of the changes is analyzed and reported, with mapping issues recorded in Eclipse's **Problems** view. If the impact on the existing mappings is as expected, you can click **OK** to re-import the services, and resolve any remaining issues that are reported in the **Problems** view.

The screenshot shows the IBM API Connect API editor interface. At the top, there's a 'Path' section with the URL `/work/{lastName}?` followed by 'Methods' which includes 'POST', 'GET', and 'PUT'. Below this is a 'Properties' tab, a 'Progress' tab, and a 'Problems' tab. The 'Problems' tab is active and displays a list of errors and warnings:

Description	Resource	Path
Errors (5 items)		<ul style="list-style-type: none"> Unable to assign a value to service field "IN_CC" request.map Unable to map path parameter "lastName" to s request.map Unable to map query parameter "extension" to request.map Unable to map query parameter "firstName" to request.map Unable to map query parameter "zipcode" to s request.map
Warnings (39 items)		<ul style="list-style-type: none"> Service field "CA_PATIENT_REQUEST" was added request.map Service field "CA_PATIENT_REQUEST/CA_ADD" request.map Service field "CA_PATIENT_REQUEST/CA_CITY" request.map

In addition to the CICS tutorial and IMS tutorial, make sure you check out the videos and information that explain z/OS Connect EE and the API editor from [IBM Z Community](#).

RESTful web services and API design

The primary focus of RESTful web service design is to identify the z/OS assets that need to be exposed, determine the HTTP methods that you want to support for those assets, and then map the resource identifiers and methods to those assets.

The methods that are defined by the HTTP specification provide a uniform interface for interacting with resources on the web. All web browsers, servers, and applications understand this uniform interface and the semantics of each operation. They can connect to one another and exchange information by using this uniform interface regardless of platforms or technology differences.

After the resources that need to be exposed for the service are determined, the next step is to design a REST API. This API is the user interface to the consumers of the API. The consumers of the API might be application developers who need to build RESTful clients to access the services, or an integration developer who publishes your APIs in IBM API Connect.

A REST API describes a set of resources and a set of methods that can be called to act on those resources. The methods in a REST API can be called from any HTTP client, including client-side JavaScript code that is running in a web browser. The REST API has a base path, which is similar to a context root. All resources in a REST API are defined relative to its base path. The base path can be used to provide isolation between different REST APIs. The HTTP client uses a path relative to the base path that identifies the resource in the REST API that the client is accessing. The paths to a resource can be hierarchical, and a well-designed path structure can help a consumer of a REST API understand the resources that are available within that REST API. The following table lists some example resources for a patient database in the REST API:

Table 73. Example resources

Resource	Description
/patients	All of the patients in the database
/patients/12345	Patient #12345
/patients/12345/orders	All prescription orders for patient #12345
/patients/12345/orders/67890	Prescription order #67890 for patient #12345

Each resource in the REST API has a set of methods that can be called by an HTTP client. The following table lists example methods for the resource /patients/12345:

Table 74. Example operations

HTTP Method	Description
GET	Retrieve the patient details from the database.
PUT	Update the patient details in the database.
DELETE	Delete the patient from the database.

To update information for that patient, the HTTP client would make an HTTP PUT request to /patients/12345.

With a uniform interface for communication, application developers can focus on the resources rather than the methods. They can create their applications without having to deal with a complex system or learn the intricacies of new interfaces. They can also freely change their applications while the communication methods that connect to these resources remain stable.

Each path and method combination in a REST API can also have a set of parameters that can be used by the HTTP client to pass arguments. Each parameter must be defined in the definitions for the REST API. Each parameter has a unique name and type. Several types of parameters are supported by REST APIs in z/OS Connect EE:

Path parameters

Can be used to identify a particular resource. The value of the parameter is passed in by the HTTP client as a variable part of the URL, and the value of the parameter is extracted from the path for use in the operation. Path parameters are denoted by using the syntax {paramName} in the path to the resource. For example, the patient ID can be passed in as a path parameter named patientID:

```
/patients/{patientID}
```

Query parameters

The value of a query parameter is passed in by the HTTP client as a key-value pair in the query string at the end of the URL. As an example, query parameters can be used to pass in a minimum and maximum number of results to be returned by a particular call:

```
/patients?min=5&max=20
```

Header parameters

The HTTP client can pass header parameters by adding them as HTTP headers in the HTTP request. As an example, a header parameter might be used to pass in a unique identifier that identifies the HTTP client that is calling the API:

```
Api-Client-Id: fffe2c5d-42d5-7428-5f5f-abc34ab7f555
```

To assist you with the design and development of this API, z/OS Connect EE provides a graphical editor, the z/OS Connect EE API toolkit.

Designing APIs for use with interceptors

Service interceptors are not called for API requests. Interceptors that are configured for services are only triggered if the service is invoked directly from an HTTP or HTTPS request. They are not triggered if the service is invoked from an API.

Consider the following criteria when designing your APIs to work with interceptors:

- You should only combine services that have the same security constraints (authorization and audit) into a single API. For example, do not combine `getBalance` and `accountTransfer` services into the same API if the authorization or audit requirements of these banking services are different.
- You should only combine services that have the same logging constraints into a single API.
- You should only combine services into a single API if API-level monitoring is sufficient. For example, you want to monitor the number of API requests to an account but not the number of balance inquiries, postings, or transfers.

Designing RESTful APIs

The way that you design APIs can have a significant impact on their adoption. This section lists considerations for API design in general, and principles for effective RESTful implementation in particular.

Think "consumer"

APIs don't exist in isolation. Modern APIs are the way in which the capabilities of services are shared with others. When implemented correctly, APIs that are used inside your organization can enforce consistency and promote efficient reuse. Public APIs that are used outside your organization can expand the reach of your business, by allowing developers to extend the services that you provide. Ease-of-use for consumers is vital for the adoption of the API.

Consumers are developers. They could be developers in your own organization, or a mix of internal and third-party developers. These developers expect APIs that make it quick for them to deliver: quick to learn, easy to use, and aimed at their use cases. Using the API must be faster and more expedient than coding an alternative solution. A successful API encourages developers to use it and to share it with other developers.

An API designer of any API must decide on the following functional requirements:

- What function needs to be exposed, and how.
- Models an API that supports the needs of the user and follows RESTful principles.

A properly designed API appeals to the user, is easy to understand and implement.

Thinking of an API as a business product helps to differentiate it from traditional application programming interfaces. A traditional application programming interface represents a piece of software that you have built and deployed. A modern API represents a package of capabilities that is both attractive to a user and independent of any specific piece of back-end software. The API is designed from the perspective of the intended user. Before you develop one, you must understand:

- Who is the user? You might have one clear target user for the API or a mix of users. If you have a mix of users, you must understand each of them.
- What do they want? Instead of focusing on what the API can do, that is, its functions and capabilities, think about the ways in which it might be used.
- How can you make those use cases as easy as possible? Think about:

Stability

How can you minimize disruption for the consumer when you change the API?

Flexibility

Although you can't exhaustively cover every possibility, how can you build in some flexibility for the consumer? A simple example is allowing either uppercase or lower-case input.

Consistency

What standards can you set for your API so that consumers know what to expect?

Documentation

What documentation can you provide, and how do you make this as straightforward to use as possible?

Think "resources"

Traditional services focused on methods, such as "createAccount" or "updateAccount". Designing RESTful services means that you have to think differently: you focus on *resources*. For example, a resource could be "Account", then the standard HTTP methods are used to operate on that resource. These methods act as verbs for the nouns of the resources.

The verbs POST, GET, PUT, and DELETE are already defined. Try to handle all operations with a combination of these verbs and the resources. The more bespoke verbs that you define, the less generalized your interface becomes.

Design the URIs

From the standpoint of client applications addressing resources, the URIs determine how intuitive the REST Web service is and whether the service will be used in ways that the designers can anticipate. REST Web service URIs should be intuitive to the point where they are easy to guess. Think of a URI as a kind of self-documenting interface that requires little, if any, explanation or reference for a developer to understand what it points to and to derive related resources. To this end, the structure of a URI should be straightforward, predictable, and easily understood.

One way to achieve this level of usability is to define directory structure-like URIs. This type of URI is hierarchical, rooted at a single path, and branching from it are sub-paths that expose the service's main areas. According to this definition, a URI is not merely a slash-delimited string, but rather a tree with subordinate and superordinate branches connected at nodes. For example, in a discussion threading service that gathers a range of topics, you might define a structured set of URIs like this:

```
http://www.myservice.org/discussion/topics/{topic}
```

The root, /discussion, has a /topics node beneath it. Underneath that there are a series of topic names, such as technology and so on, each of which points to a discussion thread. Within this structure, it's easy to pull up discussion threads just by typing something after /topics/.

In some cases, the path to a resource lends itself especially well to a directory-like structure. Take resources organized by date, for instance, which are a very good match for using a hierarchical syntax. This example is intuitive because it is based on rules: `http://www.myservice.org/discussion/2008/12/10/{topic}`. The first path fragment is a four-digit year, the second path fragment is a two-digit day, and the third fragment is a two-digit month. This is the level of simplicity we're after. Humans and machines can easily generate structured URIs like this because they are based on rules. Filling in the path parts in the slots of a syntax makes them good because there is a definite pattern from which to compose them: `http://www.myservice.org/discussion/{year}/{day}/{month}/{topic}`

Some additional guidelines while thinking about URI structure for a RESTful Web service are:

- Hide the server-side scripting technology file extensions (.jsp, .php, .asp), if any, so you can convert to another scripting language without changing the URIs.
- Keep everything lowercase.
- Substitute spaces with either hyphens or underscores
- Avoid query strings as much as you can.
- Instead of using the 404 Not Found code if the request URI is for a partial path, always provide a default page or resource as a response.
- URIs should also be static so that when the resource changes or the implementation of the service changes, the link stays the same. This allows bookmarking. It's also important that the relationship

between resources that is encoded in the URIs remains independent of the way the relationships are represented where they are stored.

Apply HTTP methods explicitly

One of the key characteristics of a RESTful Web service is the explicit use of HTTP methods in a way that follows the protocol as defined by RFC 2616. HTTP GET, for instance, is defined as a data-producing method that's intended to be used by a client application to retrieve a resource, to fetch data from a Web server, or to execute a query with the expectation that the Web server will look for and respond with a set of matching resources.

REST asks developers to use HTTP methods explicitly and in a way that's consistent with the protocol definition. This basic REST design principle establishes a one-to-one mapping between create, read, update, and delete (CRUD) operations and HTTP methods. According to this mapping:

- To create a resource on the server, use POST.
- To retrieve a resource, use GET.
- To change the state of a resource or to update it, use PUT.
- To remove or delete a resource, use DELETE.

An unfortunate design flaw inherent in many Web APIs is in the use of HTTP methods for unintended purposes. The request URI in an HTTP GET request, for example, usually identifies one specific resource. Or the query string in a request URI includes a set of parameters that defines the search criteria used by the server to find a set of matching resources. At least this is how the HTTP/1.1 RFC describes GET. But there are many cases of unattractive Web APIs that use HTTP GET to trigger something transactional on the server: for instance, to add records to a database. In these cases the GET request URI is not used properly, or at least is not used according to RESTful design principles. If the Web API uses GET to invoke remote procedures, it looks like this: GET /adduser?name=Robert HTTP/1.1

It's not a very attractive design because this Web method supports a state-changing operation over HTTP GET. Put another way, this HTTP GET request has side effects. If successfully processed, the result of the request is to add a new user: in this example, Robert, to the underlying data store. The problem here is mainly semantic. Web servers are designed to respond to HTTP GET requests by retrieving resources that match the path (or the query criteria) in the request URI and return these or a representation in a response, not to add a record to a database. From the standpoint of the intended use of the protocol method then, and from the standpoint of HTTP/1.1-compliant Web servers, using GET in this way is inconsistent.

Beyond the semantics, the other problem with GET is that to trigger the deletion, modification, or addition of a record in a database, or to change server-side state in some way, invites Web caching tools (crawlers) and search engines to make server-side changes unintentionally simply by crawling a link. A simple way to overcome this common problem is to move the parameter names and values on the request URI into XML tags. The resulting tags, an XML representation of the entity to create, can be sent in the body of an HTTP POST whose request URI is the intended parent of the entity:

Before:
GET /adduser?name=Robert HTTP/1.1

After:
POST /users HTTP/1.1
Host: myserver
Content-Type: application/xml
<user><name>Robert</name>
</user>

This method is exemplary of a RESTful request: proper use of HTTP POST and inclusion of the payload in the body of the request. On the receiving end, the request can be processed by adding the resource contained in the body as a subordinate of the resource identified in the request URI; in this case the new resource should be added as a child of /users. This containment relationship between the new entity and its parent, as specified in the POST request, is analogous to the way a file is subordinate to its parent

directory. The client sets up the relationship between the entity and its parent and defines the new entity's URI in the POST request.

A client application can then get a representation of the resource using the new URI, noting that at least logically the resource is located under /users:

```
HTTP GET request  GET /users/Robert HTTP/1.1
Host: myserver
Accept: application/xml
```

Using GET in this way is explicit because GET is for data retrieval only. GET is an operation that should be free of side effects, a property also known as idempotence. A similar refactoring of a Web method also needs to be applied in cases where an update operation is supported over HTTP GET:

```
GET /updateuser?name=Robert&newname=Bob HTTP/1.1
```

This changes the name attribute (or property) of the resource. Query strings aren't a bad thing (they're good for implementing filter specifications, for example) but the query-string-as-method-signature pattern that is used in this simple example can break down when used for more complex operations. Because your goal is to make explicit use of HTTP methods, a more RESTful approach is to send an HTTP PUT request to update the resource, instead of HTTP GET, for the same reasons stated earlier.

```
PUT /users/Robert HTTP/1.1
Host: myserver
Content-Type: application/xml

<?xml version="1.0"?>
<user><name>Bob</name>
</user>
```

Using PUT to replace the original resource provides a much cleaner interface that's consistent with REST's principles and with the definition of HTTP methods. The PUT request in this example is explicit in the sense that it points at the resource to be updated by identifying it in the request URI and in the sense that it transfers a new representation of the resource from client to server in the body of a PUT request instead of transferring the resource attributes as a loose set of parameter names and values on the request URI. This also has the effect of renaming the resource from Robert to Bob, and in doing so changes its URI to /users/Bob. In a REST Web service, subsequent requests for the resource using the old URI would generate a standard 404 Not Found error.

Another consideration is handling large result sets. A standard approach is to use explicit pagination: the GET returns a limited number of objects when it is invoked against a set (irrespective of whether it is filtered), and include a link to the next "page" or "batch" that can be requested. The size of a page can be included on the GET, for example as a query string parameter and, if there is a danger of returning too many, it should be set to default if the caller forgets to set it.

As a general design principle, it helps to follow REST guidelines for using HTTP methods explicitly by using nouns in URIs instead of verbs. In a RESTful Web service, the verbs POST, GET, PUT, and DELETE are already defined by the protocol. And ideally, to keep the interface generalized and to allow clients to be explicit about the operations they invoke, the Web service should not define more verbs or remote procedures, such as /adduser or /updateuser. This general design principle also applies to the body of an HTTP request, which is intended to be used to transfer resource state, not to carry the name of a remote method or remote procedure to be invoked.

Be stateless

REST Web services need to scale to meet increasingly high performance demands. Clusters of servers with load-balancing and failover capabilities, proxies, and gateways are typically arranged in a way that forms a service topology, which allows requests to be forwarded from one server to the other as needed to decrease the overall response time of a Web service call. Using intermediary servers to improve scale requires REST Web service clients to send complete, independent requests; that is, to send requests that include all data needed to be fulfilled so that the components in the intermediary servers can forward, route, and load-balance without any state being held locally in between requests.

A complete, independent request doesn't require the server, while processing the request, to retrieve any kind of application context or state. A REST Web service application (or client) includes within the HTTP headers and body of a request all of the parameters, context, and data needed by the server-side component to generate a response. Statelessness in this sense improves Web service performance and simplifies the design and implementation of server-side components because the absence of state on the server removes the need to synchronize session data with an external application.

Figure 1 illustrates a stateful service from which an application can request the next page in a multi-page result set, assuming that the service keeps track of where the application leaves off while navigating the set. In this stateful design, the service increments and stores a previousPage variable somewhere to be able to respond to requests for next.

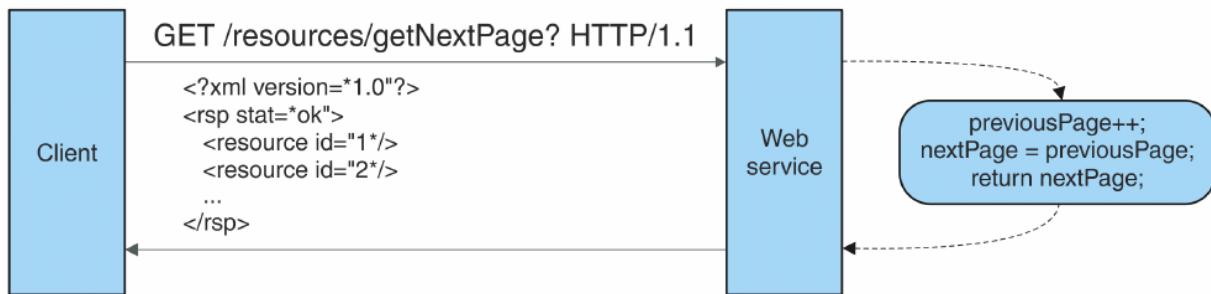


Figure 108.

Stateful services like this get complicated. In a Java Platform, Enterprise Edition (Java EE) environment stateful services require a lot of up-front consideration to efficiently store and enable the synchronization of session data across a cluster of Java EE containers. In this type of environment, there's a problem familiar to servlet/JavaServer Pages (JSP) and Enterprise JavaBeans (EJB) developers who often struggle to find the root causes of `java.io.NotSerializableException` during session replication. Whether it's thrown by the servlet container during `HttpSession` replication or thrown by the EJB container during stateful EJB replication, it's a problem that can cost developers days in trying to pinpoint the one object that doesn't implement the `Serializable` interface in a sometimes complex graph of objects that constitute the server's state. In addition, session synchronization adds overhead, which impacts server performance.

Stateless server-side components, on the other hand, are less complicated to design, write, and distribute across load-balanced servers. A stateless service not only performs better, it shifts most of the responsibility of maintaining state to the client application. In a RESTful Web service, the server is responsible for generating responses and for providing an interface that enables the client to maintain application state on its own. For example, in the request for a multi-page result set, the client should include the actual page number to retrieve instead of simply asking for next.

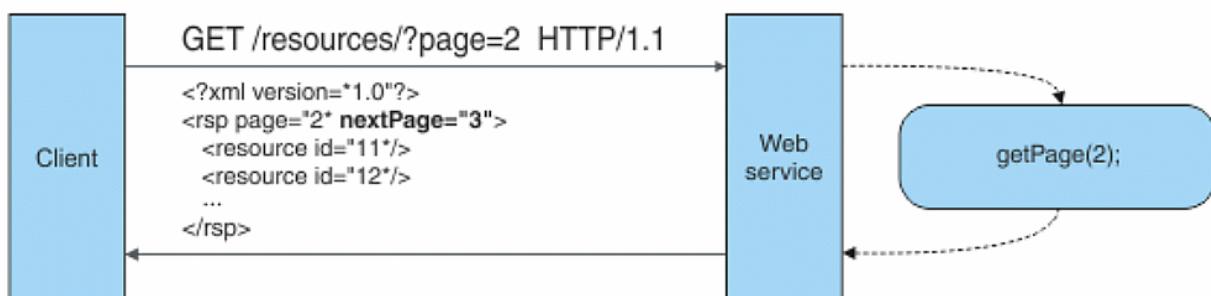


Figure 109.

A stateless Web service generates a response that links to the next page number in the set and lets the client do what it needs to in order to keep this value around. This aspect of RESTful Web service design can be broken down into two sets of responsibilities as a high-level separation that clarifies just how a stateless service can be maintained:

Server

- Generates responses that include links to other resources to allow applications to navigate between related resources. This type of response embeds links. Similarly, if the request is for a parent or container resource, a typical RESTful response might also include links to the parent's children or subordinate resources so that these remain connected.
- Generates responses that indicate whether they are cacheable or not to improve performance by reducing the number of requests for duplicate resources, and by eliminating some requests entirely. The server does this by including a Cache-Control and Last-Modified (a date value) HTTP response header.

Client application

- Uses the Cache-Control response header to determine whether to cache the resource (make a local copy of it) or not. The client also reads the Last-Modified response header and sends back the date value in an If-Modified-Since header to ask the server if the resource has changed. This is called Conditional GET, and the two headers go hand-in-hand in that the server's response is a standard 304 code (Not Modified) and omits the actual resource requested if it has not changed since that time. A 304 HTTP response code means the client can safely use a cached, local copy of the resource representation as the most up-to-date, in effect bypassing subsequent GET requests until the resource changes.
- Sends complete requests that can be serviced independently of other requests. This requires the client to make full use of HTTP headers as specified by the Web service interface and to send complete representations of resources in the request body. The client sends requests that make very few assumptions about prior requests, the existence of a session on the server, the server's ability to add context to a request, or about application state that is kept in between requests.

This collaboration between client application and service is essential to being stateless in a RESTful Web service. It improves performance by saving bandwidth and minimizing server-side application state.

Transfer XML, JSON, or both?

A resource representation typically reflects the current state of a resource, and its attributes, at the time a client application requests it. Resource representations in this sense are mere snapshots in time. This could be a thing as simple as a representation of a record in a database that consists of a mapping between column names and XML tags, where the element values in the XML contain the row values. Or, if the system has a data model, according to this definition a resource representation is a snapshot of the attributes of one of the things in your system's data model. These are the things you want your REST Web service to serve up. The last set of constraints that goes into a RESTful Web service design has to do with the format of the data that the application and service exchange in the request/response payload or in the HTTP body. This is where it really pays to keep things simple, human-readable, and connected. The objects in your data model are usually related in some way, and the relationships between data model objects (resources) should be reflected in the way they are represented for transfer to a client application. In the discussion threading service, an example of connected resource representations might include a root discussion topic and its attributes, and embed links to the responses given to that topic.

And last, to give client applications the ability to request a specific content type that's best suited for them, construct your service so that it makes use of the built-in HTTP Accept header, where the value of the header is a MIME type. Some common MIME types used by RESTful services are shown in Table 1:

Table 75. Common MIME types used by RESTful services	
MIME-Type	Content-Type
JSON	application/json
XML	application/xml
XHTML	application/xhtml+xml

This allows the service to be used by a variety of clients written in different languages running on different platforms and devices. Using MIME types and the HTTP Accept header is a mechanism known as content negotiation, which lets clients choose which data format is right for them and minimizes data coupling between the service and the applications that use it.

Designing APIs for use with interceptors

Service interceptors are not called for API requests. Interceptors that are configured for services are only triggered if the service is invoked directly from an HTTP or HTTPS request. They are not triggered if the service is invoked from an API.

Consider the following criteria when designing your APIs to work with interceptors:

- You should only combine services that have the same security constraints (authorization and audit) into a single API. For example, do not combine getBalance and accountTransfer services into the same API if the authorization or audit requirements of these banking services are different.
- You should only combine services that have the same logging constraints into a single API.
- You should only combine services into a single API if API-level monitoring is sufficient. For example, you want to monitor the number of API requests to an account but not the number of balance inquiries, postings, or transfers.

Assign versions to APIs

Consider whether you need to assign a version to your APIs. Assigning versions is necessary only if there are any breaking changes.

For minor changes that do not break existing users, you can update the operations in place and use the version field in the API editor to track these minor version increments. For example, an operation can be added to an API or an optional field can be added to a schema without breaking the existing usage of an API.

Each API name, service name, and base path name must be unique within a server.

If you decide that you do need to assign versions to your APIs, there are several methods available. The following examples show two methods that use the path.

Base path versions

You can assign versions at the API level by using the base path. This method requires a separate API project for each version but has the advantage of producing separate Swagger documents, one for each version. This method creates a simpler view when there are several versions and the changes to different versions are isolated from each other. This method also reduces the risk of regressions and the degree of testing that is required for any one change. For example,

```
My API Project V1 (basepath= /v1/myapi):  
/v1/myapi/customers  
/v1/myapi/accounts
```

```
My API Project V2 (basepath= /v2/myapi):  
/v2/myapi/customers  
/v2/myapi/accounts
```

Path versions

You can assign a version at the resource level by using the path. With this method, you have multiple versions of a resource and all versions of each resource are kept in the same API project. This method is more appropriate if changes between versions are small or infrequent, meaning that there is little justification for maintaining separate API projects and Swagger documents. For example,

```
My API Project (basepath= /myapi):  
GET /myapi/customers this might call service getCustomers  
GET /myapi/customers/v2 this might call service getCustomersV2
```

```
GET /myapi/customers/v3 this might call service getCustomersV3  
PUT /myapi/accounts this might call service updateAccounts
```

You do not need to assign a version number to your first version. If you need to add versions, you can start with V1 or, as in the above example, V2. Use the **Version** field to record minor non-breaking changes to the existing versions of APIs or services.

Note: You cannot deploy more than one API or service that differs only by the value of the **Version** field in the API toolkit. This field is only a description field and does not enable API or service versioning.

Defining your APIs with the z/OS Connect EE API toolkit

The z/OS Connect EE API toolkit helps you create an API archive file that describes the configuration of the API and the HTTP methods on the resources that can be called.

For each path and method combination, you can select an existing z/OS Connect EE service and specify optional HTTP-to-JSON mappings. The mapping is based on a service archive (.sar) file, which is a compressed collection of files that represent all the information that is needed by a z/OS Connect EE service provider to install and provide the service, and to enable the service as a JSON asset.

The service archive file is generated by the related tool for each of the z/OS Connect EE service providers. You can use the z/OS Connect EE API toolkit to create a service to access CICS, IMS, or IBM MQ through their respective service provider. For CICS access through the WebSphere Optimized Local Adapter (WOLA), you can use the build toolkit.

With the service archive, you can map an HTTP method to a field in the service, or assign a value to a field in the service, by using the z/OS Connect EE API toolkit. After the mapping is complete, you export the API project into an API archive file.

API archive file

A z/OS Connect EE *API archive file* or .aar file, is a compressed file that contains all the files required for the server to install the API.

Note: The contents of the API archive file might change in later releases.

The generated Swagger document in the API archive file is the REST API equivalent of a WSDL document for a SOAP-based web service.

Defining your API in z/OS Connect EE API toolkit

With the API toolkit, you can complete the following REST API development tasks:

- Define an API by specifying a name, some description, the base path, and a version number. The base path is the root of all resources that are associated with this API. The base path can contain a variable.
- Define API contact information by specifying a API author name, URL, and email address to provide consumers of the API additional information if support is needed for using the API.
- Add or remove one or more paths (relative paths). For each path:
 - You can specify a path parameter in the path, which is indicated with curly braces ({}), such as:

```
/myPath/{myVariable}
```

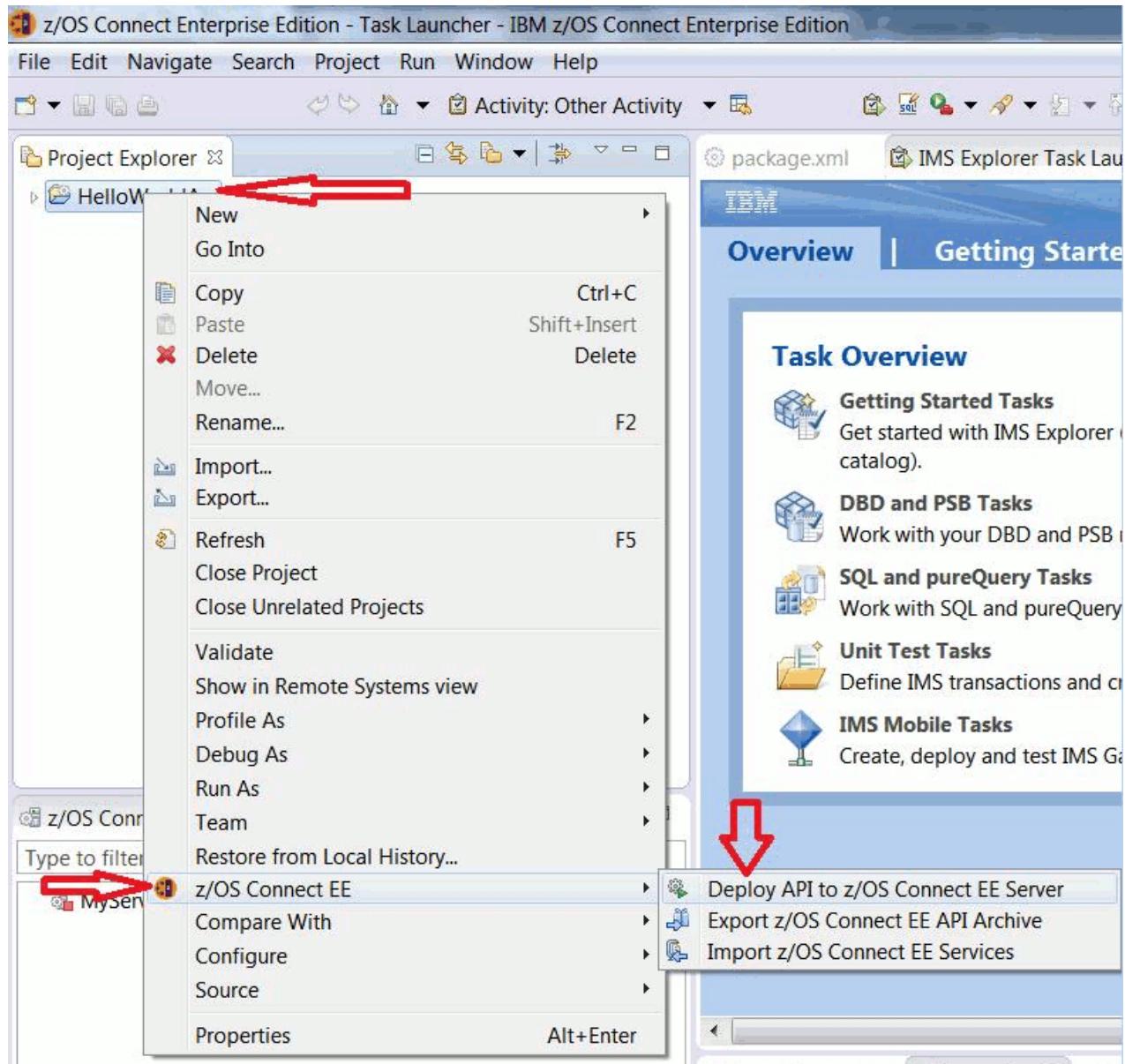
- You can add and remove HTTP methods.
- You can reorder methods.
- Each method can be associated with:
 - A z/OS Connect EE service
 - One or more query parameters
 - One or more headers
- For each method, you can specify the following actions:

- Assign a static value to a field.
- Assign a value to a field from the header, path parameter, or query parameter.
- Remove a field.

After the HTTP-to-JSON mapping is defined, you can deploy the API to the z/OS Connect EE server. From the API toolkit, right-click the API, then click **z/OS Connect EE > Deploy API to z/OS Connect EE Server**.

Testing, starting, and stopping an API

After an API is deployed, you can single-click the API in the **z/OS Connect EE Servers** view to examine its properties. You can further use the provided Swagger UI to examine and test the operations in the API by double-clicking the API in the **z/OS Connect EE Servers** view. You can also start or stop a deployed API, or remove an API from the server, all from within the API toolkit:



Related concepts

[“RESTful web services and API design” on page 570](#)

Related information

[What is Swagger?](#)

[Swagger RESTful API documentation specification](#)

Defining and mapping headers, query parameters, or path parameters

You can add HTTP headers, query parameters, and path parameters to request messages and map them to various request fields. For response messages, HTTP headers can be added and mapped to various response fields.

If a parameter or header is an array, you must specify the data type in the array and the array format. The data type can be string, integer, boolean or number. The array format specifies the how the values in the parameter or header are delimited.

- csv (comma-separated values)
 - Query parameter example: `http://localhost:8080/v1/api/customer?id=1,2,3`
 - Path parameter example: `http://localhost:8080/v1/api/customer/1,2,3/policy`
 - HTTP header example: `X-ROUTE-CODE: ab,cd,wx,yz`
- ssv (space-separated values)
 - Query parameter example: `http://localhost:8080/v1/api/customer?id=1%202%203`
 - Path parameter example: `http://localhost:8080/v1/api/customer/1%202%203/policy`
 - HTTP header example: `X-ROUTE-CODE: ab%20cd%20wx%20yz`
- tsv (tab-separated values): For query parameters and path parameters only.
 - Query parameter example: `http://localhost:8080/v1/api/customer?id=1%092%093`
 - Path parameter example: `http://localhost:8080/v1/api/customer/1%092%093/policy`
- pipes (pipes-separated values)
 - Query parameter example: `http://localhost:8080/v1/api/customer?id=1|2|3`
 - Path parameter example: `http://localhost:8080/v1/api/customer/1|2|3/policy`
 - HTTP header example: `X-ROUTE-CODE: ab|cd|wx|yz`
- multi (multiple parameter instances): For query parameters only.
 - Query parameter example: `http://localhost:8080/v1/api/customer?id=1,id=2,id=3`

Mapping rules for array type parameters and headers

For array type parameters and HTTP headers, note the following rules:

- Because a parameter or header inherits constraints from the service field that they are mapped to, an array type parameter or header inherits the values for minimum items and maximum items from the service fields.
- The HTTP element and the corresponding service field must have the same array depth (that is, they are nested within an identical number of array dimensions).
- Only single-dimension arrays can be mapped to an HTTP header or parameter.

For more information, see [“Request and response mapping rules” on page 583](#).

Request array parameter mapping example

The following example shows an API that allows adding a contact entry based on a unique last name.



Figure 110. A `lastName` path parameter that is defined in the API editor

Based on the specified last name, the POST method would add the first name, middle name, address, and phone numbers. As shown in the following figure, **middleNames**, **addressLines**, and **phoneNumber** from the request are arrays of strings (`string[]`).

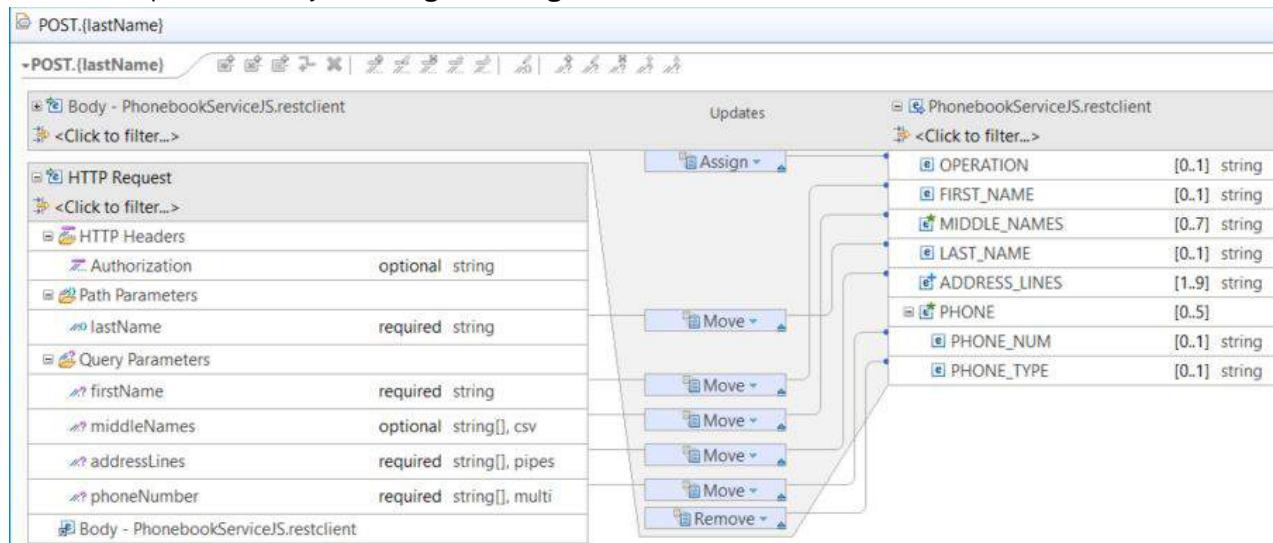


Figure 111. **middleNames**, **addressLines**, and **phoneNumber** as string arrays

Based on the following Swagger document of the API, multiple comma-separated middle names can be specified. The address are pipe-separated lines, and phone numbers can be multiple entries.

```

"post" : {
  "operationId" : "postPhonebookServiceJS.restclient",
  "parameters" : [ {
    "name" : "Authorization",
    "in" : "header",
    "required" : false,
    "type" : "string"
  }, {
    "name" : "lastName",
    "in" : "path",
    "required" : true,
    "type" : "string",
    "maxLength" : 60
  }, {
    "name" : "firstName",
    "in" : "query",
    "required" : true,
    "type" : "string",
    "maxLength" : 30
  }, {
    "name" : "middleNames",
    "in" : "query",
    "required" : false,
    "type" : "array",
    "items" : {
      "type" : "string",
      "maxLength" : 30
    },
    "collectionFormat" : "csv",
    "maxItems" : 7
  }, {
    "name" : "addressLines",
    "in" : "query",
    "required" : true,
    "type" : "array",
    "items" : {
      "type" : "string",
      "maxLength" : 100
    },
    "collectionFormat" : "pipes",
    "maxItems" : 9,
    "minItems" : 1
  }, {
    "name" : "phoneNumber",
    "in" : "query",
    "required" : true,
    "type" : "array",
    "items" : {
      "type" : "string",
      "maxLength" : 100
    }
  }
}

```

```

    "type" : "array",
    "items" : {
      "type" : "string",
      "maxLength" : 20
    },
    "collectionFormat" : "multi",
    "maxItems" : 5
  ]
}
"responses" : {
  "200" : {
    "description" : "normal response",
    "schema" : {
      "$ref" : "#/definitions/postPhonebookServiceJS.restclient_response_200"
    }
  }
}
"headers" : {
  "X-Status-Messages" : {
    "type" : "array",
    "items" : {
      "type" : "string",
      "maxLength" : 100
    },
    "maxItems" : 15,
    "collectionFormat" : "pipes"
  }
}
}
}
}

```

A valid POST request would look as follows.

```
POST https://<serverName>/<portNumber>/<apiName>/Doe?firstName=John&
middleName=J.,R.,Edgar&
addressLines=555%20Bailey%20Ave|San%20Jose,%20CA|95123&
phoneNumber=14081111111&phoneNumber=14082222222&phonenumer=14083333333
```

The response body would look as follows:

```
{
  "ADDRESS_LINES": [
    "555 Bailey Ave",
    "San Jose, CA",
    "95123"
  ],
  "PHONE": [
    {
      "PHONE_NUM": "14081111111"
    },
    {
      "PHONE_NUM": "14082222222"
    },
    {
      "PHONE_NUM": "14083333333"
    }
  ],
  "LAST_NAME": "Doe",
  "FIRST_NAME": "John",
  "MIDDLE_NAME": [
    "J.",
    "R.",
    "Edgar"
  ],
}
```

Request and response mapping rules

Request mapping allows for copying data from the HTTP request context to the service request JSON, while response mapping allows for copying data from the response JSON to the HTTP response context.

Each operation in a z/OS Connect EE API is assigned to a service, which provides the implementation (business logic). Therefore, incoming HTTP requests result in a service invocation. While services describe their request and response interfaces in terms of JSON schema, the interfaces of an API operation also include portions of the HTTP context. For example, the HTTP request context includes, among other things, HTTP headers and the URI (for example, /customer/doe?id=1). The URI includes path

parameters and query parameters. Because API operations adopt the request and response interfaces of their assigned service, mapping also allows for hiding and setting default values for fields in these interfaces so that only information that is relevant to an operation is exchanged with the API client.

The following information is described:

- [“Types of mapping transforms” on page 584](#)
- [“Request mapping compatibility” on page 585](#)
- [“Response mapping compatibility” on page 585](#)

Types of mapping transforms

There are four general types of transforms: one-to-one, one-sided, one-to-many, and many-to-one.

Note: Only one-to-one and one-sided transforms are supported by the API toolkit.

- One-to-one transforms

A one-to-one transform reads from a source element and writes to a target element. An example of a one-to-one transform is one where the value of the source element is copied to the target element.

The **Move** transform is a one-to-one transform that can be used to copy the value of an HTTP request context element to a field in the service request interface, or to copy the value of a field in the service response field to an element of the HTTP response context. A service interface field that is the target of a **Move** transform is always suppressed from the request body of the API operation.

- One-sided transforms

A one-sided transform does not read from a source element. The following examples are one-sided transforms:

- One where a static, literal value is copied into a target element
- One where the target element is suppressed
- One where the target element is renamed

The mapping editor supports the following transform actions that are one-sided transforms:

- The **Assign** transform is a one-sided transform that can be used to set a static value for a field in the service request interface, service response interface, or a header in the HTTP response context. By default, a service request or response interface field that is the target of an **Assign** transform is suppressed from the request or response body of the API operation. Clearing the **Omit from interface** option in the properties for an **Assign** transform action would cause the target field to be exposed, but documented to have a default value.
- The **Remove** transform is a one-sided transform that can be used to suppress fields in service request and response interfaces. Removed service request and response fields are not exposed in the request or response body of the API operation. In addition, removed service request fields are not included in the request JSON passed to the underlying service.
- The **Task** transform is a one-sided transform that can be used to document work yet to be done in the mapping editor. Other transforms cannot be assigned until a Task transform is removed. Tasks are not executed and therefore have no effect on the mapping result.

- One-to-many transforms

A one-to-many transform reads from a single source element and writes to multiple target elements. An example of a one-to-many transform is one where the value of the source element is tokenized using a default or custom delimiter, and the tokens are copied to target elements. Copying of source element tokens is performed in structure order, or a user-defined order.

- Many-to-one transforms

A many-to-one transform reads from multiple source elements and writes to a single target element. An example of a many-to-one transform is one where the values of source elements are concatenated using a default, custom, or no delimiter and copied to the target element. Concatenation of source element values is performed in structure order, or a user-defined order.

Request mapping compatibility

The following table describes the request mapping compatibility.

Table 76. Request mapping compatibility	
Transform action	Mapping rules
Move	<p>An element of the HTTP request context and a service request interface field can be the source and target of a Move transform if the following conditions are met:</p> <ul style="list-style-type: none">• The HTTP element is not already the source of another transform.• The service field is not already the target of another transform.• The service field is not a composite (a field that contains other fields).• The service field is not a descendant of a field that is the target of a Remove transform.• The HTTP element and the service field have the same array depth (are nested within an identical number of array dimensions). <p>Note: The source element of a Move transform inherits relevant constraints of the target element. For example, if an HTTP header of type string is mapped to a service field of type string, and the service field has a maximum length of 30, the HTTP header will be documented with a maximum length of 30.</p>
Assign	<p>A service request interface field can be the target of an Assign transform if the following conditions are met:</p> <ul style="list-style-type: none">• The service field is not already the target of another transform.• The service field is not a composite (a field that contains other fields).• The service field is not a descendant of a field that is the target of a Remove transform.• The service field is not an array nor is it a descendant of an array.
Remove	<p>A service request interface field can be the target of a Remove transform if the following conditions are met:</p> <ul style="list-style-type: none">• The service field is not already the target of another transform.• The service field does not have a descendant that is the target of a Remove transform.• The service field is not a descendant of a field that is the target of a Remove transform.
Task	<p>A service request interface field can be the target of a Task transform if the following conditions are met:</p> <ul style="list-style-type: none">• The service field is not already the target of another transform.

Response mapping compatibility

The following table describes the response mapping compatibility.

Table 77. Response mapping compatibility

Transform action	Mapping rules
Move	A service response interface field and an HTTP response context element can be the source and target, respectively, of a Move transform if the following conditions are met: <ul style="list-style-type: none"> The service field is not already the source of another transform. The HTTP element is not already the target of another transform. The service field is not a composite (a field that contains other fields). The service field and the HTTP element have the same array depth (are nested within an identical number of array dimensions).
Assign	A service response interface field can be the target of an Assign transform if the following conditions are met: <ul style="list-style-type: none"> The service field is not already the target of another transform. The service field is not a composite (a field that contains other fields). The service field is not a descendant of a field that is the target of a Remove transform. The service field is not an array nor is it a descendant of an array. An HTTP response context element can be the target of an Assign transform if the following conditions are met: <ul style="list-style-type: none"> The HTTP element is not already the target of another transform.
Remove	A service response interface field can be the target of a Remove transform if the following conditions are met: <ul style="list-style-type: none"> The service field is not already the target of another transform. The service field does not have a descendant that is the target of a Remove transform. The service field is not a descendant of a field that is the target of a Remove transform.
Task	A service response interface field can be the target of a Task transform if the following conditions are met: <ul style="list-style-type: none"> The service field is not already the target of another transform.

Tips for using the API toolkit for HTTP-to-JSON mapping

You can use the common keyboard and mouse-click actions for your API design needs when you map HTTP request and response to the JSON schema of the service.

Use the keyboard and the mouse for HTTP to JSON mapping:

- **Drag-and-drop** to map data from the HTTP header, path parameter, or query parameters to fields in the service.
- **Right-click to access menus** for actions applicable to the selected elements, such as adding/editing a query parameter, assigning a value to a field, or undo a transform action.
- **Use the Control or Shift key to select multiple fields.**

The basics

When you open the request mapping, you see the HTTP request on one side and the fields in the service on the other. As an API developer, your primary interest is to map the information from the HTTP request, typically in the header or a parameter to appropriate fields in the service.

If you open the response mapping, you see fields in the service on the left and the HTTP response on the right. As an API developer, your primary interest is to determine what information from the service to send back to the body in the HTTP response. Often times you would remove some fields from the response and send back only needed information.

Transform actions

Designing your API is achieved mostly through the data transform actions:

- **Move**: Moves data from the HTTP header, path parameter, or query parameters to appropriate fields in the service.
- **Remove**: Hides a field in the service. This action is often used in the response mapping so unnecessary fields are not available to the API for the HTTP response.
- **Assign**: Assigns a static value to a field in the service.
- **Task**: Specifies a description or detailed documentation for communication purposes.

To move data from the HTTP request to appropriate fields in the service, move your mouse over the element in the HTTP request and drag.

You can also easily add an HTTP header or a query parameter through the right-click menu.

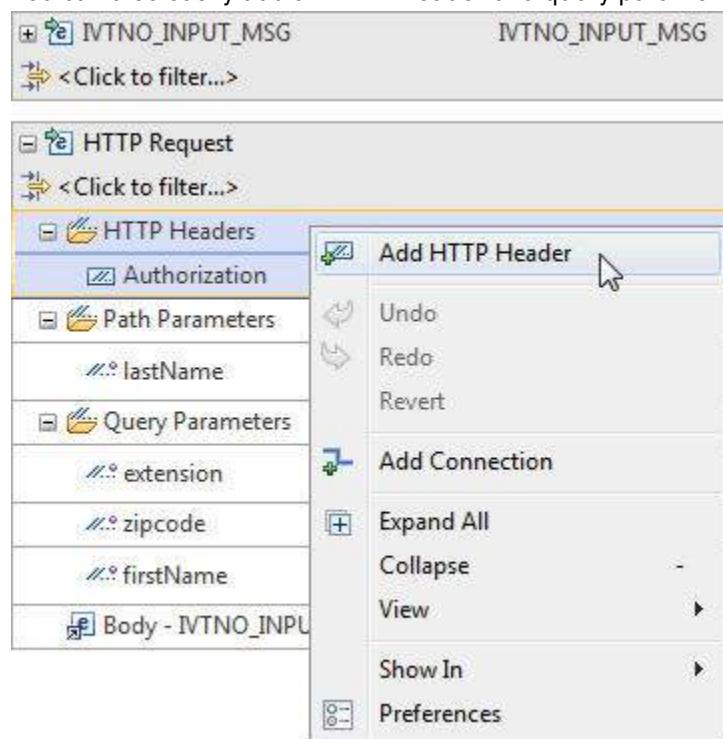


Figure 112. The right-click menu

If you make a mistake, you can undo or delete your transform action by right-clicking the action and select **Delete** from the menu.

Selecting multiple fields

To select multiple fields for the same action, you have several options:

- Select a block of fields by selecting the first field, holding down the Shift key, and clicking the Up or Down arrow to extend the selection.
- Select a block of fields by drawing a box around the fields with your mouse.

The screenshot shows a mapping editor interface for a schema named 'IVTNO_INPUT_MSG'. A dashed rectangular selection box is drawn around five fields: 'IN_TRANCODE', 'IN_EXTENSION', 'IN_LAST_NAME', 'IN_FIRST_NAME', and 'IN_COMMAND'. Each field is represented by a row in a table-like structure with columns for the field name and its type/size.

	IVTNO_INPUT_MSG
<input checked="" type="checkbox"/>	IN_TRANCODE [1..1] string
<input checked="" type="checkbox"/>	IN_EXTENSION [1..1] string
<input checked="" type="checkbox"/>	IN_LAST_NAME [1..1] string
<input checked="" type="checkbox"/>	IN_FIRST_NAME [1..1] string
<input checked="" type="checkbox"/>	IN_COMMAND [1..1] string
<input checked="" type="checkbox"/>	IN_ZIP_CODE [1..1] string

Figure 113. Selecting a block of fields by drawing a box

- Select multiple fields that are not next to one another by selecting the first field, holding down the Control key, and clicking the other fields that you want to select.

The screenshot shows a mapping editor interface for a schema named 'IVTNO_OUTPUT_MSG'. Multiple fields are selected: 'OUT_COMMAND', 'OUT_ZIP_CODE', 'OUT_FIRST_NAME', 'OUT_EXTENSION', 'OUT_MESSAGE', 'OUT_SEGNO', and 'OUT_LAST_NAME'. A context menu is open over the 'OUT_EXTENSION' field, showing options like Undo, Redo, Revert, Add Assign transform, Add Remove transform (which is highlighted), Expand All, View, Show In, and Preferences.

	IVTNO_OUTPUT_MSG
<input checked="" type="checkbox"/>	OUT_COMMAND [1..1] string
<input checked="" type="checkbox"/>	OUT_ZIP_CODE [1..1] string
<input checked="" type="checkbox"/>	OUT_FIRST_NAME [1..1] string
<input checked="" type="checkbox"/>	OUT_EXTENSION [1..1] string
<input checked="" type="checkbox"/>	OUT_MESSAGE [1..1] string
<input checked="" type="checkbox"/>	OUT_SEGNO [1..1] string
<input checked="" type="checkbox"/>	OUT_LAST_NAME [1..1] string

Figure 114. Selecting multiple fields that are not next to one another by holding down the Control key

Handling the null type in JSON schema

Swagger 2.0 does not support the null type because Swagger supports only a subset of JSON schema.

If a service that is imported into the API contains nullable fields, as shown in the following example with "type": ["null", "string"], the first supported type (in this example, "string") is used, and only the Remove transform is allowed in the mapping editor.

```
{
  "type": "object",
  "properties": {
    "location": {
      "type": [
        "null",
        "string"
      ],
      "maxLength": 16
    },
    "required": [
      "LOCATION"
    ]
}
```

Because Swagger 2.0 does not support the null type, nullable fields are indistinguishable from non-nullable fields in the Swagger document. When null types are sent to the z/OS Connect EE server, they are passed through in the body and not represented in the Swagger document.

Creating a REST API

Create a REST API for your z/OS Connect EE services by exporting your service as a service archive (.sar) file. Import the service archive into your z/OS Connect EE API project in the editor to model and define your API.

Creating an API project

Create an API project by opening the **z/OS Connect EE API Project** wizard when you are in the **z/OS Connect Enterprise Edition** perspective.

Before you begin

1. Create the service archive (.sar) file for your service in the supported tool for each of the z/OS Connect EE service providers.
2. Switch to the **z/OS Connect Enterprise Edition** perspective.
 - a. From the main menu, select **Window > Open Perspective > Other**. The Select Perspective wizard opens.
 - b. Select **z/OS Connect Enterprise Edition**.

Procedure

Create a z/OS Connect EE API by first creating an API project in the **z/OS Connect Enterprise Edition** perspective.

1. From the main menu bar, click **File > New > z/OS Connect EE API Project**.
2. In the **z/OS Connect EE API Project** wizard, enter the project properties, and click **Finish** to create the project.

Project property	Description
Project name	Unique name for your project. The project name is used as the folder name for all resources in this project. The project name can contain any of the following characters: 0-9, A-Z, a-z, period (.), underscore (_), and hyphen (-).
API name	The name of your API. If the API name matches an existing service name, the API takes precedence over the service when the request is connected through IBM API Connect. Note: The default API name is the project name, with all spaces replaced by hyphens.
Base path	The unique <i>basePath</i> attribute that specifies the root of all the resources in this API. <ul style="list-style-type: none">• If the same <i>basePath</i> is specified in different API archive files, the first is installed successfully and all others fail.• If the API base path overlaps with the <i>invokeURI</i> of a service, the API is invoked rather than the service.

Project property	Description
Description	Description for your API.

The API project is created in the **Project Explorer** view. The API package .xml file opens in the API editor where you can model your API.

What to do next

[Model your API.](#)

Modeling an API with the API toolkit

You can model your API by adding paths, creating methods, and associating the methods with related z/OS Connect EE services.

Before you begin

Create an API project. If your API project is not yet open in the API editor, in the **Project Explorer** view, open the **package.xml** file located in your API project.

Procedure

1. Import your service archive (.sar) file or files to your API project. Right-click an API project and click **z/OS Connect EE > Import z/OS Connect EE Services**.
2. In the API editor, specify a path.

You can add a URI path parameter with curly brackets ({}). For example, the following path uses *patID* as a path parameter:

```
/Patient/{patID}
```

Multiple path parameters can be added to the path:

```
/Patient/{patID}/{caseNum}
```

You can add paths by selecting the  icon next to **Path**.

Tips:

- A URI path parameter applies to all HTTP methods in the path. You can customize the data type for each HTTP method later in the mapping editor when you define the HTTP-to-JSON mapping for request and response messages. Specify a path parameter if it is applicable to all operations. Otherwise, use a query parameter, which is specific to an operation. Query parameters are specified in the mapping editor.
- When you save your changes (**Ctrl + S**), **package.xml** is updated, and the changes are reflected in **swagger.json**.

3. Add methods.

By default, the POST, GET, PUT, and DELETE methods are listed. You can add methods by clicking the  icon next to **Methods**.

HTTP method	Description
POST	Adds a new instance of the specified resource.
GET	Retrieves details about the specified resource.
PUT	Updates information for the specified resource.
DELETE	Deletes information for the specified resource.

HTTP method	Description
PATCH	Applies partial updates to the specified resource.
HEAD	Retrieves the response identical to what a GET request would retrieve about the specified resource but without the response body.

4. Assign a service to a method by clicking **Service**, then select a z/OS Connect EE service or import one from your **Workspace** or **File System**.

Tip: If a service is changed after the API is created, upon re-importing the changed service, the impact of the changes is analyzed and reported. For more information, see “[Re-importing changed services](#)” on page 600.

What to do next

Define HTTP-to-JSON mapping in the mapping editor.

Defining HTTP-to-JSON mapping

The next step in modeling your API is to map HTTP headers, path parameters, and query parameters with fields in the JSON schema of a z/OS Connect EE service. The JSON schema is the metadata that is used to communicate with the service.

Before you begin

If your API project is not yet open in the API editor, in the **Project Explorer** view, open the **package.xml** file located in your API project.

Procedure

- In the API editor, click **Mapping** on a method to map fields in the JSON schema of the service, and then select **Open Request mapping** (HTTP request) or **Open Response mapping** (HTTP response).
The mapping editor opens.
- Create HTTP headers and query parameters. Each HTTP method can have its own HTTP headers and query parameters.
 - To create an HTTP header for the request or response messages, right-click **HTTP Headers** > **Add HTTP Header**.
 - To create a query parameter, right-click **Query Parameters** > **Add Query Parameter**.

The value of a query parameter is passed in by the HTTP client as a key-value pair, connected by an ampersand ("&"), in the query string at the end of the URL. The following example shows two query parameters, **min** and **max**, that are used to pass in a minimum value and a maximum value:

```
/patients?min=5&max=20
```

- To customize the data type for a URI path parameter for this method, right-click the path parameter and select **Edit Path Parameter**.

Note:

- The data type for HTTP headers, query parameters, and path parameters can be string, integer, boolean, number, or array. When the data type is set to array, you must specify the array data type and array format. For more information, see “[Defining and mapping headers, query parameters, or path parameters](#)” on page 581.
- If there are multiple HTTP headers, query parameters, or path parameters, to enhance readability, you can reorder them by right-clicking an entry and selecting the appropriate menu option to move it up or down. The same order is reflected in the generated Swagger document but does not affect the run time. Headers or query parameters can be passed in from the API call in any order. Path

parameters, however, are defined in the API editor and their values must be passed in the defined sequence.

3. Map HTTP headers, query parameters, and path parameters with the service JSON schema by dragging the header or parameter to a field in the JSON schema.

This is known as the Move transform. For more information about the mapping rules, see “[Request and response mapping rules](#)” on page 583.

4. Assign any static values or remove any unnecessary field from the service JSON schema.

Mapping actions	Description
Assign	Assigns a static value to a field in the service JSON schema for the HTTP request or HTTP response.
Remove	Hides the value from the HTTP request or HTTP response.
Task	Allows adding a description and detailed documentation for communication purposes.

Note: Path and query parameters inherit the minimum, maximum, minLength, maxLength, exclusiveMinimum, and exclusiveMaximum properties of the service field to which they are mapped.

To use the Assign mapping action:

- a. Right-click the field to be assigned and select **Add Assign transform**.
- b. Select the **Assign** box to open the **Properties** view.
- c. In the **General** tab of the **Properties** view, type the required value into the **Value** field.
- d. Optionally, add a description in the **Documentation** tab.

Tip: Unmapped fields in the request JSON schema remain unchanged and get their values from the body of the HTTP request. If no value is provided from the JSON payload, the default value that is specified in the service interface editor during service definition is used. z/OS Connect EE will ensure that there is always a JSON object for service interface defaults to be applied to. Where the received HTTP request has no body, z/OS Connect EE will still apply default values set for fields in the service interface.

Example

In the following example, a healthcare service provider is developing an API to allow their patients to look up their registration information based on their patient ID.

In the mapping editor, an API developer created and mapped the following HTTP header, path parameter, and query parameters for the GET HTTP method:

HTTP header: X_TrackingID

Enables the service caller to provide the routing information (CA_ROUTING_CODE) for a patient with the HTTP header variable called **X_TrackingID**.

Path parameter: patID

Enables the service caller to set the value for CA_PATIENT_ID as the provided **patID** value.

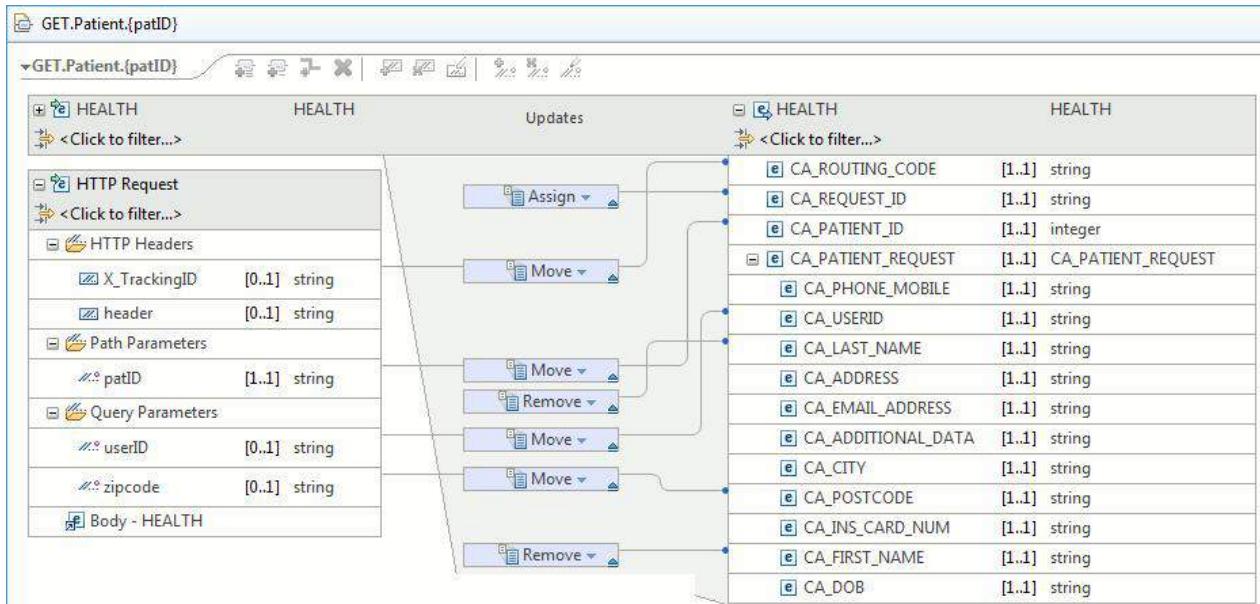
Query parameter: userID

Enables the service caller to set the value for CA_USERID as the provided **userID** value.

Query parameter: zipCode

Enables the service caller to set the value for CA_POST_CODE as the provided **zipCode** value.

An example mapping configuration for the HTTP request and request JSON schema.



How to define multiple response codes

You can define multiple HTTP response codes to allow you to quickly determine the results of a REST API operation. Defining multiple response codes also allows you to efficiently detect and handle errors with your REST API. You can define custom HTTP headers and customize the HTTP response body to relay the appropriate information from your REST API.

The screenshot shows the 'API Editor' interface. In the 'Describe your API' section, the 'Name' is set to 'contacts', 'Base path' is '/phonebook', and 'Version' is '1.0.0'. Below this, under the 'Path' section, the path '/contacts' is defined. Under the 'Methods' section, four methods are listed: POST, GET, PUT, and DELETE. Each method has a service assigned (phonebookService) and buttons for Service..., Mapping..., and file operations (up, down, delete).

Before you begin

If your API project is not yet open in the z/OS Connect EE API editor, in the **Project Explorer** view, open the **package.xml** file located in your API project.

Restriction:

To successfully create and deploy an API utilizing multiple response codes, ensure that you are using the minimum required version of z/OS Connect EE, version 3.0.16.

Procedure

1. In the API editor, click **Mapping... > Define Response Codes** for the method you want to add a response to.

Response details for your API method is shown. A default response code, 200, is preconfigured for each API method.
2. Under Responses, click the **Add Response** button () to define a new HTTP response code.

The Add Response window opens.
3. In the **Response code** field of the Add Response window, specify a HTTP response code. Select a response code using the drop-down list, or alternatively specify your own response code in the numeric range of 100 to 599 by typing into the text field.
4. Use the **Description** field to specify the purpose of the response code.
5. Define the rules that are used to determine when this response code is issued. A rule is composed of (from left to right) a name, a service response field, a comparison operator, and a comparison value.
 - a) In the **Rule name** field, specify a short name that identifies the rule.
 - b) In the **service field** drop down box, select a service response field to compare against.

Restriction: Array types cannot be used for a service field.

- c) In the **comparison operator** drop down box, select an operator to compare the service field value against the comparison value.

Comparison operators will change according to the datatype specified in the service field box.
- d) In the **comparison value** field, specify a value to be compared with the corresponding service field value.

The structure of rules used for multiple response codes typically mimics the conventions used for performing programmatic comparisons in Java. To see a demonstration of rules used for multiple response codes, see the IMS API example “[Define contacts API response codes](#)” on page 145.

Restriction: Array type service fields cannot be used for rule definition.

6. When you are finished specifying rules for your response, click **OK**.

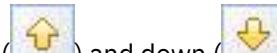
Important:

- To later edit the rules for a given response, return to the Add Response window by clicking **Response... > Edit Response**.
 - To delete a rule, click the **Delete rule** button () for that rule.
 - Rule errors will prevent you from saving a response code. To resolve errors within the Add Response window, mouse over any error indicators to be prompted with the solution.
7. (Optional): Define unique service response JSON mapping for when this response code is issued.
 - To configure unique response mapping for an API operation, click **Mapping... > Open Response Mapping** for the response you want to edit.
 - To delete a unique response mapping, and revert to the default response mapping, click **Mapping... > Clear Response Mapping** for the response you want to delete.

Specifying a unique service response mapping allows you to return the appropriate information for a given response. For more information about response field mapping, see “[Defining HTTP-to-JSON mapping](#)” on page 591.

Important:

If you do not specify a unique service response mapping, the default 1-to-1 mapping between the service and API response fields will be used.



8. Specify the order in which response codes are evaluated. Click the up () and down () arrow buttons for each response to increase or decrease the precedence of evaluation for each response code. The response with the lowest precedence is marked as the default response code.

Important:

Responses are interpreted sequentially, in the order of highest precedence (top) to the lowest precedence (bottom). During rule evaluation, all service response fields are tested against the rules defined for the API method. After rule evaluation, API response field mapping is performed.

During an API operation, if none of the rule sets for each response evaluates to true, the default response code is issued. The default response is marked with a check mark next to the response code.

9. Once you are finished, click **File > Save** (Crtl-S) to save changes made to your API methods.

Results

You have created multiple response codes for an API operation that are issued according to their specified rule evaluation.

What to do next

[Deploy an API](#).

Related reference

[“HTTP response code reference” on page 824](#)

REST APIs use HTTP status codes to relay the results from an API operation. Status codes are primarily useful for detecting REST API errors, and by specifying your own response codes you can diagnose those errors more efficiently.

Deploying an API in the API toolkit

After you define your API, you can deploy the API directly to your z/OS Connect EE server from the API toolkit.

Before you begin

By default, deployed APIs are automatically started. You can change this behavior by configuring the z/OS Connect EE preferences:

1. From the main menu bar, click **Windows > Preferences**.
2. In the **Preferences** wizard, select z/OS Connect EE.
3. Make your changes and click **Apply** and then **OK**.

Make any final edits to the Swagger document. For example, you might want to add security and authentication requirements. Manual changes to the Swagger document in the API editor are preserved when saved. To edit the Swagger document with your preferred JSON editor, find the location of the Swagger document by right-clicking the api-docs/ folder of your API project in the Project Explorer view and selecting **Properties**. The location of the Swagger document is displayed in the Location field.

Ensure that you are connected to the server. For more information, see [“Connecting to a z/OS Connect EE server” on page 603](#).

About this task

You can deploy an API directly from the package editor by clicking **Deploy API to z/OS Connect EE Server** () in the corner.

You can also deploy the API from **Project Explorer** view, or by using the **apideploy** command on the server.

Procedure

To deploy the API from the **Project Explorer** view:

1. In the **Project Explorer** view, select one or more API projects and right-click to select **z/OS Connect EE > Deploy API to z/OS Connect EE Server**.
2. In the **Deploy API** window, select the server to which to deploy the APIs.
3. If an API of the same name exists and you want to overwrite it, select the **Update existing APIs** checkbox.
4. Click **OK**.

Alternatively, if you need to deploy the API by using the **apideploy** command on the server:

- a. Right-click the API project and click **z/OS Connect EE > Export z/OS Connect EE API Archive** to save the API as an API archive (.aar) file to a location of your choice.
- b. Transfer the API archive file in binary mode to a file system that the API Deployment utility can access.
- c. In an OMVS shell, enter the **apideploy -deploy** command. For more information, see [“Deploying an API” on page 674](#).

Results

Deployment result is reported. Errors that occur during the deployment are recorded in the **Problems** view. A deployed API is automatically started unless specified otherwise in the z/OS Connect EE preferences window.

Tips:

- To see the newly deployed APIs on the server, in the **z/OS Connect EE Servers** view, right-click the **APIs** folder, and select **Refresh**.
- If you right-click the server itself and select **Refresh Server**, status for all APIs and services that are deployed on the server is fetched and refreshed.
- Save a copy of the API project in your source control management system.
- The server stores the API in a file with the API name as the file name and a file extension of .aar.

What to do next

[Examine and test the API directly within the API toolkit](#).

Related concepts

[“Setting preferences for the API toolkit” on page 606](#)

Related tasks

[“Generating an API archive in the API toolkit” on page 600](#)

Examining, testing, starting and stopping an API

You can examine, test, start, and stop an API in the **z/OS Connect EE Servers** view.

Before you begin

Switch to the **z/OS Connect Enterprise Edition** perspective.

Ensure that you are connected to the server. For more information, see [“Connecting to a z/OS Connect EE server” on page 603](#).

About this task

The **z/OS Connect EE Servers** view in the **z/OS Connect Enterprise Edition** perspective provides a list of defined host connections and the APIs that are deployed on the servers.

Procedure

1. In the **z/OS Connect EE Servers** view, ensure that the server is connected. If the server is not connected, right-click the server and select **Connect**.
2. Expand the server and the **APIs** folder to see all APIs that are deployed to the server.
3. For each API, you can examine its status, description, and documentation. You can also start, stop, or remove it.
 - To examine the status, version number, api URL, description, and documentation of an API, click the API to see the information in the **Properties** view.
 - To examine and test the operations in the API in Swagger UI, double-click the API or right-click the API and select **Open in Swagger UI**.

Important: Before you use the "Try it out!" button in the Swagger UI, if your z/OS Connect EE server connection is secure (SSL/TLS), install and trust the server's certificate by using Certificate Manager (**Start > Run** and select `certmgr.msc`) in Windows, or by using Keychain Access in MacOS. Because Swagger UI runs on a workstation rather than the server where the API is hosted, cross-origin resource sharing (CORS) must be enabled on the server, and the client (Swagger UI) must accept the self-signed certificate. For more information about CORS and related configuration, see "[Installing and trusting a self-signed certificate for Swagger UI](#)" on page 597.

Tips:

- You can disable the prompt about a valid TLS certificate is required. From the main menu bar, click **Window > Preference > z/OS Connect EE**, and specify your choice.
- If you prefer to open the Swagger UI outside of the editor in an external browser, configure your Eclipse preferences by selecting **Window > Preference > General > Web Browser**, and specify your browser of choice.
- If you added custom HTTP headers to your API response, then by default these headers are not displayed in the Swagger UI **Response Headers** section. To display custom HTTP response headers for your API, you need to add the following attribute to the `cors` element in your `server.xml` configuration file:

```
exposeHeaders="customHeader1, customHeader2"
```

For more information about configuring Cross-Origin Resource Sharing, see "[Configuring Cross-Origin Resource Sharing on a z/OS Connect Enterprise Edition Server](#)" on page 300.

- To temporarily stop further requests through the API, right-click the API and select **Stop API**.
- To restart the API, right-click and select **Start API**.
- To remove an API, the API must be stopped first. After an API is stopped, right-click and select **Remove API**.

Tip: You can set the preferences to automatically stop an API upon removal or to automatically start an API upon deployment in the z/OS Connect EE preferences window.

Related concepts

["Setting preferences for the API toolkit"](#) on page 606

Installing and trusting a self-signed certificate for Swagger UI

To use the embedded Swagger UI, because it runs on a workstation rather than the server where the API is hosted, cross-origin resource sharing (CORS) must be enabled on the server, and the client (Swagger UI) must accept the self-signed certificate.

Before you begin

For more information about enabling CORS on the z/OS Connect EE server, see "[Configuring Cross-Origin Resource Sharing on a z/OS Connect Enterprise Edition Server](#)" on page 300.

About this task

Swagger UI does not provide diagnostics when the **Try It Out!** function fails due to a CORS issue. CORS is enforced by the browser and Swagger UI does not have the visibility to the interaction between the browser and the resource (server).

When you use Swagger UI in the API toolkit to invoke an API over a secure connection, if the API is hosted on a server with a self-signed or invalid certificate, you would encounter one of the following issues:

- The browser prompts you whether to accept or decline the self-signed or invalid certificate. If you accept, the request is sent to the server. If you decline, the request is not sent and Swagger UI indicates that no response is received. You are prompted again after restarting the browser.
- The browser does not prompt you at all, the request is not sent to the server, and Swagger UI indicates that no response is received.

By default, when you open Swagger UI from within the API toolkit, the Eclipse internal web browser is used. Installing the certificate into your external web browsers might not suffice. Installing and trusting the certificate at the operating system level is the best way to ensure successful API invocation over a secure connection.

Procedure

To install and trust a self-signed certificate, export the server certificate from the server and import it to the workstation where the API toolkit is running.

1. On the z/OS Connect EE server, export the server certificate.

- If the z/OS Connect EE server is using a SAF key ring:
 - a. On the server, generate a Certificate Authority (CA) root certificate. For example:

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('ZCEE Sample CA') O('MYCOM') OU('MYORG')) SIZE(1024)  
WITHLABEL('ZCEE-Sample-Certification') TRUST  
NOTAFTER(DATE(2021/12/31))
```

- b. On the server, create a server certificate, signed by the CA. For example:

```
RACDCERT ID (ZOSCONN) GENCERT SUBJECTSDN(CN('zceeserver.mycom.com')  
O('MYCOM') OU('MYORG')) SIZE(1024)  
SIGNWITH (CERTAUTH LABEL('ZCEE')) WITHLABEL('ZCEE-Server') NOTAFTER(DATE(2021/12/31))
```

- c. Create a RACF key ring.

```
RACDCERT ADDRING(ZCEE.KEYRING.wsc) ID(ZOSCONN)
```

- d. Connect the server certificate to the key ring.

```
RACDCERT ID(ZOSCONN) CONNECT (RING(ZCEE.KEYRING.wsc) LABEL('ZCEE-Cert') CERTAUTH)  
RACDCERT ID(ZOSCONN) CONNECT (RING(ZCEE.KEYRING.wsc)  
LABEL ('ZCEE-Server') DEFAULT
```

- e. Update the server configuration in `server.xml` to point to the newly created key ring.

```
<ssl id="defaultSSLConfig" keyStoreRef="racfKeyStore"  
sslProtocol="SSL_TLS"  
serverKeyAlias="ZCEE-Server" />
```

- f. Export the certificate and save it to the USS file system. For example:

```
RACDCERT CERTAUTH EXPORT(LABEL('ZCEE-Cert'))  
DSN('ZCONN.CERT.LIBCERT') FORMAT(CERTDER) PASSWORD('password')  
OPUT 'ZCONN.CERT.LIBCERT' '/u/devuser1/zcee/zcee.certAuth.cer' BINARY  
CONVERT(NO)  
RACDCERT ID(ZOSCONN) EXPORT(LABEL('ZCEE-Server')) DSN('ZCONN.CERT.SERVER.P12') FORMAT(PKCS12DER)  
PASSWORD('password')  
OPUT 'ZCONN.CERT.SERVER.P12' '/u/devuser1/zcee/zcee.server.p12'  
BINARY CONVERT(NO)
```

- If the z/OS Connect EE server is using a keystore:

- a. Locate the keystore for your z/OS Connect EE server. The default location of the keystore is `/var/zosconnect/servers/serverName/resources/security/keys.jks`
- b. Extract your server certificate from its keystore. Use the Java keytool to extract the certificate into a file. In the following example, the file is named `testserver.cer`.

```
keytool -export -alias default -file testserver.cer
-keystore keys.jks -storepass zosConnect -storetype jks
```

The message "Certificate stored in file `testserver.cer`" is displayed.

2. Transfer the certificate to your workstation by using FTP.
3. On your workstation, complete the certification configuration steps for your operation system.
 - For Windows, take the following steps to configure the certificates.
 - a. Open a command prompt or PowerShell, type `certmgr.msc`, and press **Enter**.
 - b. Right-click **Trusted Root Certification Authorities** and select **All Tasks > Import**.
 - c. Click **Next** when the **Certificate Import Wizard** appears.
 - d. On the "File to Import" page, click **Browse** to select and open the certificate file (created in Step 1).
 - e. Click **Next** to advance to the **Certificate Store** page.
 - f. Verify that **Place all certificates in the following store** and **Trust Root Certification Authorities** are both selected, and click **Next**.
 - g. Examine the details of the import. If all is correct, click **Finish**.
 - h. When prompted if you want to continue given that the certificate cannot be verified (because it is self-signed), click **Yes**.
 - i. Restart the API toolkit and any web browsers that might be running.
 - For MacOS, take the following steps to configure the certificates.
 - a. Open the Keychain Access application.
 - b. Select **System** in the Keychains view.
 - c. Select **Certificates** in the Category view.
 - d. Select **File > Import items**.
 - e. Specify the certificate file (created in Step 1). You might be prompted for your user name and password to allow modification of the system keychain.

The certificate is now displayed in the list, with the CN value of the certificate as its name (for example, `testserver.example.org`).

 - f. Double-click the certificate to update its settings.
 - g. In the Trust section of the displayed dialog, for the When using this certificate field, select **Always Trust** and close the dialog. You might be prompted for your user name and password to allow modification of the certificate settings.
 - h. Restart the API toolkit and any web browsers that might be running.

Related information

["Configuring Cross-Origin Resource Sharing on a z/OS Connect Enterprise Edition Server" on page 300](#)

Editing an existing API

To edit an existing API in the editor, double-click the package `.xml` file in your project folder in the **Project Explorer** view. To edit the request mapping or response mapping for a method, double-click the `request.map` or `response.map` file for the method in the `/api` folder of your project.

Re-importing changed services

If a service is modified after an API is created, when it is re-imported into the project, the impact of the changes is analyzed. The impact is reported, and mapping problems are recorded in the **Problems view**.

About this task

You can re-import one or more services in one of two ways:

- Click **Service** next to a method.
- Right-click the API project in the **Project Explorer** and click **z/OS Connect EE > Import z/OS Connect EE Services**.

Important: All APIs in the project that are associated with the re-imported services are impacted.

In the impact analysis dialog that is displayed, examine the impact. If the impact on the existing mappings is as expected, click **OK** to re-import the services.

The impacts are reported in the following categories:

Errors

An error is a result of one of the following conditions:

- An existing mapping is broken. A broken mapping occurs when a field that was involved in an explicit Move or Assign transform is removed or renamed in the changed service. The broken mapping is highlighted in the mapping editor when you open the request or response mapping.
- An existing rule for a response is broken. A broken rule occurs when a service field is removed or renamed in the changed service. The broken rule is shown in the problem view after impact analysis.

Warnings

A warning is a result of one of the following conditions:

- A field that was not explicitly involved in any Move or Assign transform but was implicitly passed through the request or response body is no longer available in the changed service.
- The data type of a field is changed.
- A new field is added.

Mapping and rules related issues are recorded in the Eclipse built-in **Problems view** to assist problem resolution.

Generating an API archive in the API toolkit

You can export your API project into an API archive (.aar) file from the API toolkit for deployment to the designated API directory. In an automated DevOps process, the API archive is usually generated by using the build toolkit.

Before you begin

Make any final edits to the Swagger document. For example, you might want to add security and authentication requirements. Manual changes to the Swagger document are preserved when saved.

Take the following steps to generate the API archive from the API toolkit. For automated DevOps process by using the build toolkit, see

Procedure

1. In the **Project Explorer** view, right-click an API project and click **z/OS Connect EE > Export z/OS Connect EE API Archive**.
2. Specify the location where you want to save this API archive file. The default file name is the API name with a file extension of .aar.

Results

The API archive file is saved as a .aar file.

What to do next

Transfer the file to the API directory for your z/OS Connect EE server. For more information, see “[Automated API management](#)” on page 673.

Note: In a DevOps environment, the API archive is generated by using the build toolkit with the API project directory as input. For more information about the build toolkit, its **zconbt** command syntax, and samples, see “[zconbt command syntax](#)” on page 795.

Related tasks

[Deploying an API using the API Deployment utility](#)

Developing applications with Swagger documents

The Swagger document that is generated by the z/OS Connect EE API toolkit can be examined and exposed in various ways.

A range of third-party tools are available for you to use with Swagger documents.

Swagger UI

With Swagger UI, you can visualize and test a REST API that is defined with Swagger from any web browser. The built-in testing functions let you graphically explore the APIs, test the APIs, and inspect the results. For more information, see [GitHub: Swagger UI](#).

Swagger Codegen

With Swagger Codegen, you can generate a Software Development Kit (SDK) in various languages, including Java, Objective-C, PHP, and Python, from a Swagger document for a REST API. You can use the resulting SDK to test the API in real time with a generated sample server implementation before you fully implement the API. For more information, see [GitHub: Swagger Code Generator](#).

The [editor.swagger.io](#) site, for example, provides Swagger Editor, Swagger UI, and Swagger Codegen.

Documenting your API using Swagger

When you have created your API, you can expose the API documentation to users.

You can provide this documentation with Swagger. For more information, see [Introduction to Swagger](#). For details in the Swagger specification, see [Swagger RESTful API Documentation Specification](#).

A RESTful administration interface for an API is available on the URI <apiBasePath>/api-docs and returns the Swagger document for the API, which is typically generated using the z/OS Connect EE tooling. API Swagger documents are only returned if the API has been correctly initialized:

HTTP method

GET

URI

/<apiBasePath>/api-docs

Note: The only allowed methods for the URI are GET and OPTIONS.

Return body

Returns the Swagger document as a JSON string with an HTTP status code of 200 OK.

The host and port values are added to the returned Swagger document, and override a *host* entry specified in the Swagger document on disk. The host name and port are derived from the request string.

Errors

For any call using a HTTP method other than GET and OPTIONS, a HTTP status code of 405 Method Not Allowed is returned.

If the <apiBasePath> does not exist or cannot be accessed, a HTTP status code of 404 Not Found is returned.

For more information, see [Swagger RESTful API Documentation Specification](#).

Generating API archives for DevOps

Use the build toolkit **zconbt** command to generate the API archive from API projects that are created in the API toolkit.

Before you begin

Ensure that the z/OS Connect EE build toolkit is installed. For more information, see [“Installing the z/OS Connect EE build toolkit” on page 199](#).

About this task

An API archive (.aar) file contains the information that is needed by z/OS Connect EE to provide the API. The API project directories can also be checked in to source control management system and therefore automatically built and deployed as part of a DevOps pipeline. You can use the build toolkit to build the API projects into deployable archive files and use scripts to copy the files to the API directory.

Procedure

1. Obtain the latest API source files from your source control management system and put them in a location that the **zconbt** command can access.
2. Use the **zconbt** command to generate the API archive. Use the **--projectDirectory** parameter to specify the project directory, and **--file** parameter to specify the file name for the generated API archive.

For example:

```
zconbt --projectDirectory=C:/APIs/purchaseAPI --file=C:/APIs/purchaseAPI.aar
zconbt --projectDirectory=./u/apiProjects/lookupAPI --file=./u/AARs/
lookup.aar
```

Note: If you are running on Microsoft Windows, use either a double backslash \\ or a forward slash character / in the path name.

Note: If you are running on z/OS, you can use the sample JCL provided in <hlq>.SBAQSAM (BAQZCBT) to run the zconbt.zos command. You must customize the sample JCL following the instructions in BAQZCBT.

3. If any errors occur, make the necessary corrections and repeat the procedure.
4. You can now deploy the API archive file by copying it to the designated API directory.

Results

An API is automatically deployed when the API archive is copied into the API directory if the server is configured to monitor the API directory for changes.

What to do next

For more information about API archive management, see [“Automated API management” on page 673](#).

Related concepts

[“DevOps with z/OS Connect EE” on page 14](#)

Identify a DevOps process before you start your development to automate the development and deployment of services, APIs, and API requesters for continuous integration and delivery.

Related reference

[“zconbt command syntax” on page 795](#)

The **zconbt** command starts the build toolkit tool. You can use the build toolkit to generate archive files for services, APIs or API requesters.

Configuring the API toolkit

Create a host connection to the z/OS Connect EE server to view, deploy, start, or stop a service or an API. Configure the preferences for default behaviors for host connections, service deployment, API deployment, and logging.

Connecting to a z/OS Connect EE server

Create a host connection to the z/OS Connect EE server to view, deploy, start, or stop a service or an API.

Before you begin

Switch to the **z/OS Connect Enterprise Edition** perspective.

About this task

In the **Host Connections** view, you can add connections to z/OS Connect EE servers and credentials to store your user IDs and passwords.

Tip: If you don't see the **Host Connections** view, from the menu bar, click **Window > Manage Connections** to open the **Host Connections** view.

When the z/OS Connect EE host connection successfully establishes a connection to the z/OS Connect EE server, it attempts to retrieve the Open API documentation for the z/OS Connect EE RESTful administration interface (`zosConnect/api-docs`) to determine which operations are supported by the z/OS Connect EE server. It then uses the RESTful administration interface to obtain a list of deployed APIs and services, and their status. An entry for the server, including a list of deployed APIs and services, is displayed in the z/OS Connect EE **Servers** view.

Note: If the authorization interceptor is configured in the z/OS Connect EE server, then the user ID specified on the z/OS Connect EE host connection must be authorized to list all APIs and services and get details of individual APIs and services. For more information about the required authorization for these actions, see [“Using the RESTful administration interface” on page 675](#) and [“Administering services with the administration interface” on page 655](#).

Procedure

1. In the **Host Connections** view, click **Add** and select **z/OS Connect Enterprise Edition**.
2. Specify the following information:

Table 78. Adding a server connection

Field	Description
Name	A descriptive name for the server connection.
Host name	The name or the IP address of the server.
Port number	The port number. Select the Secure connection (TLS/SSL) checkbox for secure connections.
Connection timeout	The amount of time in milliseconds the API toolkit waits for a successful connection to be established with the z/OS Connect EE server. The default is 30 seconds unless specified otherwise in the z/OS Connect EE preferences window. A value of 0 indicates to wait forever.

Table 78. Adding a server connection (continued)

Field	Description
Read timeout	The amount of time in milliseconds that the API editor reads response data from the z/OS Connect EE server. The default is 30 seconds unless specified otherwise in the z/OS Connect EE preferences window. A value of 0 indicates to wait forever.

Tip: You can change the default connection timeout and read timeout values. From the main menu bar, click **Window > Preference > z/OS Connect EE**, and specify your default timeout values.

3. Click **Save and Connect** to save the definition and connect to the server.

You are prompted to either specify an existing credential to use, or create a new credential. To add a credential, click **Add** in the **Credentials** section. For more information, see [Defining connection credentials in IBM Explorer for z/OS documentation](#).

Note: If client authentication is enabled on the server, this user credential is used for authorization. A trusted client certificate must be configured for user authentication. For more information, see [“Configuring client certificates for server connections” on page 604](#).

4. After you select an existing credential or create a new one, click **OK** to use the credential to connect.

Results

The defined host connection shows up in the **z/OS Connect EE Servers** view. Deployed API archive files are listed under the **APIs** folder.

Related concepts

[“Setting preferences for the API toolkit” on page 606](#)

Configuring client certificates for server connections

Generate a client certificate and configure the Eclipse preferences to send the certificate from the API toolkit to the z/OS Connect EE server. Import the client certificate into the server truststore.

Before you begin

If SSL is enabled on the z/OS Connect EE server, and a trusted client certificate is not sent in for authentication, an HTTP 403 Forbidden error is returned when connecting to the server.

Note: The embedded Swagger UI supports only basic authentication. Client authentication is not supported.

To connect to the server from the API toolkit, generate a client certificate first and import it into the truststore on the server.

1. Generate your client certificate.

This certificate is sent to the server for authentication. Use a tool such as Keytool to create a keystore and then export the client certificate from the keystore. The following example shows the keytool command to create a keystore called `myclient.keystore.jks`:

```
keytool -genkey -alias myclient.cert -dname
"CN=API editor client Keystore, OU=IBM Systems z, O=IBM, C=US"
-keyalg RSA -keypass mypassword -storepass mypassword
-keystore <path_to>/myclient.keystore.jks
```

Then, export the client certificate, `myclient.cert.cer` from the client keystore:

```
keytool -export -alias myclient.cert -storepass mypassword
-file <path_to>/myclient.cert.cer -keystore <path_to>/myclient.keystore.jks
```

2. Transfer the client certificate to a location accessible to the z/OS Connect EE server.
3. On the z/OS Connect EE server, import the client certificate into the server truststore.

The following example shows the keytool command to import the client certificate into the server truststore.

```
keytool -import -v -trustclientcerts -alias apieditor.client  
-file myclient.cert.cer -keystore "<path_to>\server.truststore.jks"  
-keypass mypassword -storepass mypassword
```

4. Modify the `server.xml` file to ensure that the following information is specified.

- If the z/OS Connect EE server is configured to require client certificate authentication and TLS client authentication to provide confidentiality and integrity on the connection, then ensure that the `allowFailOverToBasicAuth` attribute of the `webAppSecurity` element is set to `false`.

For more information about configuring a z/OS Connect EE server to use TLS, see “[API provider confidentiality and integrity](#)” on page 336. For more information about using client-certificate authentication, see “[API provider authentication and identification](#)” on page 349.

Note: When you create an IMS service in IMS Explorer, the `allowFailOverToBasicAuth` attribute must be set to `true`.

Procedure

Configure the API toolkit with the client certificate.

1. Open the Preferences window by clicking **Window > Preferences** on the main menu.
2. Expand **Explorer** and click **Certificate management**.
3. In the Keystore details section, next to the File name field, enter the full path and name of the file where the certificates are saved. You can also click **Browse** to navigate to the client key, select the client key, and click **Open**.
4. In the Pass phrase field, enter the password for this keystore.
5. In the Store type field, select the correct store type.
6. If you are using a smart card, select **Use Windows cryptography services** for the Windows operating system, which uses the standard Windows cryptography mechanism. To use a PKCS11 driver (mandatory on Mac OS and Linux operating systems), select **Use PKCS11 driver** and specify the driver path and PIN.
7. If you are instructed by your network administrator, select the correct protocol for your organization in the **Secure socket protocol** field.
8. Click **Apply** and **OK** to save your settings and close the window.
9. Add a credential for the client certificate for host connections.
 - a) Click **Window > Manage Connections** to open the **Host Connections** view.
 - b) Click **Add** in the **Credentials** section and select **Certificate from Keystore**.
 - c) Specify the user ID that is associated with the certificate.
 - d) Choose the appropriate certificate from the list and click **OK**.
10. Select your z/OS Connect Enterprise Edition connection in the Connections section, right-click, and select **Set Credentials**.
11. Select the credential that you just defined, and click **Connect**.

Results

The client certificate is used for server connection authentication, allowing the server to use the user ID that is specified for the host connection for authorization based on the defined security role.

Related tasks

[“Connecting to a z/OS Connect EE server” on page 603](#)

Related information

[Defining connection credentials \(IBM Explorer for z/OS\)](#)

Setting preferences for the API toolkit

You can specify your preferences for default behaviors for host connections, service deployment, API deployment, and logging in the preferences window.

To set your preferences, from the menu bar, click **Window > Preferences > z/OS Connect EE**.

The API deployment and service deployment preferences apply to APIs and services that you will be deploying, updating, or removing. By default, APIs and services are started upon deployment and stopped before they are updated. The connection preferences apply to new connections that you will be creating.

You can override these default settings when you are creating individual host connections or deploying individual APIs or services.

Client application development

When z/OS Connect EE is successfully installed, you can create the artifacts that are needed for client applications to access your z/OS services.

Related concepts

[Auditing and tracking](#)

Retrieving service information

z/OS Connect EE provides an administration interface that you can use to discover services, retrieve details of a service, get service request or response schema, and complete other operations.

You can use the z/OS Connect EE administration interface from any client that is running in a web, mobile, or cloud environment. Specific authority (Admin, Operations, Reader, or Invoke) is required depending on the operations.

The following examples demonstrate some of the calls that can be made by using the administration interface.

- Discover services in the z/OS Connect EE configuration by using an HTTP **GET** request. For example,

```
https://<host>:<port>/zosConnect/services
```

- Retrieve details of a service. For example,

```
https://<host>:<port>/zosConnect/services/<service_name>
```

- Retrieve request or response schema.

```
https://<host>:<port>/zosConnect/services/<service_name>/schema/request
```

```
https://<host>:<port>/zosConnect/services/<service_name>/schema/response
```

Note: If **<service_name>** contains forward slashes, they must be escaped by using %2F. For example, if the service name is MyService/v1, it must be provided as MyService%2Fv1.

For more information about the supported methods and operations by the service administration interface, see [“Administering services with the administration interface” on page 655](#).

Conversion for z/OS Connect Enterprise Edition data transformation

You can use the **BAQLS2JS** and **BAQJS2LS** utilities to generate the necessary z/OS Connect Enterprise Edition artifacts to facilitate data conversion with the z/OS Connect Enterprise Edition data transformation provider.

The **BAQLS2JS** and **BAQJS2LS** utilities are provided for use with the supplied service provider for CICS WOLA.

Note: z/OS Connect EE supports JSON integer values in the range -2^{63} through $2^{63} - 1$. Any language structures that you use as input to the BAQLS2JS utility, which generates JSON schemas with "type": "integer" and values below -2^{63} or values above $2^{63} - 1$ are not supported. For example,

- For C, do not exceed long long. Unsigned long long is not supported.
- For COBOL, do not exceed PIC S9(18) COMP-5 or PIC S9(18) DISPLAY.
- For Enterprise PL/I, do not exceed SIGNED FIXED BINARY(63). UNSIGNED FIXED BINARY(64) is not supported.

If you are using existing JSON schemas as input to BAQJS2LS, ensure that any integer values do not exceed these limits.

For more information about the language structure to JSON mappings and JSON to language structure mappings, see [High-level language and JSON schema mapping](#) in the *CICS TS documentation* documentation.

BAQLS2JS: High-level language to binding and schema file conversion for z/OS Connect Enterprise Edition data transformation

The **BAQLS2JS** Job Control Language procedure generates a JavaScript Object Notation (JSON) schema and bind files from a high-level data structure. The files that are generated are used by the z/OS Connect Enterprise Edition data transformation process.

Note: If you are generating a WSBIND file and JSON schemas to use the z/OS Connect EE data transformation feature, you must use the naming convention that is required by the `zosConnectDataXform` element in the `server.xml` file. The names of the request and response schemas must be of the format that is shown in the following example. The suffixes `_request` and `_response` are mandatory. The following example uses this naming convention:

```
SERVICE-ARCHIVE={servicearchive_dir}/{servicearchive}
SERVICE-NAME={serviceName}
WSBIND={wsbind_dir}/{serviceName}.wsbind
JSON-SCHEMA-REQUEST={json_schema_dir}/{serviceName}_request.json
JSON-SCHEMA-RESPONSE={json_schema_dir}/{serviceName}_response.json
```

where

- `{serviceName}` must match the value that is specified on the **serviceName** attribute of the associated `zosConnectService` element when you configure `server.xml` file to use the `DataXform`.
- `{servicearchive_dir}` is the UNIX System Services directory in which the service archive file is created.
- `{wsbind_dir}` is the UNIX System Services directory in which the `wsbind` file is created.
- `{json_schema_dir}` is the UNIX System Services directory in which the JSON request and response schemas are created.

For example:

```
<zosconnect_zosConnectService id="inquireCatalogService"
    serviceName=<serviceName>
    serviceRef="wolaCatalogManager"
    dataXformRef="xformJSON2Byte" />
<!-- z/OS Connect data transformation provider -->
<zosconnect_zosConnectDataXform id="xformJSON2Byte"
    bindFileLoc=<wsbind_dir>
    bindFileSuffix=".wsbind"
    requestSchemaLoc=<json_schema_dir>
    responseSchemaLoc=<json_schema_dir>
    requestSchemaSuffix=".json"
    responseSchemaSuffix=".json">
</zosconnect_zosConnectDataXform>
```

Job control statements for **BAQLS2JS**

JOB

Starts the job.

EXEC

Specifies the procedure name (BPXBATCH).

Input Parameters

BAQLS2JS runs a version of the CICS JSON assistant utility program **DFHLS2JS**. For input parameter descriptions, see the [DFHLS2JS reference documentation](#). Where the documentation refers to **DFHLS2JS**, use **BAQLS2JS**.

Two additional input parameters can be used to create a service archive for z/OS Connect Enterprise Edition:

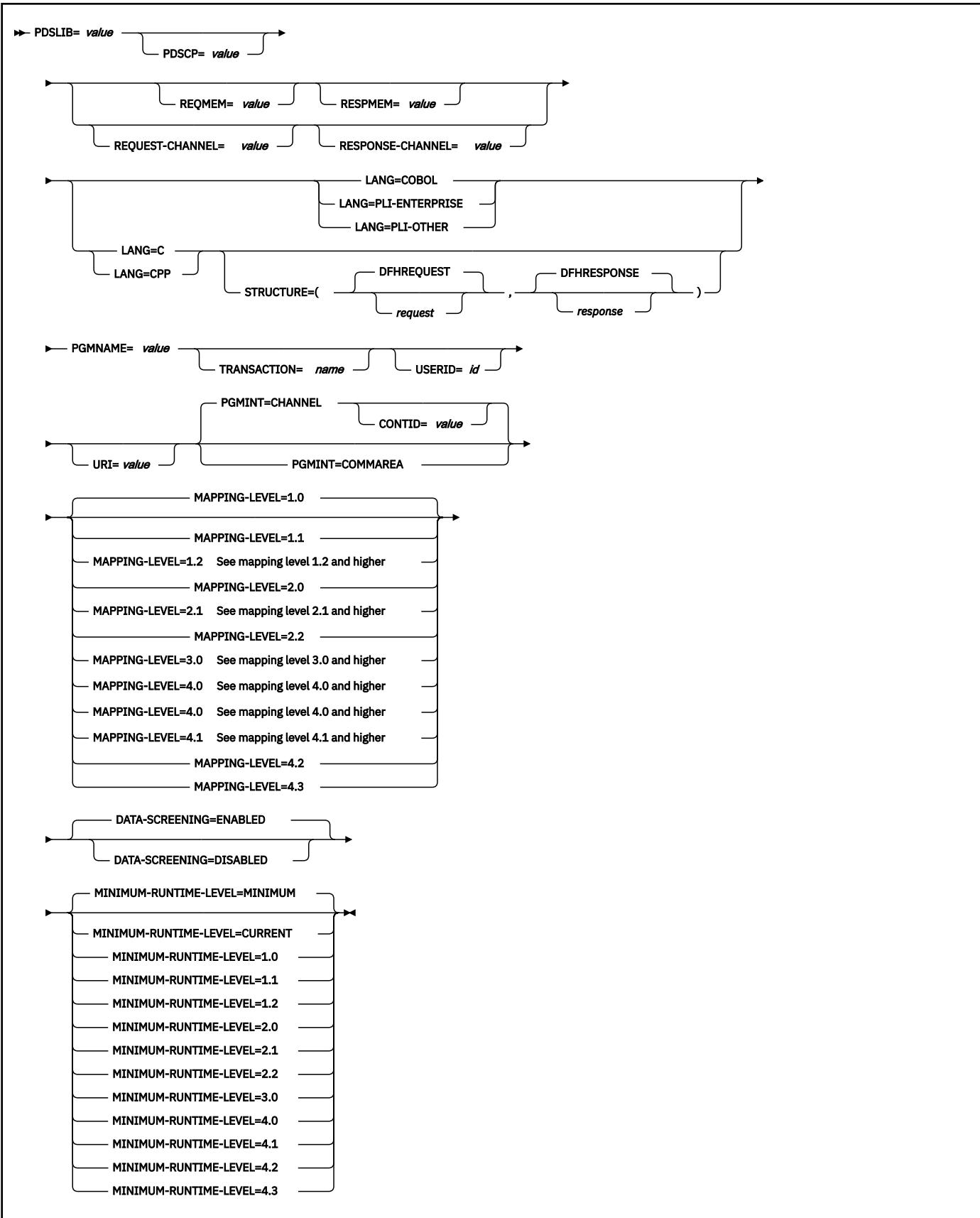
SERVICE-ARCHIVE

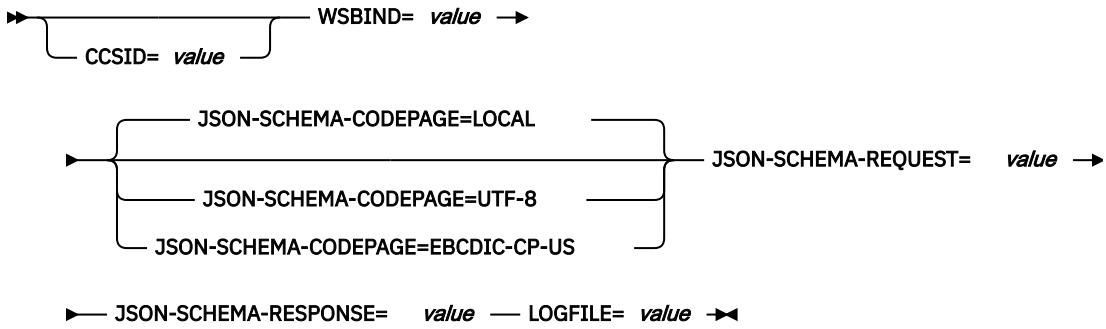
The fully-qualified z/OS UNIX name of the service archive file. **DFHLS2JS** creates the file, but not the directory structure, if it does not exist.

SERVICE-NAME

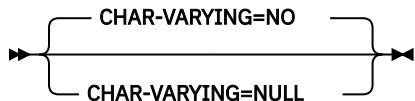
The name of the z/OS Connect Enterprise Edition service that is described by the service archive. This name is specified by the **SERVICE-ARCHIVE** parameter. If you develop APIs from the generated service archive files, the API toolkit uses the value that is specified on the **SERVICE-NAME** parameter as the service name associated with a specific API path and method.

The following input parameters are valid for **BAQLS2JS**. The value that is specified for the **PGMNAME** parameter is used in the generated JSON schema.

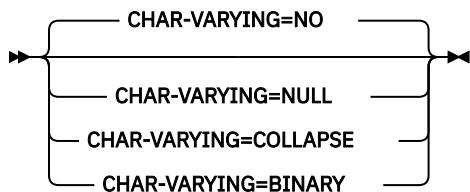




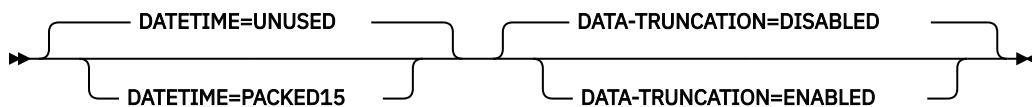
Mapping level 1.2 and higher



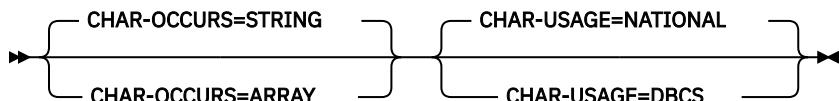
Mapping level 2.1 and higher



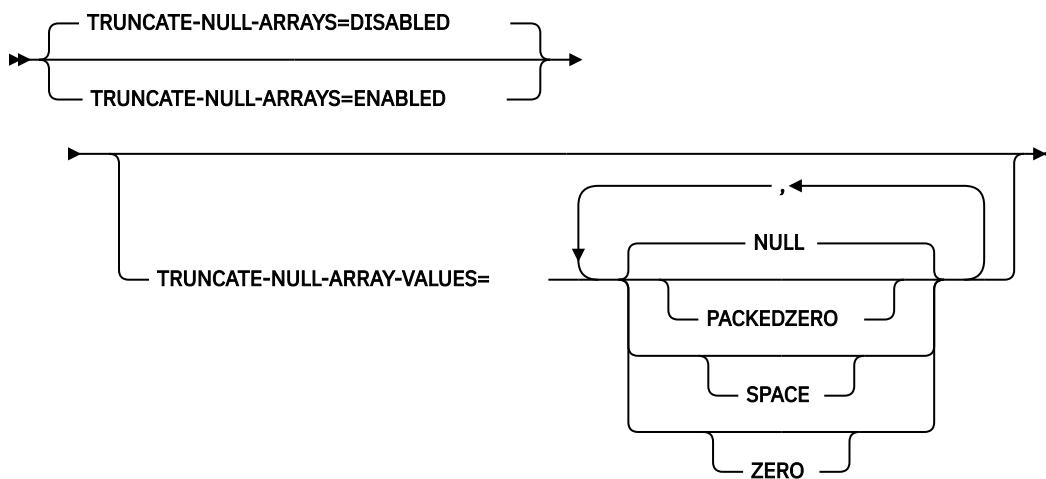
Mapping level 3.0 and higher



Mapping level 4.0 and higher



Mapping level 4.1 and higher



The following is sample JCL to run the **BAQLS2JS** tool. You can find this sample in the <hlq>.SBAQSAMP partitioned dataset.

```
//BAQLS2JS JOB (0),MSGCLASS=X,CLASS=A,NOTIFY=&SYSUID,REGION=500M
//ASSIST EXEC PGM=BPXBATCH
//STDPARM DD *
PGM /usr/lpp/IBM/zosconnect/v3r0/bin/baqls2js
LOGFILE=/u/userid/BAQLS2JS.log
LANG=COBOL
MAPPING-LEVEL=4.0
PDSLIB=/PROJECT.COBOL
REQMEM=REQLS
RESPMEM=RESPLS
JSON-SCHEMA-REQUEST=/u/userid/schema/inquireCatalog_request.json
JSON-SCHEMA-RESPONSE=/u/userid/schema/inquireCatalog_response.json
WSBIND=/u/userid/wsbind/inquireCatalog.wsbind
PGMNAME=DFH0XCMN
PGMINIT=COMMAREA
/*
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV DD *
JAVA_HOME=/java/java71_64
//
```

BAQJS2LS: JSON schema to high-level language conversion for z/OS Connect Enterprise Edition data transformation

The **BAQJS2LS** JCL procedure generates a high-level language data structure and binding file from a JSON schema. The files that are generated are used by the z/OS Connect Enterprise Edition data transformation process.

The value that is specified for the **PGMNAME** parameter is used in the name of the generated high-level language structure.

Note:

1. If you use the WSBind file to call CICS via an API, **JSON-SCHEMA-REQUEST** and **JSON-SCHEMA-RESPONSE** must use schemas that specify only a single property at the top level of the JSON format described.
2. If you are generating a WSBind file and JSON schemas for use in a z/OS Connect Enterprise Edition DataXform, you must use a naming convention that is expected by the `zosConnectDataXform` element in the `server.xml` file. Specify the following values for the **BAQJS2LS** parameters:

```
SERVICE-NAME={serviceName}
WSBIND={wsbind_dir}/{serviceName}.wsbind
JSON-SCHEMA-REQUEST={json_schema_dir}/{serviceName}_request.json
JSON-SCHEMA-RESPONSE={json_schema_dir}/{serviceName}_response.json
```

where

- `{serviceName}` must match the value that is specified on the **serviceName** attribute of the associated `zosConnectService` element when you configure `server.xml` file to use the `DataXform`
- `{wsbind_dir}` is the UNIX System Services directory in which the `wsbind` file is created.
- `{json_schema_dir}` is the UNIX System Service directory in which the JSON request and response schemas are stored.

Job control statements for BAQJS2LS

JOB

Starts the job.

EXEC

Specifies the procedure name (BPXBATCH).

Input parameters for BAQJS2LS

BAQJS2LS runs a version of the CICS JSON assistant utility program **DFHJS2LS**. For input parameter descriptions, see the [**DFHJS2LS** reference documentation](#). Where the documentation refers to **DFHJS2LS**, use **BAQJS2LS**.

Two additional input parameters can be used to create a service archive for z/OS Connect Enterprise Edition:

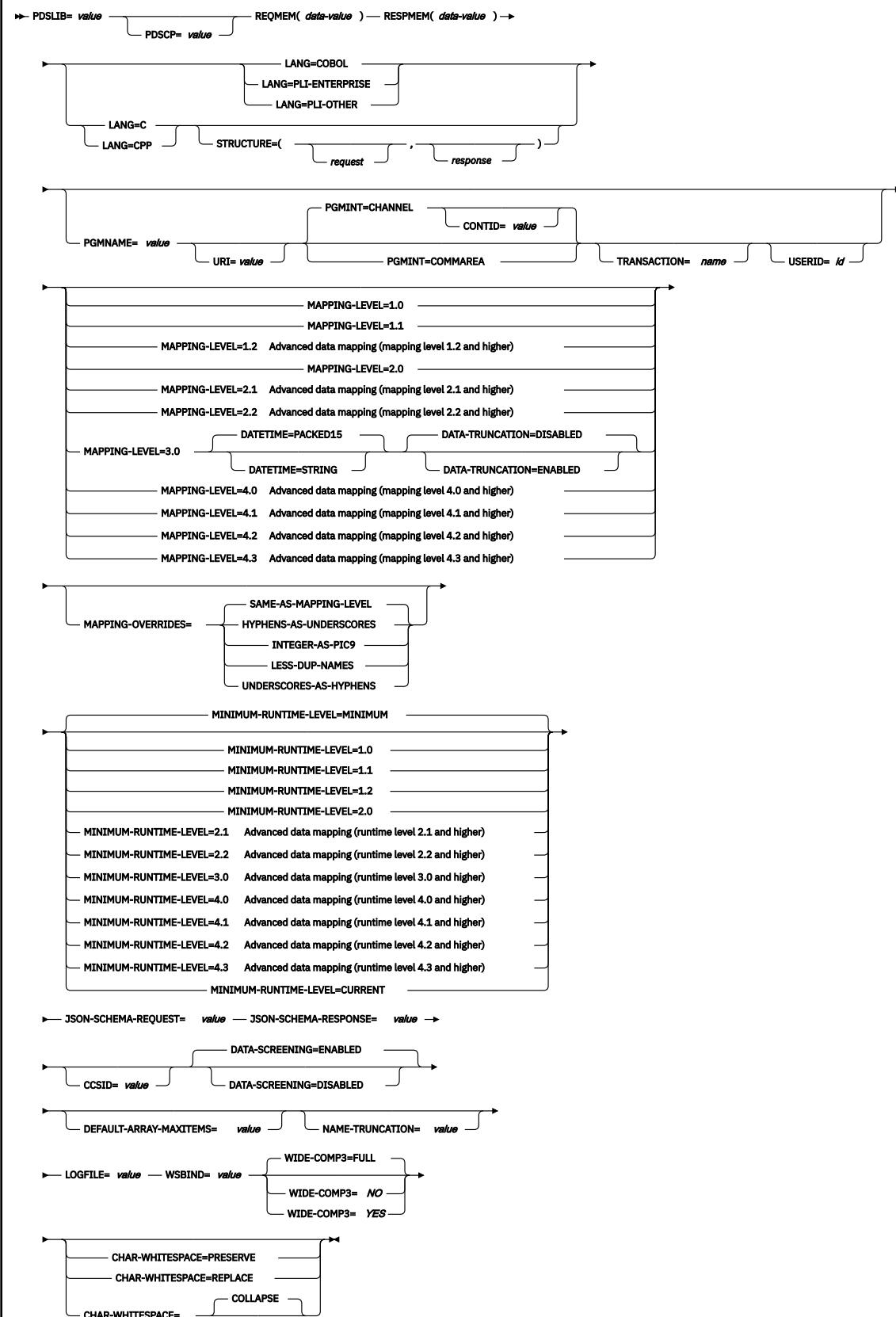
SERVICE-ARCHIVE

The fully-qualified z/OS UNIX name of the service archive file. **DFHJS2LS** creates the file, but not the directory structure, if it does not exist.

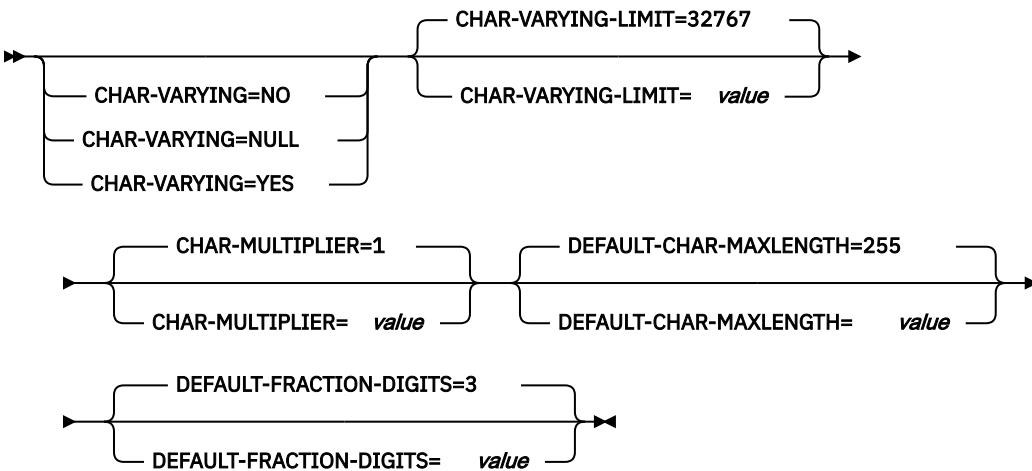
SERVICE-NAME

The name of the z/OS Connect Enterprise Edition service that is described by the service archive. This archive is specified by the **SERVICE-ARCHIVE** parameter. If you develop APIs from the generated service archive files, the API toolkit uses this value as the service name associated with a specific API path and method.

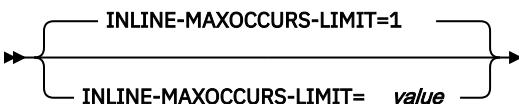
The following input parameters are valid for **BAQJS2LS**. The value that is specified for the **PGMNAME** parameter is used in the name of the generated high-level language structure.



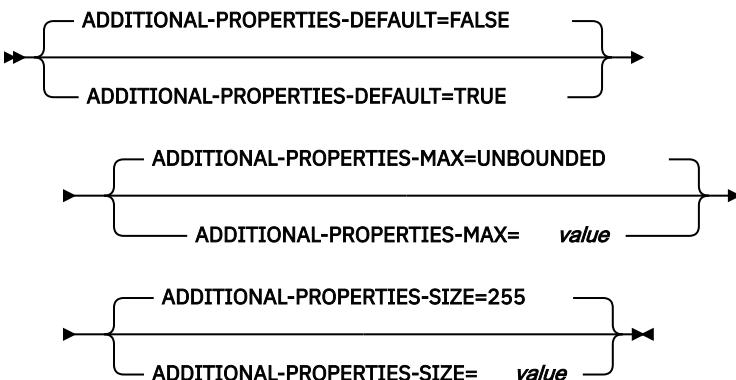
Advanced data mapping (mapping level 1.2 and higher)



Advanced data mapping (mapping level 2.1 and higher)



Advanced data mapping (mapping level 4.2 and higher)



Use this sample JCL to run the **BAQJS2LS** tool. You can find this sample in the `<hlq>.SBAQSAMP` partitioned dataset.

```

//BAQJS2LS JOB (0),MSGCLASS=X,CLASS=A,NOTIFY=&SYSUID,REGION=500M
//ASSIST EXEC PGM=BPXBATCH
//STDPARM DD *
PGM /usr/lpp/IBM/zosconnect/v3r0/bin/baqjs2ls
LOGFILE=/u/userid/BAQJS2LS.log
LANG=COBOL
MAPPING-LEVEL=4.0
PDSLIB=/PROJECT.COBOL
REQMEM=REQLS
RESPMEM=RESPLS
JSON-SCHEMA-REQUEST=/u/userid/schema/inquireCatalog_request.json
JSON-SCHEMA-RESPONSE=u/userid/schema/inquireCatalog_response.json
WSBIND=/u/userid/wsbind/inquireCatalog.wsbind
PGMNAME=DFH0XCMN
PGMINT=COMMAREA
/*
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV DD *
JAVA_HOME=/java/java71_64
//
```

Return codes

If the running utility program issues a non-zero return code, look in the job STDOUT and STDERR for messages about the cause of the problem. If no warning or error messages are found in STDOUT or STDERR, look for security-related messages in the system log of the LPAR. Fix the problem and rerun the utility program until a zero return code is obtained. Do not use artifacts that are generated with a non-zero return code.

Related concepts

[z/OS Connect EE API toolkit for defining your APIs](#)

Error responses

If the `useJsonErrorResponses` attribute on the `zosconnect_zosConnectManager` configuration element is not set, or set to `true`, all error responses from the z/OS Connect EE runtime are returned in JSON format.

For more information on the attribute, see “[zosconnect_zosConnectManager](#)” on page 784 in the Reference section. If the attribute is set to `false`, the error responses are a mixture of JSON and plain text, depending on the error case encountered.

If an error occurs while processing a request, an appropriate HTTP status code and the following JSON will be returned:

```
{  
    "errorMessage": "{message}",  
    "errorDetails": "{details}"  
}
```

Note:

- `errorDetails` is optional and will only be returned for some error scenarios.
- Service providers might use a proprietary error response format, so the format described above cannot be guaranteed when invoking an API or service.

For example:

A GET request to `/zosConnect/services/doesNotExist`, would return a HTTP status code of 404 and JSON body:

```
{  
    "errorMessage": "BAQR0431W: Service doesNotExist is not available."  
}
```

The following message is also written to the message log:

BAQR0403W: Service doesNotExist specified under URL `http://server:1234/zosConnect/services/doesNotExist` is not available.

A POST request to `/zosConnect/services/myService` with content-type set to `application/json` but an invalid body of: `notJson` would return a HTTP status code of 400 Bad Request and the following JSON body:

```
{  
    "errorMessage": "BAQR0417W: The request payload could not be parsed.  
    A JSON object format payload is expected. Error: Unexpected identifier 'notjson' on line 1, column  
    7."  
}
```


Chapter 21. Calling RESTful APIs from z/OS applications

Refer to this section when you want your z/OS applications to call RESTful APIs through z/OS Connect Enterprise Edition.

For step-by-step examples of calling an API from a System of Record (SoR), see:

- [“Call an API from a CICS application” on page 176](#)
- [“Call an API from an IMS or z/OS application” on page 184](#)

Using the build toolkit to generate artifacts for an API requester

You can use the build toolkit either from the command line interface or with the SDK.

Based on a Swagger document that is described in JSON or YAML format, you can use the build toolkit to generate all the artifacts to enable your z/OS application to call an API.

The following table shows the generated artifacts for an API requester and their functions:

Table 79. Generated artifacts for an API requester and their functions	
Generated artifact	Function
API requester archive file (.ara)	An archive file that can be deployed into the z/OS Connect EE run time. It contains the transformations so that z/OS Connect EE can convert the request payload from binary format to JSON format and convert the response payload from JSON format to binary format.
API information file	A data structure that contains variables that z/OS applications use when calling an API. Values of these variables are populated automatically based on the Swagger file to specify the name, path and method of an operation in the API. The data structure is generated for each operation in the API.
Data structure	Request and response data structures that are generated for each operation in the API.
Log	A log file for each operation defined in Swagger.
Summary report	A log file for each API that logs all the operations and generated artifacts of each API.

Note:

- The Swagger file must be encoded in UTF-8.
- The build toolkit can process most of the Swagger 2.0 fields and objects except the following limitations:
 - Limited combinations of type and format are now supported. For more information, see [“JSON schemas supported by the build toolkit” on page 627](#).
 - \$ref defined in path is not supported.
 - Parameters defined in form are not supported.

- Data transformation is not applied to the response with a non-2xx response code. The response body is directly returned to the z/OS application.
- If more than one 2xx response codes are defined in one operation, only the response data with the minimum response code can be transformed.
- \$ref to a file is not supported.
- Fields of consumes or produces that are populated with values containing no application/json are not supported.
- Not all the characters in the title and operation path of an API are accepted.

For the title of an API

- Characters that are accepted are uppercase characters (A-Z), lowercase characters (a-z), digits (0-9) and special characters - _ . ! ' , ; : @= & % \$ + * (and).
- Characters that are accepted but will be replaced with the hyphen character are # | \ \ ^ [] ~ ` < > ? / and blanks.

For the operation path of an API

Characters that are accepted are uppercase characters (A-Z), lowercase characters (a-z), digits (0-9) and special characters - _ . ! , ' % { } ; : @ = and &.

- Arrays that are wrapped up with more than 7 layers are not supported.

Generating artifacts for an API requester from the command line

Enter the **zconbt** command on the command line to generate the artifacts to call an API from your z/OS application.

Before you begin

- Ensure that the z/OS Connect EE build toolkit is installed. For more information, see “[Installing the z/OS Connect EE build toolkit](#)” on page 199.
- Prepare the Swagger file of the API that your z/OS application wants to call. The z/OS Connect EE build toolkit uses the Swagger file as an input to generate artifacts to call the API. For more information about the Swagger file and the generated artifacts, see “[Using the build toolkit to generate artifacts for an API requester](#)” on page 617.

About this task

Follow these steps to use the z/OS Connect EE build toolkit to generate the API information file, data structures, and API requester archive for a z/OS application to call a RESTful API. The following steps use an example of calling the Watson™ translator API from a COBOL application.

To learn more about the build toolkit, see “[The z/OS Connect EE build toolkit](#)” on page 13.

For best results, generate the data structures on the z/OS platform directly. If you want to run the build toolkit on a non-z/OS platform, set a non-z/OS code page, such as UTF-8, by specifying the **generatedCodePage** property in the build toolkit properties file and upload the generated data structures to the z/OS data set in ASCII mode via FTP.

If you use EDCCDIC when generating data structures on a non-z/OS platform and upload the data structures to a z/OS data set in Binary mode via FTP, the data structures might be corrupted because of the different newline characters in different platforms.

For more information about the **generatedCodePage** property, see “[The build toolkit properties file](#)” on page 621.

Procedure

1. Create directories to contain the generated artifacts from the build toolkit.

Based on the Swagger file that describes Watson translator API , you might create the directories as the following table shows:

Table 80. Directory paths and what they are created for

Path	What it is created for
./watson/	To store the Swagger file of the Watson translator API and the generated log file for each operation defined in the Watson translator API.
./watson/archives	To store the generated API requester archive file and the summary report for the Watson translator API.
./watson/structures	To store the generated data structures for each operation.
./watson/apiInfoFile	To store the generated API information file for each operation.

2. Create a properties file that is used as the input into the build toolkit.

The file contains name value pairs that include which Swagger file to use as input, the language of the data structures that are generated, and where the generated files are stored.

Here is an example of properties file based on the directories that were created in the previous step:

```
apiDescriptionFile=./watson/watson_translator_v2.json
dataStructuresLocation=./watson/structures
apiInfoFileLocation=./watson/apiInfoFile
requesterPrefix=API
connectionRef=BluemixAPIConnect
logFileDirectory=./watson
language=COBOL
```

A description of the properties is provided in [“The build toolkit properties file” on page 621](#).

Note:

- Ensure the Swagger file is encoded in UTF-8.
- The generated API requester archive is language specific. You must ensure the language that is specified in the properties file is the same as that used with your client z/OS application. For example, if the client z/OS application to call an API is developed in COBOL, you must specify the language attribute in the properties file as COBOL. Otherwise, an error occurs because the API requester archive is incompatible with your client z/OS application.

3. Use the **zconbt** command to generate API requester artifacts.

Use the **--properties** parameter to specify the properties file, and **--file** parameter to specify the file name for the generated API requester archive. For example:

```
zconbt --properties=watson.properties --file=./watson/archives/Watson.ara
```

Note: If you are running on Microsoft Windows, run the **zconbt.bat** command. If you are running on z/OS, run the **zconbt.zos** command.

4. If any errors occur, take the user action that is indicated in the error message to make the necessary corrections and repeat the procedure.

Results

Several artifacts for the API requester are created in the directories you have created. For each API, an API requester archive (.ara) file and a summary report are generated. For each API operation, an API information file, request data structures and response data structures are generated. The names of the generated request and response data structures and API information file are based on the

requesterPrefix value that is specified in the properties file. For more information about the names of the generated artifacts, see [“Naming conventions for the API requester artifacts” on page 621](#).

The following screen capture shows the expected API requester artifacts.

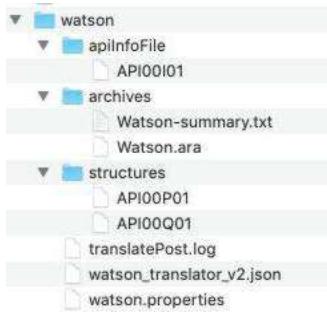


Figure 115. A screen capture of the expected API requester artifacts

Generating artifacts for an API requester with the build toolkit SDK

Use the build toolkit SDK to build your artifacts for the API requester.

Before you begin

Ensure that the z/OS Connect EE build toolkit is installed. For more information, see [“Installing the z/OS Connect EE build toolkit” on page 199](#).

About this task

Follow these steps to use the z/OS Connect EE build toolkit SDK to generate the API information file, data structures and API requester archive for a z/OS application to call a RESTful API. To learn more about the build toolkit, see [“The z/OS Connect EE build toolkit” on page 13](#).

Procedure

1. In your IDE, create a `Map<String, String>` object to contain the properties required to build the artifacts for the API requester.
2. Create a `ARAGenerator` object passing in the Map created in Step 1.
If any properties are invalid, an `InvalidPropertyException` occurs. The error message contains the property in error and the reason.
3. Save the `.ara` file.
Call the `save()` method and pass the filename.
4. You can now deploy the `.ara` file to z/OS Connect EE.

Example

```
Map<String, String> properties = new HashMap<String, String>();  
properties.put("apiDescriptionFile", "swagger.json");  
properties.put("dataStructuresLocation", "/tmp");  
properties.put("apiInfoFileLocation", "/tmp");  
properties.put("requesterPrefix", "API");  
properties.put("connectionRef", "test");  
properties.put("logFileDirectory", "/tmp");  
properties.put("language", "COBOL");  
  
ARAGenerator araGenerator = new ARAGenerator(properties);  
araGenerator.save("/example/output/dir/file.ara");
```

Naming conventions for the API requester artifacts

When generating the data structures and API information file for an operation in an API, the build toolkit names the generated artifacts in a specific pattern. You can control this naming convention. Be aware that the naming convention that you specify affects the maximum number of operations that can be generated from an API.

Name format

The names of the data structures and API information file are composed of four elements:

<**requestPrefix**><**operation_number**><**type**><**suffix**>

requesterPrefix

A prefix that is used in the names of the generated request data structure, response data structure and API information file. The parameter can be defined in the build toolkit properties file either from the command line interface or with the SDK. The value of this parameter has a maximum length of 3 characters. The length of this parameter affects the length of **operation_number** because the first two elements in the name have a total length of 5 characters. The first character must be either a letter or one of the following three special characters: #, @, \$. The remaining characters can be letters, numbers, or one of the following special characters: #, @, or \$. By default, **requesterPrefix** is set to API.

operation_number

A base-36 number padded after the **requesterPrefix** to identify an operation in the same API. The length of this element is affected by the length of **requesterPrefix** because the first two elements in the name have a total length of 5 characters. The length of this element determines the maximum amount of numbers that can be used to identify operations. The number of an operation is guaranteed to be the same every time you run the build toolkit for the same API description file.

Note: When you want to add a new operation in an API description file without changing the number to be generated for all the existing operations in the same API description file, you must add the new operation after all the existing operations.

type

One character reserved to identify the type of API requester artifact that is generated:

- Q represents the request data structure.
- P represents the response data structure.
- I represents the API information file.

suffix

A double-character number reserved by the build toolkit. By default, 01 is always used.

The build toolkit properties file

Define attributes in the build toolkit properties file before you run the build toolkit from the command line.

To use the build toolkit to generate your artifacts for an API requester from a Swagger document, the following attributes must be defined in the properties file.

Table 81. Mandatory attributes

Attributes	Description
apiDescriptionFile	Specify the location and name of the JSON or YAML Swagger document to be used to generate the data structures, API information file, and an API requester archive file. The Swagger file is not required to have a particular file type extension; the file contents dictate the file format. For more information about the Swagger files supported and the artifacts generated, see “ Using the build toolkit to generate artifacts for an API requester ” on page 617.
language	Specify the language for the code generation. COBOL The language for the code generation is COBOL. PLI-ENTERPRISE The language for the code generation is PL/I . Note: The language specified in the properties file must be the same as that used with your client z/OS application."
dataStructuresLocation	Specify where the generated data structure should be stored. For z/OS, dataStructuresLocation is a data set or a directory. For other supported platforms, dataStructuresLocation is a directory.
apiInfoFileLocation	Specify where the generated API information file should be stored. For z/OS, apiInfoFileLocation is a data set or a directory. For other supported platforms, apiInfoFileLocation is a directory.
logFileDirectory	Specify the directory where the log generated from the build toolkit is stored. For each operation, there is a log.

The following attributes are optional in the properties file:

Table 82. Optional attributes

Attributes	Description
connectionRef	Specify the name of the connection to identify the RESTful endpoint. If you do not set connectionRef in the build toolkit properties file, you must set connectionRef of the apiRequester element in the server.xml file; otherwise, an error will occur.

Table 82. Optional attributes (continued)

Attributes	Description
requesterPrefix	Specify a prefix for the generated request data structure, response data structure and API information file. The value of this parameter has a maximum length of 3 characters. The first character must be either a letter or one of the following three special characters: #, @, \$. The remaining characters can be letters, numbers, or one of the following special characters: #, @, or \$. By default, requesterPrefix is set to API.
runtimeCodePage	Specify the CCSID that is used at run time to encode character data in the application data structure. By default, runtimeCodePage is set to CP037. The CCSID list supported by the build toolkit is the same with the one supported by the CICS conversion program. For more information about the supported list, see CICS-supported conversions in the CICS Transaction Server for z/OS 5.4.0 documentation .
characterMultiplier	<p>Specify the number of bytes that is allowed for each character. The value of this parameter can be a positive integer in the range of 1 - 2,147,483,647. All nonnumeric character-based mappings, are subject to this multiplier. Binary, numeric, zoned, and packed decimal fields are not subject to this multiplier. By default, characterMultiplier is set =1.</p> <p>This parameter can be useful if, for example, you are planning to use DBCS characters where you might opt for a multiplier of 3 to allow space for potential shift-out and shift-in characters around every double-byte character at run time.</p> <p>When you set runtimeCodePage=1200 (indicating UTF-16), the only valid values characterMultiplier are 2 or 4. When you use UTF-16, the default value is 2. Use characterMultiplier=2 when you expect application data to contain characters that require 1 UTF-16 encoding unit. Use characterMultiplier=4 when you expect application data to contain characters that require 2 UTF-16 encoding units.</p> <p>Note: Setting characterMultiplier to 1 does not preclude the use of DBCS characters, and setting it to 2 does not preclude the use of UTF-16 surrogate pairs. However, if wide characters are routinely used then some valid values will not fit into the allocated field. If a larger characterMultiplier value is used, it can be possible to store more characters in the allocated field than are valid in the XML. Care must be taken to conform to the appropriate range restrictions.</p>

Table 82. Optional attributes (continued)

Attributes	Description
characterWhiteSpace	<p>Specify how white space in values of type string will be handled.</p> <p>COLLAPSE Leading, trailing, and embedded white space is removed and all tabs, new lines, and consecutive spaces are replaced with single space characters.</p> <p>REPLACE Any tabs or new lines are replaced with the appropriate number of spaces.</p> <p>PRESERVE Retains any white space in the data value.</p> <p>By default, characterWhiteSpace is set =COLLAPSE.</p> <p>Note: This parameter does not apply to fields with a format of date-time, uri, base64Binary or hexBinary, where white space is always collapsed.</p>
characterVarying	<p>Specifies how variable-length character data is mapped to the language structure.</p> <p>NO Variable-length character data is mapped as fixed-length strings.</p> <p>NULL Variable-length character data is mapped to null-terminated strings.</p> <p>YES Variable-length character data is mapped to a CHAR VARYING data type in PL/I. In COBOL variable-length character data is mapped to an equivalent representation that consists of two related elements: the data-length and the data.</p> <p>By default, characterVarying is set =YES.</p>
characterVaryingLimit	<p>Specifies the maximum size of binary data and variable-length character data that is mapped to the language structure.</p> <p>When characterVarying is set to NO or NULL the value of this parameter can be a positive integer in the range of 1 to 16777214</p> <p>When characterVarying is set to YES the value of this parameter can be a positive integer in the range of 1 to 32767.</p> <p>By default, characterVaryingLimit is set =32767.</p>

Table 82. Optional attributes (continued)

Attributes	Description
dataTruncation	<p>Specify whether variable length data is tolerated in a fixed-length field structure.</p> <p>DISABLED</p> <p>If the data is less than the fixed length that z/OS Connect EE is expecting, z/OS Connect EE rejects the truncated data and issues an error message.</p> <p>ENABLED</p> <p>If the data is less than the fixed length that z/OS Connect EE is expecting, z/OS Connect EE tolerates the truncated data and processes the missing data as null values.</p> <p>By default, dataTruncation is set =DISABLED.</p>
dateTimeFormat	<p>Specify how JSON date-time elements are mapped to the data structure.</p> <p>PACKED15</p> <p>The default is that any JSON date-time element is processed as a timestamp and is mapped to absolute format.</p> <p>STRING</p> <p>The JSON date-time element is processed as text.</p> <p>By default, dateTimeFormat is set =PACKED15.</p>
defaultCharacterMaxLength	<p>Specify the default array length of character data in characters for mappings where no length is implied in the JSON schema document. When characterVarying is set to YES the value of this parameter can be a positive integer in the range of 1 to 32767.</p> <p>When characterVarying is set to NO or NULL the value of this parameter can be a positive integer in the range of 1 to 16777214</p> <p>By default, defaultCharacterMaxLength is set =255.</p>
nameTruncation	<p>Specify how JSON names are truncated. nameTruncation can be set =LEFT RIGHT. By default, names are truncated from the right.</p>
generatedCodePage	<p>Specify the code page that is used for the generated language structure and API information file, where value is a runtimeCodePage number or Java code page number. If this parameter is not specified, the z/OS UNIX System Services code page is used. For example, you might specify generatedCodePage =037.</p>

Table 82. Optional attributes (continued)

Attributes	Description
wideComp3	<p>For COBOL only. Control the maximum size of the packed decimal variable length in the COBOL data structure.</p> <p>No</p> <p>The packed decimal variable length is limited to 18 when generating the COBOL data structure type COMP-3. If the packed decimal size is greater than 18, message DFHPI9022W is issued to indicate that the specified type is being restricted to a total of 18 digits.</p> <p>Yes</p> <p>The maximum size of 31 is supported when generating the COBOL data structure type COMP-3.</p> <p>By default, wideComp3 is set=Yes.</p>
defaultArrayMaxItems	<p>Specify the maximum array boundary to apply where no maximum occurrence information (maxItems) is implied in the Swagger. The value of this parameter can be a positive integer in the range 1 - 32767. By default, defaultArrayMaxItems is set to 255.</p>
ignoreMIMETypeCheck	<p>Specify whether the MIME type check should be skipped. You can set ignoreMIMETypeCheck=true to allow the MIME type check to be skipped when you used the customized MIME type in the fields of consumes or produces. By default, ignoreMIMETypeCheck is set=false to enable the MIME type check.</p>
useMIMEType	<p>Specify whether the MIME type set in the consumes or produces fields in the Swagger document will be used to populate the Content-Type and Accept request headers and to validate the Content-Type response header. By default, useMIMEType is set =false to disable the use of the Swagger-defined MIME types. When useMIMEType is set =false, the default values for the Content-Type and Accept request headers will be "application/json"; the expected Content-Type in a response header will be "application/json".</p> <p>Note: If useMIMEType=true and the consumes field is not specified or is empty, the Content-Type and Accept headers will not be present on the request. If useMIMEType=true and the produces field is not specified or is empty, a response that contains a Content-Type header will result in an error. When useMIMEType=true, the Swagger document should fully define the consumes and produces for all methods or the endpoint should be configured to handle the situation where headers are not present.</p>

Table 82. Optional attributes (continued)

Attributes	Description
additionalPropertiesSize	Specify the maximum size for each of the JSON additional properties that are not explicitly defined in the properties section of a Swagger file. The value of this parameter can be a positive integer in the range of 16 - 32767. By default, additionalPropertiesSize is set to 255. A field of the size specified by additionalPropertiesSize will be generated into the data structures that are produced by the build toolkit. The generated data structures have bindings to support 20 kinds of additional properties.
apiKeyParmNameInHeader	Specify the name of an API key that is sent as a request header. The value of this parameter can be set in a comma separated list of a combination of client ID and client secret. For example, you can set apiKeyParmNameInHeader =header-IBM-Client-ID, header-IBM-Client-secret when a client ID and a client secret are used to protect an API.
apiKeyParmNameInQuery	Specify the name of an API key that is sent in a query string. The value of this parameter can be set in a comma separated list of a combination of client ID and client secret. For example, you can set apiKeyParmNameInQuery =query-IBM-Client-ID, query-IBM-Client-secret when a client ID and a client secret are used to protect an API.
apiKeyMaxLength	Specify the maximum length of the values set for API keys. The value of this parameter can be a positive integer in the range 1 - 32767. By default, apiKeyMaxLength is set to 255.
addLanguageSuffix	Specify whether the API requester name should be suffixed by the language used for code generation. When you set addLanguageSuffix=true , the same JSON or YAML Swagger document can be used to generate API requesters for both COBOL and PLI-ENTERPRISE, ensuring that both requesters have unique names, so they can be deployed to the same z/OS Connect EE server. When language=COBOL is used, the suffix is _cbl. When language=PLI-ENTERPRISE is used, the suffix is _pli. The value of this parameter can be true or false. The default is false.
defaultFractionDigits	Specify the default number of fraction digits to use on a JSON decimal schema type. For COBOL, the valid range is 0-17, or 0-3 if parameter wideComp3 is being used. For PLI, the valid range is 0-30. By default, defaultFractionDigits is set to 3.

JSON schemas supported by the build toolkit

Refer to this topic for the JSON schemas that z/OS Connect EE build toolkit supports for Swagger 2.0 file.

The following table shows the combinations of data type and data format that z/OS Connect EE build toolkit supports for Swagger 2.0 file.

Table 83. Data type and data format supported

Data type	Data format
integer	int32
integer	int64
number	float
number	double
number	decimal
string	
string	date-time
string	uri
String	email
String	hostname
String	ipv4
String	ipv6
Boolean	

The following table shows the combinations of data type and data format that z/OS Connect EE build toolkit doesn't fully support for Swagger 2.0 file. The formats of the items listed in Table 2 are just ignored by the build toolkit without any processing.

Table 84. Data type and data format not fully supported

Data type	Data format
string	byte
string	binary
String	date
String	password

There are some other limitations on the data type and keyword that z/OS Connect EE build toolkit can process for Swagger 2.0 files:

- Data type as file is not supported.
- Array parameters in the HTTP request or response body are supported, while array parameters not in the HTTP request or response body are handled as string.

Establishing a connection from z/OS Connect EE to the request endpoint

To establish a connection from a z/OS Connect EE server to a request endpoint you must add the `zosconnect_endpointConnection` element to the `server.xml` file.

The following excerpt from the `server.xml` shows how to define the `zosconnect_endpointConnection` element and how it is associated with the `zosconnect_apiRequesters` element.

```
<featureManager>
  <feature>zosconnect:apiRequester-1.0</feature>
```

```

</featureManager>

<zosconnect_endpointConnection id="watson-api-explorer"
    host="https://watson-api-explorer.myblumix.net"
    port="80"/>

<zosconnect_apiRequesters
    location="/var/zosconnect/servers/zosconnect/resources/zosconnect/apis">
    <apiRequester name="Book_Inventory" connectionRef="watson-api-explorer"/>
</zosconnect_apiRequesters>

```

In this example, when the Book_Inventory API requester is called by the CICS, IMS or z/OS application, the z/OS Connect EE server establishes a connection to the request endpoint with the host name and port number specified in the `zosconnect_endpointConnection` element.

The `zosconnect_endpointConnection` element can be referenced by the `connectionRef` attribute in the `apiRequester` sub element or associated with the `connectionRef` attribute in the build toolkit properties file. For more information about the `connectionRef` attribute in the `apiRequester` sub element and in the build toolkit properties file, see “[zosconnect_apiRequesters](#)” on page 754 and “[The build toolkit properties file](#)” on page 621.

Moving API requester artifacts between environments for DevOps.

You can move your API requester artifacts between environments such as Development, Test, and Production without changing the `.ara` configuration files. The attributes of the API requester artifacts are stored in the `.ara` files while environment attributes are specified in the `server.xml` configuration file. This means that API requester artifacts can be moved from one environment to another without the need to regenerate the artifact. The artifact uses *connection by reference*, so if you have three environments, development, test, and production, you define a connection reference with the same ID in each environment. The connection reference defines the connection to the System of Record (SoR) for that environment. The API requester artifact is configured to use the ID of the connection reference. Because the ID is the same in each environment, the API requester artifact can be moved without regeneration from one environment to another even though each environment uses a different SoR.

One HTTP endpoint is used by many ARA files because a single API gateway is used for the API requests. More than one HTTP endpoint might be needed if, for example, you have an internal and external gateway. In that case, create an HTTP endpoint for each API gateway and use the same `HTTPEndpoint` IDs in each environment to avoid having to regenerate the ARA as it moves through your DevOps process.

Advanced settings for the connection

The following table lists the advanced settings that you can set when you establish the connection between the z/OS Connect EE server and the request endpoint.

Table 85. Advanced settings for the connection	
Advanced setting task	Comments
Setting timeout values	Use the <code>connectionTimeout</code> and <code>receiveTimeout</code> attributes to set timeout values for the connection.
Setting a proxy	Use the <code>proxyConfigRef</code> attribute and the <code>zosconnect_proxyConfig</code> element to set a proxy to route the request for an API.
Connection pooling	Enable connection pooling in the z/OS Connect EE server to improve the efficiency of resource access.
Setting the domain base path	Use the <code>domainBasePath</code> attribute to specify an additional domain path in a URL for an API.

Setting timeout values

You can set the timeout for the z/OS Connect EE server to connect and wait for a response from the request endpoint by setting the `connectionTimeout` and `receiveTimeout` attributes in the `zosconnect_endpointConnection` element.

The following excerpt from the `server.xml` configuration file shows how to define the timeout configuration for the `watson-api-explorer` connection endpoint.

```
<zosconnect_endpointConnection id="watson-api-explorer"
    host="https://watson-api-explorer.mybluemix.net"
    port="80"
    connectionTimeout="30s"
    receiveTimeout="60s"/>
```

Setting a proxy

You can allow requests to be routed from the z/OS Connect EE to a request endpoint via a proxy by referencing the `zosconnect_proxyConfig` element.

The following excerpt from the `server.xml` shows how to define the proxy configuration for the `watson-api-explorer` endpointConnection.

```
<zosconnect_endpointConnection id="watson-api-explorer"
    host="https://watson-api-explorer.mybluemix.net"
    port="80"
    basicAuthRef="watson-api-explorer-basicAuthRef"
    proxyConfigRef="watson-api-explorer-proxyConfigRef"/>

<zosconnect_proxyConfig id="watson-api-explorer-proxyConfigRef"
    host="192.168.1.1"
    port="7777"
    type="HTTP" />
```

For more information about the `zosconnect_proxyConfig` and `zosconnect_endpointConnection` elements, see “[zosconnect_proxyConfig](#)” on page 774 and “[zosconnect_endpointConnection](#)” on page 765 in the Configuration elements section.

Connection pooling

You can use connection pooling in the z/OS Connect EE server to improve the efficiency of resource access to the request endpoint.

When you set up connection pooling, instead of closing the HTTP connection after use, z/OS Connect EE keeps the connection open and stores it in a pool in a dormant state. The dormant connection can be reused by the same API requester or by another API requester that connects to the same host and port. When an API requester issues a command to open an HTTP connection, z/OS Connect EE checks whether a dormant connection is available in the pool for that host and port, and if so, uses it rather than opening a new connection.

When a pooled connection is reused, CPU use is reduced compared with opening a new connection. An additional saving occurs where connections use TLS, because the security measures do not need to be repeated when the connection is reused. z/OS Connect EE reuses a pooled connection in exactly the same way as a new connection, and the connection can be pooled again after use.

To enable connection pooling in the z/OS Connect EE server, you must set the `http.keepAlive` JVM option to `true`. By default, the value of the `http.keepAlive` option is set to `true` in HTTP 1.1 and `false` in HTTP 1.0. You can also specify the maximum number of HTTP connections that you can create in the connection pools by configuring the `http.maxConnections` JVM option. By default, the value of the `http.maxConnections` option is set to 5.

You can change the value of the `http.keepAlive` and `http.maxConnections` options in the following two ways:

- Set the values in a .options file.
 1. Create a z/OS UNIX EBCDIC file called <name>.options in the <WLP_USER_DIR>/servers directory, where <name>.options is the name of the file that contains the JVM options.
 2. Enter the JVM options on separate lines in the <name>.options file, for example:

```
-Dhttp.keepAlive=true
-Dhttp.maxConnections=20
```

3. Add the following to the STDENV statement of the started task procedure for your z/OS Connect EE server:

```
JVM_OPTIONS=-Xoptionsfile=<WLP_USER_DIR>/servers/<name>.options
```

- Set the values in the JCL. For example:

```
JVM_OPTIONS=-Dhttp.maxConnections=3 -Dhttp.keepAlive=true
```

You can also specify the maximum number of persistent requests that are allowed on a single HTTP connection by configuring the maxKeepAliveRequests option in the Liberty configuration profile. By default, the value of the maxKeepAliveRequests option is 100. After 100 requests, the HTTP connection and the underlying socket are closed. To enable an unlimited number of the persistent requests, you must set the value of the maxKeepAliveRequests option to -1.

Setting the domain base path

A domain base path allows you to customize the URL for an API. For example, this allows you to distinguish between a development and a production environment.

The syntax for a URL including the domain base path is as follows:

```
http(s)://[host]:[port]/[domainBasePath]/[basePath]/[path]
```

The basePath and path are set in the API information file that is generated by the Swagger document.

To add the domain base path to a URL, you must set the domainBasePath attribute in the zosconnect_endpointConnection element. For example:

```
<zosconnect_endpointConnection id="watson-api-explorer"
    host="https://watson-api-explorer.myblumix.net"
    port="80"
    domainBasePath="/development/dev01"/>
```

This will result in a URL value of https://watson-api-explorer.myblumix.net/development/dev01/language-translator/v3/translate, where

- host is watson-api-explorer.myblumix.net
- port is 80
- domain base path is /development/dev01
- base path is language-translator and the path is v3/translate, which are both defined in the Swagger document.

Deploying the API requester to z/OS Connect EE

Deploy and undeploy your API requesters automatically when you add or remove API requester archive (.ara) files from the API requester folder.

You can configure z/OS Connect EE to deploy an API requester automatically when you copy your API requester archive (.ara) file to the location where your API requesters are stored. For example, resources/zosconnect/apiRequesters. For more information about configuring your z/OS Connect EE to support API requesters, see “Configuring z/OS Connect EE to support API requesters” on page 304.

When you remove the API requester archive (.ara) file from the folder, it is automatically undeployed.

The API requester location is monitored for changes to the API requester. You can control how and when z/OS Connect EE reacts to these changes by setting the *updateTrigger* and *pollingRate* attributes of the **zosconnect_apiRequesters** element of the configuration file `server.xml`. The default value for *pollingRate* is 5 seconds. For more information, see “[zosconnect_apiRequesters](#)” on page 754.

When you copy an API requester archive (.ara) file to the API requester folder, ensure that the API requester name is unique within the folder. For example, if `test1.ara` exists in the folder and contains the API requester `test`, it must be removed before you copy `test2.ara`, which also contains an API requester named `test`. If both files are present when z/OS Connect EE starts, one of the API requesters is deployed. When the other file is read, an error message is returned that the API requester is already deployed.

To update an API requester, replace the API requester archive (.ara) file in the API requester folder. The file must have the same name. If the *updateTrigger* and *pollingRate* attributes of the **zosconnect_apiRequesters** element are set, the API requester is installed automatically when the folder is polled.

For more information about the API requesters deployed in the z/OS Connect EE, see [Chapter 24, “How to manage API requesters,” on page 693](#).

Developing a z/OS application to call APIs

You can develop CICS, IMS and other z/OS applications to call RESTful APIs. Both COBOL and PL/I languages are supported.

To communicate with the z/OS Connect EE server and call an API, you must modify your z/OS application to include the required data structures, prepare the data for the API request, call the communication stub, and handle the response.

Before you begin

Use the build toolkit to generate the API requester artifacts, follow the instructions in “[Generating artifacts for an API requester from the command line](#)” on page 618.

Note:

1. For COBOL applications, RTEREUS must be set to off as it is unsupported when calling BAQCSTUB.
2. For long running API requester applications that use the IMS or z/OS communication stub, a call to the BAQCTERM function is required to close and clear the cached connection used by z/OS Connect EE. Follow step “[7](#)” on page 635 to ensure that your application BAQCSTUB connections are terminated after the application module for your program is unloaded.
3. z/OS applications should be compiled with the RENT option as this is required by both z/OS Connect EE and the Web Toolkit that it uses.

About this task

To call an API from your z/OS application, you must include the following data structures in your z/OS applications:

- The request data structure that contains the request message.
- The response data structure that contains the response message.
- The API information file that contains variables that z/OS applications use when calling an API.

Tip: The request and response data structures and the API information file are generated from the build toolkit for each operation in the API. You can use the generated summary report or the console log to identify the names of the generated artifacts for each operation.

- The data structure that contains the parameters to pass to the communication stub for communication with the z/OS Connect EE server. The request and response data structures have been made available in a COBOL copybook or a PL/I include file.

The COBOL copybook BAQRINFO is provided in the hlq.SBAQC0B data set and the PL/I include file BAQRINFP is provided in the hlq.SBAQPLI data set. You must ensure the data set for your application is in the SYSLIB concatenation for the compilation JCL.

The following table shows the communication stub data structures to use in your application:

Table 86. Data structures to pass to the communication stub		
BAQRINFO copybook COBOL data structures	BAQRINFP include file PL/I data structures	Description
BAQ-REQUEST-INFO	BAQ_REQUEST_INFO	For passing security parameters to the API request.
BAQ-RESPONSE-INFO	BAQ_RESPONSE_INFO	For passing the return code and response message back to the calling application.

The BAQ REQUEST INFO data structures contain a compatibility level, BAQ-REQUEST-INFO-COMP-LEVEL for COBOL and BAQ_REQUEST_INFO_COMP_LEVEL for PL/I. The following table shows the supported levels:

Table 87. BAQ REQUEST INFO Compatibility levels	
Compatibility Level	Description
0	The initial value, no security parameters are supported.
1	Support OAuth parameters, see “OAuth parameters and variables” on page 639 .
2	Support JSON Web Token parameters, see “JWT parameters and variables” on page 642 .
3	Support dynamic routing to z/OS Connect EE servers from CICS, see “Overriding the URIMAP in a CICS application” on page 643 .

The higher compatibility levels will automatically support the parameters enabled by the lower compatibility levels.

z/OS Connect EE supplies sample programs that demonstrate how to call APIs through the z/OS Connect EE server in the hlq.SBAQSAMP data set. The samples contain comments to help you write your own applications. For COBOL, the sample program is BAQCAPPO; for PL/I, the sample program is BAQCAPP.

Tip: You can use the communication stub trace as an aid with your application development. For more information, see [“Enabling trace in communications stubs” on page 716](#).

Procedure

The following steps demonstrate how to modify a CICS COBOL application to call an operation of an API using the communication stub.

1. Include the BAQRINFO data structure.

Add the line COPY BAQRINFO into the WORKING-STORAGE-SECTION of your z/OS application program.

```
COPY BAQRINFO
```

2. Include the request data structure, response data structure and the API information file that have been generated for the operation you want to call.

Assuming the request data structure is API00Q01, the response data structure is API00P01, and the API information file is API00I01, insert the following code snippet:

```
01 REQUEST.
  COPY API00Q01.
01 RESPONSE.
  COPY API00P01.
01 API-INFO.
  COPY API00I01.
```

3. Declare variables for the request, response and communication stub.

The following table shows the variables that you need to declare for the request and response:

Variables for COBOL	Variables for PL/I	Description
BAQ-REQUEST-PTR	BAQ_REQUEST_PTR	Pointer to the storage for request messages.
BAQ-REQUEST-LEN	BAQ_REQUEST_LEN	Size of the storage for request messages.
BAQ-RESPONSE-PTR	BAQ_RESPONSE_PTR	Pointer to the storage for the response message.
BAQ-RESPONSE-LEN	BAQ_RESPONSE_LEN	Size of the storage for the response message.
COMM-STUB-PGM-NAME	COMM_STUB_PGM_NAME	Program name of the communication stub. The value of the variable must be set to BAQCSTUB.
COMM-TERM-PGM-NAME	COMM_TERM_PGM_NAME	Used to terminate the reusable connection of BAQCSTUB after IMS unloads the application module. The value of the variable must be set to BAQCTERM.

For example, insert the following code snippet:

```
01 BAQ-REQUEST-PTR          USAGE POINTER.
01 BAQ-REQUEST-LEN          PIC S9(9) COMP-5 SYNC.
01 BAQ-RESPONSE-PTR         USAGE POINTER.
01 BAQ-RESPONSE-LEN         PIC S9(9) COMP-5 SYNC.
77 COMM-STUB-PGM-NAME       PIC X(8) VALUE 'BAQCSTUB'.
77 COMM-TERM-PGM-NAME       PIC X(8) VALUE 'BAQCTERM'.
```

Note: If the request or response message is empty, you must set the pointer to NULL and set the storage size to 0, for example, for an empty COBOL request, set **BAQ-REQUEST-PTR** to NULL and set **BAQ-REQUEST-LEN** to 0.

4. Populate values for the request.

- a. Assuming Xtext is a parameter in the request data structure API00Q01, insert the following code line:

```
MOVE 'How are you' TO Xtext
```

Tip: To populate values for a request that contains fields of array or string, you can refer to ["Sample: Specifying values for arrays and strings" on page 636](#).

- b. Specify values for the variables that have been declared for the request and response in Step 3.

```
SET BAQ-REQUEST-PTR TO ADDRESS OF REQUEST.
MOVE LENGTH OF REQUEST TO BAQ-REQUEST-LEN.
```

```
SET BAQ-RESPONSE-PTR TO ADDRESS OF RESPONSE.  
MOVE LENGTH OF RESPONSE TO BAQ-RESPONSE-LEN.
```

5. Call the communication stub with the required parameters.

In this example, dynamic linkage is used by using the CALL *identifier* statement, where *identifier* is a variable named COMM-STUB-PGM-NAME for the BAQCSTUB load module. The variable has been declared in Step 3 with the value of 'BAQCSTUB'.

```
CALL COMM-STUB-PGM-NAME USING  
      BY REFERENCE API-INFO  
      BY REFERENCE BAQ-REQUEST-INFO  
      BY REFERENCE BAQ-REQUEST-PTR  
      BY REFERENCE BAQ-REQUEST-LEN  
      BY REFERENCE BAQ-RESPONSE-INFO  
      BY REFERENCE BAQ-RESPONSE-PTR  
      BY REFERENCE BAQ-RESPONSE-LEN.
```

6. Retrieve the values from the response.

```
IF BAQ-SUCCESS THEN  
  DISPLAY "The program call is successful. "  
ELSE  
  EVALUATE TRUE  
  WHEN BAQ-ERROR-IN-API  
    DISPLAY "API RETURN ERROR: " BAQ-STATUS-CODE  
    DISPLAY "API RETURN ERROR MESSAGE: "  
    BAQ-STATUS-MESSAGE(1:BAQ-STATUS-MESSAGE-LEN)  
  WHEN BAQ-ERROR-IN-ZCEE  
    DISPLAY "Z/OS CONNECT EE RETURN ERROR: " BAQ-STATUS-CODE  
    DISPLAY "SERVER RETURN ERROR MESSAGE: "  
    BAQ-STATUS-MESSAGE(1:BAQ-STATUS-MESSAGE-LEN)  
  WHEN BAQ-ERROR-IN-STUB  
    DISPLAY "STUB RETURN ERROR: " BAQ-STATUS-CODE  
    DISPLAY "STUB RETURN ERROR MESSAGE: "  
    BAQ-STATUS-MESSAGE(1:BAQ-STATUS-MESSAGE-LEN)  
  END-EVALUATE  
END-IF.
```

As shown in the sample code, if the API call is successful, the region log displays that the program call is successful; otherwise, the region log displays the detailed error message. Errors might come from the API, the z/OS application server, or the z/OS subsystems. A return code and a status code are included in the response message to provide the error detail. In a COBOL program, the special variable *RETURN-CODE* is set to the value of BAQ-RETURN-CODE. For more information about error handling, see ["Error handling for API requester calls" on page 648](#).

Note: For batch applications, the message is displayed in the job log.

7. If you have any long running API requester applications that use the IMS or z/OS communication stub and cached connections, you must close and clear the cached connection by calling the BAQTERM function. If BAQTERM is not called when the application is unloaded, the connection used by BAQCSTUB is not closed and the memory it owns is not re-usable until the IMS or z/OS communication process terminates.

The following example shows how to call the BAQTERM function, which should be performed at the logical end of your application.

```
CALL COMM-TERM-PGM-NAME USING  
      BY REFERENCE BAQ-RESPONSE-INFO.  
  
IF BAQ-SUCCESS THEN  
  DISPLAY "The BAQCSTUB connection has been terminated."  
ELSE  
  EVALUATE TRUE  
  WHEN BAQ-ERROR-IN-API  
    DISPLAY "API RETURN ERROR: " BAQ-STATUS-CODE  
    DISPLAY "API RETURN ERROR MESSAGE: "  
    BAQ-STATUS-MESSAGE(1:BAQ-STATUS-MESSAGE-LEN)  
  WHEN BAQ-ERROR-IN-ZCEE  
    DISPLAY "Z/OS CONNECT EE RETURN ERROR: " BAQ-STATUS-CODE  
    DISPLAY "SERVER RETURN ERROR MESSAGE: "  
    BAQ-STATUS-MESSAGE(1:BAQ-STATUS-MESSAGE-LEN)
```

```

WHEN BAQ-ERROR-IN-STUB
    DISPLAY "STUB RETURN ERROR: " BAQ-STATUS-CODE
    DISPLAY "STUB RETURN ERROR MESSAGE: "
        BAQ-STATUS-MESSAGE(1:BAQ-STATUS-MESSAGE-LEN)
    END-EVALUATE
END-IF.

```

As in step “6” on page 635, if the BAQCSTUB connection is terminated successfully, the region log indicates that the connection termination is successful; otherwise, the region log displays the detailed error message.

Results

The CICS application program is modified to call the operation of the API using the communication stub.

What to do next

- Refer to “[Testing the z/OS application for API calls](#)” on page 653 to compile and run your z/OS application program to call the API.
- If the RESTful API that you want to call is secured with OAuth 2.0 or an API key, you must make some other modifications to your z/OS application to ensure the required parameters are provided in the API request. For more information, see “[Securing your z/OS application calls to APIs](#)” on page 637.

Sample: Specifying values for arrays and strings

A sample is provided to show how a generated copybook looks like when the schema in a Swagger file uses the data type of array and string. This sample also shows how to specify values for the array and string fields in the copybook.

A Swagger file sample

Assume a Swagger file in JSON format:

```
{
  "swagger": "2.0",
  "info": {
    "version": "0.0.1",
    "title": "Sample Swagger file to explain how to populate data to array and string in program."
  },
  "paths": {
    "/": {
      "get": {
        "produces": [ "application/json" ],
        "description": "operation with array and optional string parameters",
        "responses": {
          "200": {
            "description": "successful operation",
            "schema": { "$ref": "#/definitions/sampleArray" }
          }
        }
      }
    }
  },
  "definitions": {
    "sampleArray": {
      "required": ["myarray"],
      "properties": {
        "myarray": {
          "type": "array",
          "maxItems": 10,
          "items": {
            "type": "object",
            "properties": {
              "optionalString": {
                "type": "string"
              }
            }
          }
        }
      }
    }
  }
}
```

```
    }  
  }  
}
```

In the definition of `sampleArray`, an array named `myarray` is defined. In this array, an object with one optional string parameter named `optionalString` is defined.

Fields in the generated copybook

Based on the Swagger file above, the following lines are generated in the copybook:

```
06 RespBody.  
          09 myarray2-num           PIC S9(9) COMP-5 SYNC.  
          09 myarray OCCURS 10.  
            12 optionalString-num   PIC S9(9) COMP-5 SYNC.  
            12 optionalString.  
              15 optionalString2-length  PIC S9999 COMP-5 SYNC.  
              15 optionalString2      PIC X(255).  
            12 filler                PIC X(3).
```

The **myarray** field represents an array. The size of array is specified in the **myarray2-num** field. The maximum size is 10, which is indicated by the clause of `OCCURS 10`.

Tip: You can find the minimum size of the array in the comments of the copybook.

The **optionalString** field represents an element of the array. If an element is empty, which means no value needs to be set to the **optionalString** field, you must set **optionalString-num** as 0; otherwise, set **optionalString-num** as 1.

The **optionalString2** field represents a variable-length array of characters of binary data. This field is used to specify the value of an array element. The **optionalString2-length** field is used to specify the length of the value of **optionalString2**.

Example: specifying values for the array and string fields

Assume the following array data to be populated into the generated copybook:

```
[[ {"MY"}, {"HELLO"}, {} ]]
```

You can specify values for the array and string fields as follows:

```
MOVE 3 TO myarray2-num.  
MOVE 1 TO optionalString-num IN myarray(1).  
MOVE '2' TO optionalString2-length IN myarray(1).  
MOVE 'MY' TO optionalString2 IN myarray(1).  
  
MOVE 1 TO optionalString-num IN myarray(2).  
MOVE '5' TO optionalString2-length IN myarray(2).  
MOVE 'HELLO' TO optionalString2 IN myarray(2).  
  
MOVE 0 TO optionalString-num IN myarray(3).
```

Securing your z/OS application calls to APIs

To call a RESTful API that is secured with OAuth 2.0, an API key or a JSON Web Token (JWT), you must modify your CICS, IMS, or other z/OS application to include and specify the required security parameters in the API request.

Pre-requisites: Follow the instructions in [“Developing a z/OS application to call APIs” on page 632](#) to learn about how to modify a z/OS application to call a RESTful API.

API keys

To call a RESTful API that is secured with an API key, ensure that the information of an API key is provided. For example, if an API key X-IBM-Client-Secret is defined in a Swagger file, ensure that the request data structure that is generated by the build toolkit has the following parameters:

- **X-IBM-Client-Secret**: The name of the API key.
- **X-IBM-Client-Secret-Length**: The length of the value that is set to **X-IBM-Client-Secret**.

You must populate values for **X-IBM-Client-Secret** and **X-IBM-Client-Secret-Length**.

For more information about how API key authentication works with the z/OS Connect EE, see [“Call an API secured with an API key” on page 419](#).

OAuth 2.0

To call a RESTful API that is secured with OAuth 2.0, ensure that the information that is used for OAuth authorization is included in the request.

z/OS Connect EE supports two grant types: Resource Owner Password Credentials grant type and Client Credentials grant type. If the API is protected by the Resource Owner Password Credentials grant type, you must include the information of resource owner's username and password, client ID, client secret, and scope according to your setting preferences. If the API is protected by the Client Credentials grant type, you must include the information of client ID, client secret, and scope.

For COBOL, the OAuth information is defined by related parameters in BAQ-REQUEST-INFO that is in the BAQRINFO data structure and an OAuth scope variable BAQ-OAUTH-SCOPE. For PL/I, the OAuth information is defined by related parameters in BAQ_REQUEST_INFO that is in the BAQRINFP data structure and an OAuth scope variable BAQ_OAUTH_SCOPE. And you must declare the OAuth scope variable and specify values for the parameters and variables as you need.

If a request to an API endpoint uses an OAuth 2.0 access token and fails with a 401 HTTP response code and the response in the WWW_AUTHENTICATE header contains "invalid_token", then the request is retried once.

For more information about the parameters and the variable for OAuth, see [“OAuth parameters and variables” on page 639](#).

For more information about how OAuth 2.0 works with the z/OS Connect EE, see [“Call an API secured with OAuth 2.0” on page 421](#).

JSON Web Token (JWT)

To retrieve a JWT from the authentication server and call a RESTful API that is secured with the JWT, ensure that user credentials are provided. The user credentials include the user name and password that are used to authenticate with an authentication server to obtain a JWT.

Note: The user credentials might represent an end user or an application.

You can provide user credentials in your z/OS application or in the `server.xml` file. If both the z/OS application and the `server.xml` file are set up with user credentials for a JWT, the user credentials that are specified in the `server.xml` file are used.

To include user credentials in the z/OS application, specify the parameters that are defined for the JWT in the API request. For more information, see the [“JWT parameters and variables” on page 642](#).

If a request to an API endpoint uses a JWT and fails with a 401 HTTP response code, the request is retried once.

For more information about how JWT is used with an API requester, see [“Calling an API secured with a JSON Web Token \(JWT\)” on page 426](#).

OAuth parameters and variables

When you develop a CICS, IMS or z/OS application to call an API that is protected by OAuth 2.0, you can include the information for the OAuth parameters in the request.

z/OS Connect EE supplies sample programs in the hlq.SBAQSAMP data set. These samples demonstrate how to call an API that are protected by OAuth 2.0 using z/OS Connect EE. For COBOL, the sample program is BAQAUTHO; for PL/I, the sample program is BAQAUTHP.

Information that your application must include for OAuth 2.0

- Resource owner's username and password: to be used for the authorization server to validate the resource owner's account on the request endpoint. Only required for the Resource Owner Password Credentials grant type.
- Client ID or client secret: to be used for the authorization server to authenticate the client. Applied for both the Resource Owner Password Credentials and the Client Credentials grant types.

Whether to specify client secret depends on whether client is set as public in the authorization server. If client is set as public in the authorization server, only the client ID needs to be set; if client is set as confidential in the authorization server, both the client ID and client secret need to be set.

For the Client Credentials grant type, client is always confidential in the authorization server, so both client ID and client secret are required if the Client Credentials grant type is used.

Important:

If you do not include client ID or client secret in your application, you must specify them in the basicAuthRef attribute of the zosconnect_authorizationServer element in the server.xml file. If client ID or client secret is set in both the server.xml file and the z/OS application program, then the value set in the server.xml file is used.

- Scope: to be used for indicating the permissions granted to your application to access the API owner's data. Optional for both the Resource Owner Password Credentials and the Client Credentials grant types.

Scope is defined in an authorization server by the API owner. You can contact the admin of the authorization server for the scope-related information if you want to know what your application can do and cannot do with the resource owner's account.

Parameters for OAuth in the request

z/OS Connect EE provides a data structure to specify the OAuth parameters in the request. Two versions of the data structure are provided, a COBOL version called BAQRINFO in hlq.SBAQCOP and a PL/I version called BAQRINFP in hlq.SBAQPLI.

Ensure the communication stub request information structure is set with a compatibility level of 1 or higher. The compatibility level value is defined for COBOL by the BAQ-REQUEST-INFO-COMP-LEVEL variable or for PL/I by the BAQ_REQUEST_INFO_COMP_LEVEL variable.

The following table shows the parameters that are defined for OAuth 2.0. For COBOL, these parameters are defined in BAQ-REQUEST-INFO; for PL/I, these parameters are defined in BAQ_REQUEST_INFO. You can specify values for the parameters as required.

Table 89. Variables defined for OAuth

Variables for COBOL	Variables for PL/I	Description
BAQ-OAUTH-USERNAME	BAQ_OAUTH_USER_NAME	Username value used for the authorization server to validate the resource owner's credentials. Only required for the Resource Owner Password Credential grant type.
BAQ-OAUTH-USERNAME-LEN	BAQ_OAUTH_USER_NAME_LEN	Length of username. The maximum value is 256.

Table 89. Variables defined for OAuth (continued)

Variables for COBOL	Variables for PL/I	Description
BAQ-OAUTH-PASSWORD	BAQ_OAUTH_PASSWORD	Password used for the authorization server to validate the resource owner's credentials. Only required for the Resource Owner Password Credential grant type.
BAQ-OAUTH-PASSWORD-LEN	BAQ_OAUTH_PASSWORD_LEN	Length of password. The maximum value is 256.
BAQ-OAUTH-CLIENTID	BAQ_OAUTH_CLIENTID	Client ID value used for the authorization server to authenticate the client. Applied for both the Resource Owner Password Credential and Client Credentials grant types.
BAQ-OAUTH-CLIENTID-LEN	BAQ_OAUTH_CLIENTID_LEN	Length of client ID. The maximum value is 256.
BAQ-OAUTH-CLIENT-SECRET	BAQ_OAUTH_CLIENT_SECRET	Client secret value used for the authorization server to authenticate the client. Applied for both the Resource Owner Password Credential and Client Credentials grant types.
BAQ-OAUTH-CLIENT-SECRET-LEN	BAQ_OAUTH_CLIENT_SECRET_LEN	Length of client secret. The maximum value is 256.
BAQ-OAUTH-SCOPE-PTR	BAQ_OAUTH_SCOPE_PTR	Pointer to the storage for the <u>scope variable</u> that indicates what your application can perform on the resource owner's data protected by OAuth.
BAQ-OAUTH-SCOPE-LEN	BAQ_OAUTH_SCOPE_LEN	Size of the storage for the <u>scope variable</u> .

Variables to declare

The following table shows the OAuth scope variable that you might need to declare for the request:

Table 90. Variables to declare

Variables for COBOL	Variables for PL/I	Description
BAQ-OAUTH-SCOPE	BAQ_OAUTH_SCOPE	Scope that specifies what your application wants to perform on the resource owner's data protected by OAuth. The value of this variable can be set to a scope or a blank separated list of scopes.

For more information about how these parameters and variables are used in your z/OS application, see the [Example](#) or [“Securing your z/OS application calls to APIs” on page 637](#).

Example: Developing a COBOL application to call an API protected by OAuth 2.0

The following example demonstrates how to develop a COBOL application to call an API that is protected by OAuth 2.0.

In this example, the names of the generated artifacts are as follows:

- The request copybook: API00Q01
- The response copybook: API00P01
- The API information file: API00I01

The information for OAuth that needs to be included is as follows:

- Resource owner's user name: oauthuser1
- Resource owner's password: oauthpassword1
- Client ID: clientid1
- Client secret: clientsecret1
- Scope: /getAccount

Include copybooks

```
01 REQUEST.
  COPY API00Q01.
01 RESPONSE.
  COPY API00P01.
01 API-INFO.
  COPY API00I01.
```

Declare variables for the request and response

```
01 BAQ-REQUEST-PTR          USAGE POINTER.
01 BAQ-REQUEST-LEN          PIC S9(9) COMP-5 SYNC.

01 BAQ-OAUTH-SCOPE          PIC X(255).
01 BAQ-RESPONSE-PTR         USAGE POINTER.
01 BAQ-RESPONSE-LEN         PIC S9(9) COMP-5 SYNC.
77 COMM-STUB-PGM-NAME       PIC X(8) VALUE 'BAQCSTUB'.
```

Populate values for the request

```
MOVE 'How are you' TO Xtext.

MOVE "oauthuser1" TO BAQ-OAUTH-USERNAME.
MOVE 10 TO BAQ-OAUTH-USERNAME-LEN.

MOVE "oauthpassword1" TO BAQ-OAUTH-PASSWORD.
MOVE 14 TO BAQ-OAUTH-PASSWORD-LEN.

MOVE "clientid1" TO BAQ-OAUTH-CLIENTID.
MOVE 9 TO BAQ-OAUTH-CLIENTID-LEN.

MOVE "clientsecret1" TO BAQ-OAUTH-CLIENT-SECRET.
MOVE 13 TO BAQ-OAUTH-CLIENT-SECRET-LEN.

MOVE "/getAccount" TO BAQ-OAUTH-SCOPE.
SET BAQ-OAUTH-SCOPE-PTR TO ADDRESS OF BAQ-OAUTH-SCOPE.
MOVE 11 TO BAQ-OAUTH-SCOPE-LEN.
```

Prepare the data for call

```
SET BAQ-REQUEST-PTR TO ADDRESS OF REQUEST.
MOVE LENGTH OF REQUEST TO BAQ-REQUEST-LEN.
SET BAQ-RESPONSE-PTR TO ADDRESS OF RESPONSE.
MOVE LENGTH OF RESPONSE TO BAQ-RESPONSE-LEN.
```

Call the communication stub

```
CALL COMM-STUB-PGM-NAME USING
  BY REFERENCE API-INFO
  BY REFERENCE BAQ-REQUEST-INFO
  BY REFERENCE BAQ-REQUEST-PTR
  BY REFERENCE BAQ-REQUEST-LEN
  BY REFERENCE BAQ-RESPONSE-INFO
  BY REFERENCE BAQ-RESPONSE-PTR
  BY REFERENCE BAQ-RESPONSE-LEN.
```

JWT parameters and variables

When you develop a CICS, IMS or z/OS application to call an API that is secured with a JSON Web Token (JWT), you can include the information for the JWT parameters in the request.

z/OS Connect EE supplies a sample COBOL program called BAQJWT in the hlq.SBAQSAMP data set. The sample demonstrates how to call an API that are protected by JWT using z/OS Connect EE.

Parameters for JWT in the request

z/OS Connect EE provides a data structure to specify the JWT parameters in the request. Two versions of the data structure are provided, a COBOL version called BAQRINFO in hlq.SBAQC0B and a PL/I version called BAQRINFP in hlq.SBAQPLI.

Ensure the communication stub response information structure is set with a compatibility level of 2 or higher. The compatibility level value is defined for COBOL by the BAQ-REQUEST-INFO-COMP-LEVEL variable or for PL/I by the BAQ_REQUEST_INFO_COMP_LEVEL variable.

The following table shows the parameters that are defined for JWT. For COBOL, these parameters are defined in BAQ-REQUEST-INFO; for PL/I, these parameters are defined in BAQ_REQUEST_INFO. You can specify values for the parameters as required.

Table 91. Parameters defined for JWT		
Variables for COBOL	Variables for PL/I	Description
BAQ-TOKEN-USERNAME	BAQ_TOKEN_USERNAME	The user name that is used for the authentication server to authenticate the user.
BAQ-TOKEN-USERNAME-LEN	BAQ_TOKEN_USERNAME_LEN	The length of the user name. The maximum value is 256.
BAQ-TOKEN-PASSWORD	BAQ_TOKEN_PASSWORD	The password that is used for the authentication server to authenticate the user.
BAQ-TOKEN-PASSWORD-LEN	BAQ_TOKEN_PASSWORD_LEN	The length of the password. The maximum value is 256.

Example: Developing a COBOL application to call an API secured with a JWT

The following example demonstrates how to develop a COBOL application to call an API that is secured with a JWT.

In this example, the names of the generated artifacts are as follows:

- The request copybook: API00Q01
- The response copybook: API00P01
- The API information file: API00I01

The following information for JWT is included:

- User name: jwtuser
- Password: jwtpassword

Include the BAQRINFO data structure

```
COPY BAQRINFO
```

Include copybooks

```
01 REQUEST.  
      COPY API00Q01.  
01 RESPONSE.
```

```
COPY API00P01.  
01 API-INFO.  
COPY API00I01.
```

Declare variables for the request and response

```
01 BAQ-REQUEST-PTR           USAGE POINTER.  
01 BAQ-REQUEST-LEN          PIC S9(9) COMP-5 SYNC.  
01 BAQ-RESPONSE-PTR         USAGE POINTER.  
01 BAQ-RESPONSE-LEN         PIC S9(9) COMP-5 SYNC.  
77 COMM-STUB-PGM-NAME       PIC X(8) VALUE 'BAQCSTUB'.
```

Populate values for the request

```
MOVE 'How are you' TO Xtext.  
  
MOVE "jwtuser" TO BAQ-TOKEN-USERNAME.  
MOVE 7 TO BAQ-TOKEN-USERNAME-LEN.  
MOVE "jwtpassword" TO BAQ-TOKEN-PASSWORD.  
MOVE 11 TO BAQ-TOKEN-PASSWORD-LEN.
```

Prepare the data for call

```
SET BAQ-REQUEST-PTR TO ADDRESS OF REQUEST.  
MOVE LENGTH OF REQUEST TO BAQ-REQUEST-LEN.  
SET BAQ-RESPONSE-PTR TO ADDRESS OF RESPONSE.  
MOVE LENGTH OF RESPONSE TO BAQ-RESPONSE-LEN.
```

Call the communication stub

```
CALL COMM-STUB-PGM-NAME USING  
      BY REFERENCE API-INFO  
      BY REFERENCE BAQ-REQUEST-INFO  
      BY REFERENCE BAQ-REQUEST-PTR  
      BY REFERENCE BAQ-REQUEST-LEN  
      BY REFERENCE BAQ-RESPONSE-INFO  
      BY REFERENCE BAQ-RESPONSE-PTR  
      BY REFERENCE BAQ-RESPONSE-LEN.
```

Overriding the URIMAP in a CICS application

When a CICS application is developed to call a RESTful service, the z/OS Connect EE server that is used to process the request can be chosen at run time within the CICS application. This option allows for splitting of workloads between servers, for example, based on business area. This dynamic routing capability is available for CICS applications only.

Overview

Figure 116 on page 644 illustrates how CICS applications can choose which z/OS Connect EE server to route the request.

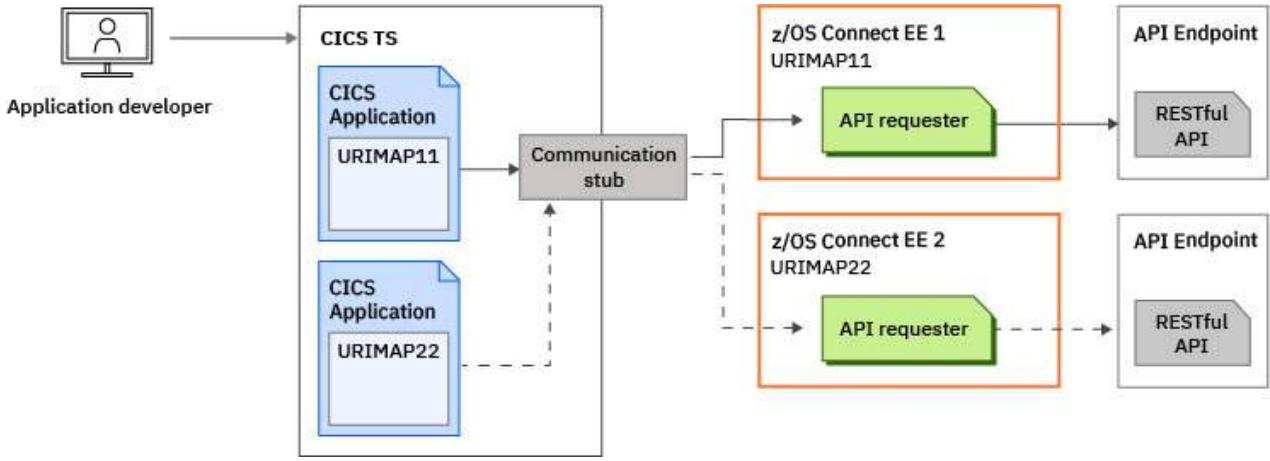


Figure 116. Routing requests from CICS applications

The two z/OS Connect EE servers shown in Figure 116 on page 644 can have different configurations and support different API requesters. You can still configure multiple servers with the same configuration behind shared ports for work load management.

Parameters for dynamic routing in the request

z/OS Connect EE provides a data structure to specify the dynamic routing parameters in the request. Two versions of the data structure are provided, a COBOL version called `BAQRINFO` in `<hlq>.SBAQCOB` and a PL/I version called `BAQRINFP` in `<hlq>.SBAQPLI`.

Ensure that the communication stub request information structure is set with a compatibility level of 3 or higher. The compatibility level value is defined for COBOL by the `BAQ-REQUEST-INFO-COMP-LEVEL` variable and for PL/I by the `BAQ_REQUEST_INFO_COMP_LEVEL` variable.

The following table shows the parameters that are defined for dynamic routing. For COBOL, these parameters are defined in `BAQ-REQUEST-INFO`. For PL/I, these parameters are defined in `BAQ_REQUEST_INFO`. You can specify values for the parameters as required.

Table 92. Parameter defined for dynamic routing		
Variables for COBOL	Variables for PL/I	Description
<code>BAQ-ZCON-SERVER-URI</code>	<code>BAQ_ZCON_SERVER_URI</code>	A URIMAP resource name used to route to a particular z/OS Connect EE server. The name must be a valid URIMAP resource name of 1 - 8 characters in length. The field size is 256 characters, but this is to accommodate future enhancements.

The `BAQ-ZCON-SERVER-URI` field is optional. If the field is left initialized with spaces, then the default URIMAP of `BAQURIMP` is used to route the request. If the field is populated, the URIMAP resource name is validated and used to route the request. If there is a problem with the URIMAP resource name, or the URIMAP is unknown, unavailable, or links to an unknown host then an appropriate error message is returned and the request is not processed.

All URIMAP resources to be used must be defined in the CICS CSD.

Example: Developing a COBOL application to dynamically route to a z/OS Connect EE server.

The following example demonstrates how to develop a COBOL application to dynamically route to a particular server.

In this example, the names of the generated artifacts are as follows:

- The request copybook: `API00Q01`

- The response copybook: API00P01
- The API information file: API00I01

Include the BAQRINFO data structure.

```
COPY BAQRINFO
```

Include copybooks.

```
01 REQUEST.
  COPY API00Q01.
01 RESPONSE.
  COPY API00P01.
01 API-INFO.
  COPY API00I01.
```

Declare variables for the request and response.

```
01 BAQ-REQUEST-PTR          USAGE POINTER.
01 BAQ-REQUEST-LEN          PIC S9(9) COMP-5 SYNC.
01 BAQ-RESPONSE-PTR         USAGE POINTER.
01 BAQ-RESPONSE-LEN         PIC S9(9) COMP-5 SYNC.
77 COMM-STUB-PGM-NAME       PIC X(8) VALUE 'BAQCSTUB'.
```

Populate values for the dynamically routed request.

```
MOVE 'Req Data' TO Xtext.
MOVE "URIMAP01" TO BAQ-ZCON-SERVER-URI.
```

Prepare the data for call.

```
SET BAQ-REQUEST-PTR TO ADDRESS OF REQUEST.
MOVE LENGTH OF REQUEST TO BAQ-REQUEST-LEN.
SET BAQ-RESPONSE-PTR TO ADDRESS OF RESPONSE.
MOVE LENGTH OF RESPONSE TO BAQ-RESPONSE-LEN.
```

Call the communication stub.

```
CALL COMM-STUB-PGM-NAME USING
  BY REFERENCE API-INFO
  BY REFERENCE BAQ-REQUEST-INFO
  BY REFERENCE BAQ-REQUEST-PTR
  BY REFERENCE BAQ-REQUEST-LEN
  BY REFERENCE BAQ-RESPONSE-INFO
  BY REFERENCE BAQ-RESPONSE-PTR
  BY REFERENCE BAQ-RESPONSE-LEN.
```

Linking by dynamic calls

When developing your z/OS application to call RESTful APIs in COBOL or PL/I, ensure dynamic linkage is used, where possible, between the z/OS application code and the z/OS Connect EE-supplied communication stub module, BAQCSTUB.

Why dynamic linkage is strongly recommended?

Use of dynamic linkage is always recommended over static linkage because it enables your program to use, without relinking, the most current version of the BAQCSTUB modules delivered through regular z/OS Connect EE product maintenance. This is not possible with static linkage. When static linkage is used, your z/OS application program needs to be relinked to benefit from the latest maintenance.

When dynamic linkage can be used?

Dynamic linkage can be applied to both versions of the communication stub, <hlq>.SBAQLIB1(BAQCSTUB) that is used for CICS applications and <hlq>.SBAQLIB(BAQCSTUB) that is used for IMS and other z/OS applications.

Important: Dynamic linkage is not supported for PL/I applications running under CICS; therefore, you must use static linkage for CICS PL/I applications. For more information, see [Making static calls from CICS PL/I applications](#).

How to make dynamic calls?

Follow [Table 1](#) to find out how to make dynamic calls based on your environment and the application programming language used.

Table 93. Methods to make dynamic calls		
Environment	Language	Programming instructions
CICS	COBOL	Use the CALL <i>identifier</i> statement 1 . For detailed instructions, see Using CALL identifier to make dynamic calls in COBOL .
IMS, other z/OS applications	COBOL	Two ways are available: <ul style="list-style-type: none">• Use the CALL <i>identifier</i> 1 statement. For detailed instructions, see Using CALL identifier to make dynamic calls in COBOL.• Use the CALL <i>literal</i> 2 statement with the DYNAM compile option specified to YES. For detailed instructions, see Using CALL literal and DYNAM to make dynamic calls in COBOL (non-CICS).
IMS, other z/OS applications	PL/I	Use EXTERN + FETCH + CALL. For detailed instructions, see Making dynamic calls from PL/I applications (non-CICS) .

Note:

[1](#)CALL *identifier*, where *identifier* is a data item that contains the name of a nonnested subprogram at run time, always results in the target subprogram being loaded when it is called.

[2](#)CALL *literal*, where *literal* is the explicit name of a nonnested target subprogram, can be resolved either statically or dynamically.

Making dynamic calls from COBOL applications

To ensure that your COBOL application uses dynamic linkage with BAQCSTUB, you can use one of the following two ways:

Using CALL *identifier* to make dynamic calls in COBOL

Using the CALL *identifier* statement is an explicit method for the COBOL programmer to use dynamic linkage.

Note: For CICS COBOL applications, this is the only supported method.

An example of using this explicit dynamic link convention for COBOL is shown below:

```
* Stub program name
77 COMM-STUB-PGM-NAME      PIC X(8) VALUE 'BAQCSTUB'.
...
* Invoke the z/OS Connect EE communication stub by using dynamic linkage
MOVE 'BAQCSTUB' to COMM-STUB-PGM-NAME.
CALL COMM-STUB-PGM-NAME USING MY-DATA-AREA.
```

This is the method that is also used in the z/OS Connect EE-supplied samples BAQCATPO and BAQAUTHO, which can be found in the `<hlq>.SBAQSAMP` product dataset.

Using CALL *literal* and DYNAM to make dynamic calls in COBOL (non-CICS)

If your COBOL application specifies the CALL target as a literal string, 'BAQSTUB', static linkage is used. To force your z/OS application to use dynamic linkage, you must set the DYNAM compile option to YES when you compile your application.

Note: This method is not supported for CICS COBOL applications when you use the z/OS Connect EE API requester feature, because BAQCSTUB uses the CICS translator but the CICS translator cannot be used with the DYNAM option.

An example of using a static link convention for COBOL is shown below:

```
* Invoke the z/OS Connect EE communication stub by using static linkage
CALL 'BAQSTUB' USING MY-DATA-AREA.
```

For more information about how to use DYNAM, see the [option reference](#) in the Enterprise COBOL for z/OS documentation.

Making dynamic calls from PL/I applications (non-CICS)

To ensure that your PL/I application uses dynamic linkage with BAQCSTUB, you must declare the BAQCSTUB module as EXTERNAL together with the ENTRY OPTIONS attribute, NODESCRIPTOR, then use a FETCH statement to dynamically load BAQCSTUB, and then use a CALL statement to invoke BAQCSTUB. This method is strongly recommended as best practice for PL/I applications when you use the API requester feature.

An example of using this dynamic link convention for PL/I is shown below:

```
/* option NODESCRIPTOR must be used when calling BAQCSTUB */
DCL BAQCSTUB EXTERNAL ENTRY OPTIONS(NODESCRIPTOR);

/* FETCH the z/OS Connect EE communication stub for dynamic linkage */
FETCH BAQCSTUB;

/* Invoke the z/OS Connect EE communication stub by using dynamic linkage */
CALL BAQCSTUB(API_INFO_API1,
               BAQ_REQUEST_INFO,
               BAQ_REQUEST_PTR,
               BAQ_REQUEST_LEN,
               BAQ_RESPONSE_INFO,
               BAQ_RESPONSE_PTR,
               BAQ_RESPONSE_LEN);
```

This is the method that is also used in the z/OS Connect EE-supplied samples BAQCATPP and BAQAUTHP, which can be found in the `<hlq>.SBAQSAMP` product dataset.

Making static calls from CICS PL/I applications

Dynamic linkage is not supported on calls by CICS PL/I applications to BAQCSTUB, because BAQCSTUB itself is a C application that uses the CICS translator. This interlanguage communication restriction is documented in the z/OS Language Environment documentation, "[z/OS XL C/C++ and PL/I](#)".

So you must use static linkage for CICS PL/I applications when using the API requester feature, and relink the applications to exploit new capabilities in the z/OS Connect EE communication stub, or benefit from APAR fixes.

To use static linkage, you must declare the ENTRY point for BAQCSTUB.

Note:

- Do not declare BAQCSTUB as EXTERNAL.
- Do not use the FETCH statement.
- For non-CICS PL/I applications, static linkage is strongly discouraged.

An example of using a static link convention for PL/I is shown below:

```
/* option NODESCRIPTOR must be used when calling BAQCSTUB */
DCL BAQCSTUB ENTRY OPTIONS(NODESCRIPTOR);

/* Invoke zOS Connect EE communication stub by using static linkage */
CALL BAQCSTUB(API_INFO_API1,
               BAQ_REQUEST_INFO,
               BAQ_REQUEST_PTR,
               BAQ_REQUEST_LEN,
               BAQ_RESPONSE_INFO,
               BAQ_RESPONSE_PTR,
               BAQ_RESPONSE_LEN);
```

Error handling for API requester calls

Use this information to help you design error handling into your applications

You can enable application user trace in the communication stubs for application development and problem determination. For more information, see “[Enabling trace in communications stubs](#)” on page [716](#).

The communication stub response information structure captures the result of an API call. The response information structure is populated with the return code and the message when the API call returns.

For COBOL

The response information structure is named BAQ-RESPONSE-INFO. The request information structure is named BAQ-REQUEST-INFO.

For PL/I

The response information structure is named BAQ_RESPONSE_INFO. The request information structure is named BAQ_REQUEST_INFO.

Note: This topic takes a COBOL program as an example. The names from the COBOL data structures are used in the documentation. The PL/I names are generally similar to the COBOL names except that PL/I uses '_' where COBOL uses '-'.

Errors can occur for different reasons:

- API errors, as described in the Swagger document.
- z/OS Connect EE server runtime errors, such as data transformation errors or timeout issues.
- Communication stub program errors, such as connection errors or timeout issues.

The communication stub response information structure contains the response information from the API call. If an error occurs, the return code value indicates the source of the error, and the status code and status message provide further information.

The response information has the following language structure, as indicated in the BAQRINFO copybook.

```
01 BAQ-RESPONSE-INFO.
  03 BAQ-RESPONSE-INFO-COMP-LEVEL PIC S9(9) COMP-5 SYNC VALUE 0.
  03 BAQ-STUB-NAME          PIC X(8).
  03 BAQ-RETURN-CODE        PIC S9(9) COMP-5 SYNC.
    88 BAQ-SUCCESS           VALUE 0.
    88 BAQ-ERROR-IN-API     VALUE 1.
    88 BAQ-ERROR-IN-ZCEE    VALUE 2.
    88 BAQ-ERROR-IN-STUB   VALUE 3.
    88 BAQ-ERROR-NO-RESPONSE VALUE 4.
  03 BAQ-STATUS-CODE        PIC S9(9) COMP-5 SYNC.
  03 BAQ-STATUS-MESSAGE    PIC X(1024).
  03 BAQ-STATUS-MESSAGE-LEN PIC S9(9) COMP-5 SYNC.
```

Note: For PL/I programs, the response information is indicated in the BAQRINFP copybook.

The following table shows the possible return codes, their meanings, and how further information can be obtained.

Table 94. Return code for API requester calls

Return code	Description	Further information
0 : BAQ-SUCCESS	The call to the API is successful.	The BAQ-STATUS-CODE field contains the HTTP status code. The BAQ-STATUS-MESSAGE field contains the message that is returned from the API.
1 : BAQ-ERROR-IN-API	Error in the API endpoint.	The BAQ-STATUS-CODE field contains the HTTP status code. The BAQ-STATUS-MESSAGE field contains the message that is returned from the API in JSON format. If BAQ-STATUS-MESSAGE-LEN is 0 then no message was returned from the API.
2 : BAQ-ERROR-IN-ZCEE	Error on the z/OS Connect EE server.	The BAQ-STATUS-CODE field contains the HTTP status code. The BAQ-STATUS-MESSAGE field contains the message that is returned from the server in JSON format.
3 : BAQ-ERROR-IN-STUB	Error in the communication stub on the z/OS subsystem.	The BAQ-STATUS-CODE field contains the error code set by the communication stub. The BAQ-STATUS-MESSAGE field contains the communication stub error message. The communication stub for CICS logs the error message in TDQUEUE BAQQ.
4 : BAQ-ERROR-NO-RESPONSE	Returned as the value 4 in the COBOL special variable <i>RETURN-CODE</i> . This value is returned if the BAQ-RESPONSE-INFO is passed as null to the BAQCSTUB program. In this case this is the only mechanism to note an error has occurred.	Ensure the program is passing the address of BAQ-RESPONSE-INFO correctly.

In a COBOL or PL/I program, use the BAQ-RETURN-CODE value to determine if the call was successful or where the error occurred.

For a COBOL program, the special variable *RETURN-CODE* is set to the value of BAQ-RETURN-CODE upon return from the call to the communication stub.

The following sample code snippet calls the communication stub (COMM-STUB-PGM-NAME) with the required parameters and displays the values for BAQ-STATUS-CODE and BAQ-STATUS-MESSAGE when BAQ-RETURN-CODE is not BAQ-SUCCESS.

```

CALL COMM-STUB-PGM-NAME USING
      BY REFERENCE    API-INFO
      BY REFERENCE    BAQ-REQUEST-INFO
      BY REFERENCE    BAQ-REQUEST-PTR
      BY REFERENCE    BAQ-REQUEST-LEN
      BY REFERENCE    BAQ-RESPONSE-INFO
      BY REFERENCE    BAQ-RESPONSE-PTR
      BY REFERENCE    BAQ-RESPONSE-LEN.

IF BAQ-SUCCESS THEN
  DISPLAY "The program call is successful. "
ELSE
  EVALUATE TRUE
  WHEN BAQ-ERROR-IN-API
    DISPLAY "API RETURN ERROR: " BAQ-STATUS-CODE
    DISPLAY "API RETURN ERROR MESSAGE: "
    BAQ-STATUS-MESSAGE(1:BAQ-STATUS-MESSAGE-LEN)

```

```

WHEN BAQ-ERROR-IN-ZCEE
  DISPLAY "Z/OS CONNECT EE RETURN ERROR: " BAQ-STATUS-CODE
  DISPLAY "SERVER RETURN ERROR MESSAGE: "
    BAQ-STATUS-MESSAGE(1:BAQ-STATUS-MESSAGE-LEN)
WHEN BAQ-ERROR-IN-STUB
  DISPLAY "STUB RETURN ERROR: " BAQ-STATUS-CODE
  DISPLAY "STUB RETURN ERROR MESSAGE: "
    BAQ-STATUS-MESSAGE(1:BAQ-STATUS-MESSAGE-LEN)
END-EVALUATE
END-IF.

```

The hlq.SBAQSAMP data set contains sample programs that demonstrate how to call APIs through the z/OS Connect EE server. The samples contain comments that help you write your own applications. The following sample programs are provided:

- COBOL: BAQCAPPO
- PL/I: BAQCAPP

Return code: BAQ-ERROR-IN-STUB

A return code of BAQ-ERROR-IN-STUB indicates a communication error with the subsystem. The following table lists the status codes returned by the communication stub and the corresponding error messages returned in BAQ-STATUS-MESSAGE. In addition to returning the error in BAQ-STATUS-MESSAGE the communication stub for CICS logs the error in TDQUEUE BAQQ.

For an explanation and the suggested user action for each error message, see the following topics:

- [“Communication stub \(CICS\) messages” on page 725](#)
- [“Communication stub \(IMS and z/OS\) messages” on page 732](#)

Table 95. Status code by communication stub when the return code is BAQ-ERROR-IN-STUB

Communication stub	Status code	Error message in BAQ-STATUS-MESSAGE
CICS	1	BAQT0001E [date time] URIMAP(<i>mapname</i>) is not found.
	2	BAQT0002E [date time] URIMAP(<i>mapname</i>) is not available.
	3	BAQT0003E [date time] Hostname in URIMAP(<i>mapname</i>) cannot be resolved.
	4	BAQT0004E [date time] Timeout occurred before a connection to the z/OS Connect EE server could be established.
	5	BAQT0005E [date time] Timeout occurred before the request could be sent to the z/OS Connect EE server.
	6	BAQT0006E [date time] Timeout occurred before a response could be received from the z/OS Connect EE server.
	7	BAQT0007E [date time] The length of the response message is invalid.
	8	BAQT0008E [date time] Socket error.
	9	BAQT0009E [date time] Incorrect client certificate.
	10	BAQT0010E [date time] All requested cipher codes were rejected.
	11	BAQT0011E [date time] An unexpected error occurred when BAQCSTUB processed a request for REST API [<i>REST API name</i>]. EXEC CICS [<i>CICS API name</i>] gave RESP [<i>RESP value</i>] and [<i>RESP2 value</i>].
	12	BAQT0012E [date time] The communication stub failed to connect the z/OS Connect EE server.
	13	BAQT0013E [date time] The size of the storage for the request message is invalid.
	14	BAQT0014E [date time] OAuth parameter username length is invalid.
	15	BAQT0015E [date time] OAuth parameter password length is invalid.
	16	BAQT0016E [date time] OAuth parameter client ID length is invalid.
	17	BAQT0017E [date time] OAuth parameter client secret length is invalid.
	18	BAQT0018E [date time] The communication stub is misused.
	19	BAQT0019E [date time] An error occurred in the communication stub [<i>reason</i>].
	20	BAQT0020E [date time] The communication stub currently in use cannot process the [<i>name</i>] data structure because [<i>element</i>] has an invalid value [<i>value</i>].
	21	BAQT0021E [date time] TOKEN parameter username length is invalid.
	22	BAQT0022E [date time] TOKEN parameter password length is invalid.
	23	BAQT0023E [date time] The communication stub cannot process the [<i>name</i>] data structure because the pointer is null.
	24	BAQT0024E [date time] The communication stub cannot process the [<i>name</i>] data structure because the [<i>element</i>] element is not defined.
	25	BAQT0025E [date time] The communication stub cannot write message [<i>number</i>] because the message text is too long.
	26	BAQT0026E [date time] BAQ-ZCON-SERVER-URI URIMAP <i>urimap</i> does not match CICS resource naming standards.
	27	BAQT0027E [date time] The BAQ-ZCON-SERVER-URI URIMAP z/OS application maximum of 8 characters is allowed.

Table 95. Status code by communication stub when the return code is BAQ-ERROR-IN-STUB (continued)

Communication stub	Status code	Error message in BAQ-STATUS-MESSAGE
IMS and other non-CICS z/OS applications	1	BAQI0001E: z/OS Connect EE server URI [<i>uri</i>] is invalid. [<i>error_detail</i>]
	2	BAQI0002E: z/OS Connect EE port number [<i>port</i>] is invalid. [<i>HWTHXXXX_error</i>]
	3	BAQI0003E: Unable to connect to the z/OS Connect EE server URI [<i>uri</i>] on port [<i>port</i>]. [<i>HWTHXXXX_error</i>]
	4	BAQI0004E: Configuration property [<i>property_name</i>] was not specified.
	5	BAQI0005E: Unable to send request to or receive response from the z/OS Connect EE server. [<i>HWTHXXXX_error</i>]
	6	BAQI0006E: Unable to set property [<i>property_name</i>] to value [<i>value</i>]. [<i>HWTHXXXX_error</i>]
	7	BAQI0007E: The specified pointer to the storage for the response message is null.
	8	BAQI0008E: z/OS Connect EE server URI [<i>uri</i>] is greater than 255 characters. [<i>HWTHXXXX_error</i>]
	9	BAQI0009E: The name of the API is not specified or the length is invalid.
	10	BAQI0010E: The path to the API is not specified or the length is invalid.
	11	BAQI0011E: The HTTP method to invoke the API is not specified or the length is invalid.
	12	BAQI0012E: The specified storage size for the request message is invalid.
	13	BAQI0013E: The specified storage size for the response message is invalid.
	14	BAQI0014E: Verbose value [<i>value</i>] is invalid.
	15	BAQI0015E: Value for BAQVERBOSE_DEST_HWT [<i>value</i>] is invalid.
	16	BAQI0016E: Timeout value [<i>value</i>] is invalid. [<i>HWTHXXXX_error</i>]
	17	BAQI0017E: Username [<i>user_name</i>] is invalid. Length is greater than the maximum of [<i>number</i>] characters. [<i>error_detail</i>]
	18	BAQI0018E: Password is invalid. Length is greater than the maximum of [<i>number</i>] characters. [<i>error_detail</i>]
	19	BAQI0019E: Unable to disconnect and close the socket to the z/OS Connect EE server. [<i>HWTHXXXX_error</i>]
	20	BAQI0020E: Unable to terminate the connection to the z/OS Connect EE server. [<i>HWTHXXXX_error</i>]
	21	BAQI0021E: OAuth parameter username length is invalid.
	22	BAQI0022E: OAuth parameter password length is invalid.
	23	BAQI0023E: OAuth parameter client ID length is invalid.
	24	BAQI0024E: OAuth parameter client secret length is invalid.
	25	BAQI0025E: The communication stub is misused.
	26	BAQI0026E: Unable to request additional memory storage..
	27	BAQI0027E: The communication stub currently in use cannot process the BAQ* data structure.
	28	BAQI0028E: Parameter error - a NULL value is defined for [<i>parameter</i>]
	29	BAQI0029E: Internal error - Storage allocation error.

Related tasks

[“Developing a z/OS application to call APIs” on page 632](#)

Testing the z/OS application for API calls

Ensure that error handling code is added to your application to process the return code that is captured in the communication stub response information structure, use BAQ-RESPONSE-INFO for COBOL or BAQ_RESPONSE_INFO for PL/I.

Before you begin

- Your z/OS subsystem must be configured to use the communication stub that handles the communication and connection between the z/OS subsystem and the z/OS Connect EE server. For more information about configuration, see [“Configuring z/OS Connect EE to support API requesters” on page 304](#).
- The generated API requester archive (.ara file) must be deployed to the API requester folder. This file contains the data mappings between the JSON format and binary format. For more information about deploying the API requester archive, see [“Deploying the API requester to z/OS Connect EE” on page 631](#).

About this task

You must include the generated data structures and API information file. Then compile and link-edit your application.

Procedure

1. Add the following data sets to the JCL for compiling your application:

- The data set that contains the generated data structures.
- The data set that contains the generated API information file.
- The hlq.SBAQCOB data set that contains the BAQRINFO data structure for COBOL programs or the hlq.SBAQPLI data set that contains the BAQRINFP data structure for PL/I programs.

2. Compile and link-edit your program.

Important:

- As the communication stub is compiled as reentrant, you must compile and link-edit your application with the RENT option.
- You can enable your z/OS applications to make dynamic calls to the communication stub module so that you don't need to recompile your application when BAQCSTUB is updated in subsequent maintenance releases. For more information, see [“Linking by dynamic calls” on page 645](#).

Chapter 22. How to manage services

The administration interface supports various service administration tasks such as deploying, starting, stopping, or deleting a service, as well as getting details, statistics, or the request or response schema of a service. For service archives, you can configure the server for automatic service deployment or to override service archive properties upon deployment.

Administering services with the administration interface

The administration interface for services is available in paths under /zosConnect/services.

Most administration tasks are supported by the RESTful administration interface.

z/OS Connect EE supports V1.2.0 of the RESTful administration interface. This version includes new and enhanced methods and operations that are not available in earlier versions, duplicating some non-RESTful administration functions to provide a more natural RESTful interface. A Swagger document that describes the RESTful administration interface for z/OS Connect EE is available on the URI /zosConnect/api-docs.

Administration interface supported operations

Table 96 on page 655 lists the methods and administrative tasks that are supported by the RESTful administration interface.

Table 96. Methods and administrative tasks supported by the RESTful administration interface		
Method	Tasks supported by the RESTful administration interface	Enhancements
GET	“Get a list of services” on page 656	None
	“Get details of a service” on page 657	v1.1.0, v1.2.0
	“Get the status of a service” on page 659	v1.1.0
	“Get the request schema of a service” on page 660	v1.1.0
	“Get the response schema of a service” on page 661	v1.1.0
POST	“Deploy a service” on page 663	v1.1.0
PUT	“Update a service” on page 668	v1.1.0
	“Change the status of a service” on page 667	v1.1.0
DELETE	“Delete a service” on page 669	None

Table 97 on page 656 lists the methods and administrative tasks from V1.0.0 that are supported. Some administrative tasks are supported only in the non-RESTful administration interface, some only in the RESTful administration interface, and some are supported in both.

Table 97. V1.0.0 methods and tasks supported in the administration interface

Method	Administration tasks supported	Comments
GET	“Get a list of services” on page 656	
	“Get details of a service” on page 657	
	“Get the status of a service” on page 659	Supported in both the RESTful administration interface and the non-RESTful administration interface, but with a different request URI and query string.
	“Get the request schema of a service” on page 660	Supported in both the RESTful administration interface and the non-RESTful administration interface, but with a different request URI and query string.
	“Get the response schema of a service” on page 661	Supported in both the RESTful administration interface and the non-RESTful administration interface, but with a different request URI and query string.
	“Get the statistics of a service” on page 661	Supported only in the non-RESTful administration interface.
	“Get the statistics for multiple services” on page 662	Supported only in the non-RESTful administration interface.
POST	“Change the status of a service” on page 667	

Important:

- The `zosconnect_services` element must be added to the `server.xml` configuration file, even if no attributes are specified. In this case, the attributes are set to default values. If the `zosconnect_services` element is not added, a BAQR7070E error message is generated when the administration interface is called. For more information, see [“zosconnect_services” on page 774](#).

Notes for all calls

- When a service name is required for a call, if the service name contains forward slashes, they must be escaped by using %2F in these calls. For example, if the service name is MyService/v1, it must be provided as MyService%2Fv1.
- All successful calls return HTTP status code 200 unless otherwise specified.
- Any call that uses an HTTP method with a URI not mentioned in these topics returns an error 405 Method Not Allowed.
- If an error occurs during request processing, an appropriate HTTP status code and the following JSON is returned:

```
{
    "errorMessage": "{message}",
    "errorDetails": "{details}"
}
```

`errorDetails` is optional and is returned only for some error scenarios.

Get a list of services

Use the HTTP method GET to obtain a list of the z/OS Connect EE services installed in the run time.

HTTP method

GET

URI

/zosConnect/services

Description

Gets a list of the z/OS Connect EE services installed in the runtime, including some basic information. Services are included in the list whether they are started or stopped.

Security

Users with Admin, Operations or Reader authority can get a list of services, users with Invoke authority cannot. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Note: A user must be a member of a global group to be able to see a list of services. If individual services have also configured the authorization interceptor explicitly with the interceptorsRef attribute of the service or zosConnectService element, and defined service scope groups, then the list of installed services returned will be restricted to those for which the authenticated user ID is a member of both a global and a service scope Admin, Operations or Reader group

Response body

```
{  
  "zosConnectServices": [  
    {  
      "ServiceName": "<service name>",  
      "ServiceDescription": "<service description>",  
      "ServiceProvider": "<service provider>",  
      "ServiceURL": "<service URL>"  
    },  
    ... (repeats)  
  ]  
}
```

Example response body

```
{  
  "zosConnectServices": [  
    {  
      "ServiceName": "recordOpsCreate",  
      "ServiceDescription": "Creates a new record",  
      "ServiceProvider": "SAMPLE-1.0",  
      "ServiceURL": "https://host:port/zosConnect/services/recordOpsCreate"  
    },  
    {  
      "ServiceName": "recordOpsDelete",  
      "ServiceDescription": "Deletes an existing record",  
      "ServiceProvider": "SAMPLE-1.0",  
      "ServiceURL": "https://host:port/zosConnect/services/recordOpsDelete"  
    },  
    {  
      "ServiceName": "patient",  
      "ServiceDescription": "Patient lookup service",  
      "ServiceProvider": "imsmobile-2.0",  
      "ServiceURL": "https://host:port/zosConnect/services/patientService"  
    }  
  ]  
}
```

Get details of a service

Use the HTTP method GET to obtain information about a specific z/OS Connect EE service.

HTTP method

GET

URI

/zosConnect/services/{serviceName}

If the service name contains forward slashes, they must be escaped by using %2F. For example, if the service name is MyService/v1, it must be provided as MyService%2Fv1.

Description

Gets the details of the requested z/OS Connect EE service.

Security

Users with Admin, Operations or Reader authority can get details of a service, users with Invoke authority cannot. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Response body

The output is returned in two parts. The first part contains the z/OS Connect EE configuration parameters and the second part has the configuration that is returned by the service provider implementation.

Note: The enhanced administration interface V1.2.0 includes version information for services in the response body.

```
{  
  "zosConnect": {  
    "serviceName": "<service name>",  
    "serviceDescription": "<service description>",  
    "serviceProvider": "<service provider>",  
    "version": "<version>",  
    "serviceURL": "<service URL>",  
    "serviceInvokeURL": "<service invocation URL>",  
    "dataXformProvider": "<data transformation provider>",  
    "serviceStatus": "<service status>"  
  },  
  "<service name>": {  
    ( ... <service provider-specific information> ... )  
  }  
}
```

Example response body

The following sample shows the JSON payload that is returned from the CICS service provider:

```
{  
  "zosConnect": {  
    "serviceName": "MyService",  
    "serviceDescription": "",  
    "serviceProvider": "CICS service provider",  
    "version": "1.2",  
    "serviceURL": "http://host:port/zosConnect/services/MyService",  
    "serviceInvokeURL": "http://host:port/zosConnect/services/MyService?action=invoke",  
    "dataXformProvider": "zosConnectWVXform-10",  
    "serviceStatus": "Started"  
  },  
  "MyService": {  
    "transidUsage": "EIB_AND_MIRROR",  
    "transid": "CSMI",  
    "program": "MyProg",  
    "connectionRef": "MyConnection",  
    "ccsid": "37"  
  }  
}
```

The following sample output is from the IMS service provider:

```
{  
  "zosConnect": {  
    "serviceName": "patient",  
    "serviceDescription": "Patient lookup service",  
    "serviceProvider": "imsmobile-2.0",  
    "version": "2.0",  
    "ServiceURL": "https://server1.mycom.com:53633/zosConnect/services/patientService",  
    "serviceInvokeURL": "https://server1.mycom.com:53633/zosConnect/services/  
patientService?action=invoke",  
    "dataXformProvider": "DATA_UNAVAILABLE",  
    "serviceStatus": "Started"  
  },  
  "patient": {  
    "imsServiceType": "ims-sar",  
    "serviceDescription": "Patient lookup service",  
  }  
}
```

```

        "id": "patient",
        "tranCode": "IVTNO",
        "serviceProviderName": "IMS service provider",
        "status": "Started"
    }
}

```

Errors

The following error can occur:

404 Not found
The service was not found.

Get the status of a service

Use the HTTP method GET to obtain information, including the status, about a specific z/OS Connect EE service.

HTTP method

GET

URI

/zosConnect/services/{*serviceName*}

If the service name contains forward slashes, they must be escaped by using %2F. For example, if the service name is MyService/v1, it must be provided as MyService%2Fv1.

Note: Enhanced in V1.1.0 of the administration interface. Service status is now reported.

Description

Gets the details, including the status, of the requested z/OS Connect EE service.

Security

Users with Admin, Operations or Reader authority can get the status of a service; users with Invoke authority cannot. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Response body

Note: The enhanced administration interface V1.2.0 includes version information for services in the response body.

The output is returned in two parts. The first part contains the z/OS Connect EE configuration parameters and the second part has the configuration that is returned by the service provider implementation.

```

{
    "zosConnect": {
        "serviceName": "<service name>",
        "serviceDescription": "<service description>",
        "serviceProvider": "<service provider>",
        "version": "<version>",
        "serviceURL": "<service URL>",
        "serviceInvokeURL": "<service invocation URL>",
        "dataXformProvider": "<data transformation provider>",
        "serviceStatus": "<service status>"
    },
    "<service name>": {
        ( ... <service provider-specific information> ... )
    }
}

```

Example response body

The following sample shows the JSON payload that is returned from the SAMPLE-1.0 service provider:

```
{
    "zosConnect": {
        "serviceName": "recordOpsCreate",
        "serviceDescription": "Creates a new record",
        "serviceProvider": "SAMPLE-1.0",
        ...
    }
}
```

```

    "version": "1.1.0",
    "serviceURL": "https://host:port/zosConnect/services/recordOpsCreate",
    "serviceInvokeURL": "https://host:port/zosConnect/services/recordOpsCreate?action=invoke",
    "dataXformProvider": "jsonByte-1.0",
    "serviceStatus": "Started"
},
recordOpsCreate: {
    "targetProgram": "CREATREC",
    "timeout": "300ms"
}
}

```

Note: Alternatively, use the HTTP method GET with the **action=status** query parameter to retrieve the status of a service.

```
/zosConnect/services/{serviceName}?action=status
```

The sample response body is as follows:

```
{
  "zosConnect": {
    "serviceName": "recordOpsCreate",
    "serviceDescription": "Creates a new record",
    "serviceProvider": "SAMPLE-1.0",
    "version": "1.1.0",
    "serviceURL": "https://host:port/zosConnect/services/recordOpsCreate",
    "serviceInvokeURL": "https://host:port/zosConnect/services/recordOpsCreate?
action=invoke",
    "dataXformProvider": "jsonByte-1.0",
    "serviceStatus": "Started"
  }
}
```

Related information

["Change the status of a service" on page 667](#)

Get the request schema of a service

Use the HTTP method GET to retrieve the request schema of a service.

HTTP method

GET

URI

```
/zosConnect/services/{serviceName}/schema/request
```

If the service name contains forward slashes, they must be escaped by using %2F. For example, if the service name is MyService/v1, it must be provided as MyService%2Fv1.

Note: Added in V1.1.0 of the administration interface.

Description

Gets the request schema of the specified service.

Security

Users with Admin, Operations or Reader authority can get the request or response schema of a service, users with Invoke authority cannot. For more information about user authorization, see ["Overview of z/OS Connect EE security" on page 331](#).

Response body

```
{
<Request schema as returned by the service provider-specific data transformer>
}
```

Note:

In V1.0.0 of the administration interface, retrieving the schema requires the use of the **action=getRequestSchema** query string, which is not as light-weight as the URI approach added in V1.1.0.

```
https://myhost:myport/zosConnect/services/myService?action=getRequestSchema
```

If you use the getRequestSchema call to find JSON schema files, zosconnect_zosConnectDataXform automatically appends _request to the service name.

Get the response schema of a service

Use the HTTP method GET to retrieve the response schema of a service.

HTTP method

GET

URI

```
/zosConnect/services/{serviceName}/schema/response
```

If the service name contains forward slashes, they must be escaped by using %2F. For example, if the service name is MyService/v1, it must be provided as MyService%2Fv1.

Note: Added in V1.1.0 of the administration interface.

Description

Gets the response schema of the specified service.

Security

Users with Admin, Operations or Reader authority can get the request or response schema of a service, users with Invoke authority cannot. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Response body

```
{  
  <Response schema as returned by the service provider-specific data transformer>  
}
```

Note:

In V1.0.0 of the administration interface, retrieving the schema requires the use of the **action=getResponseSchema** query string, which is not as light-weight as the URI approach added in V1.1.0.

```
https://myhost:myport/zosConnect/services/myService?action=getRequestSchema
```

If you use the getRequestSchema call to find JSON schema files, zosconnect_zosConnectDataXform automatically appends _request to the service name.

Get the statistics of a service

You can use the HTTP method GET with the **action=getStatistics** query parameter to retrieve the statistics of a service.

Statistics include z/OS Connect EE data for a service such as InvokeRequestCount, and a TimeOfRegistrationWithZosConnect, along with any other statistics returned by the service provider by using the getStatistics() SPI implementation in the provider.

Note: Statistics are only available for z/OS Connect EE services and not all z/OS Connect EE service providers provide statistics. For example the CICS service provider, IBM MQ service provider, and REST client service provider do not provide statistics.

HTTP method

GET

URI

/zosConnect/services/{serviceName}?action=getStatistics

If the service name contains forward slashes, they must be escaped by using %2F. For example, if the service name is MyService/v1, it must be provided as MyService%2Fv1.

Description

Gets the status of the requested service.

Security

Users with Admin or Operations authority can get the statistics of a service, users with Invoke or Reader authority cannot. For more information about user authorization, see “[Overview of z/OS Connect EE security](#)” on page 331.

Sample response body

```
{ "recordOpsCreate": { "ServiceProvider": "SAMPLE-1.0", "InvokeRequestCount": "100", "TimeOfRegistrationWithZosConnect": "<yyyy-mm-dd hh:mm:ss:mmm tzn>" "ServiceStatistics": { .. <JSON name value pairs showing statistical information about the service returned.> } } }
```

where yyyy-mm-dd is the date, hh:mm:ss:mmm time and tzn is the time zone. For example:
2017-11-13 17:28:28:589 GMT.

Get the statistics for multiple services

Use the HTTP method GET to retrieve the statistics of all services that are defined for a specific service provider by using the /zosConnect/operations/getStatistics URI and the **provider** query parameter.

Statistics include z/OS Connect EE data for a service such as InvokeRequestCount, and a TimeOfRegistrationWithZosConnect, along with any other statistics returned by the service provider by using the getStatistics() SPI implementation in the provider. Statistics for a particular service can be retrieved through a /zosConnect/operations or an action=request call. /zosConnect/operations requests offer more flexibility because the product can retrieve statistics for all services. If the authorization interceptor is enabled, the product returns statistics for only those services that the user can request. For more information about user authorization, see “[Overview of z/OS Connect EE security](#)” on page 331.

Note: Statistics are only available for z/OS Connect EE services and not all z/OS Connect EE service providers provide statistics. For example the CICS service provider and REST client service provider do not provide statistics.

HTTP method

GET

URI

/zosConnect/operations/getStatistics?provider=<service provider>

Description

Gets the statistics for all services of the specified service provider.

Security

Users with Admin or Operations authority can get the statistics, users with Invoke or Reader authority cannot. For more information about user authorization, see “[Overview of z/OS Connect EE security](#)” on page 331.

Note: A user must be a member of a global group to be able to see a list of services. If individual services have also configured the authorization interceptor explicitly with the interceptorsRef attribute of the service or zosConnectService element, and defined service scope groups, then

the list of installed services returned will be restricted to those for which the authenticated user ID is a member of both a global and a service scope Admin or Operations group

Sample call

```
https://myhost:myport/zosConnect/operations/getStatistics?provider=imsmobile-2.0
```

Sample response body

```
{ "recordOpsCreate": { "ServiceProvider": "imsmobile-2.0", "InvokeRequestCount": "100", "TimeOfRegistrationWithZosConnect": "<yyyy-mm-dd hh:mm:ss:mmm tzn>", "ServiceStatistics": { .. <JSON name value pairs showing statistical information about the service returned.> } } }
```

Alternatively, service statistics can be obtained by using an HTTP GET request and the /zosConnect/operations/getStatistics URI:

```
https://host:port/zosConnect/operations/getStatistics
```

The following sample shows the JSON payload that is returned:

```
{ "zosConnectStatistics": [ { "recordOpsCreate": { "ServiceProvider": "SAMPLE-1.0", "InvokeRequestCount": "100", "TimeOfRegistrationWithZosConnect": "<yyyy-mm-dd hh:mm:ss:mmm tzn>", "ServiceStatistics": { .. <JSON name value pairs showing statistical information about the service returned.> } } }, { "recordOpsDelete": { "ServiceProvider": "SAMPLE-1.0", "InvokeRequestCount": "100", "TimeOfRegistrationWithZosConnect": "<yyyy-mm-dd hh:mm:ss:mmm tzn>", "ServiceStatistics": { .. <JSON name value pairs showing statistical information about the service returned.> } } } ] }
```

If no services are registered with z/OS Connect EE, the output resembles the following example:

```
{ "zosConnectStatistics": "" }
```

Deploy a service

Use the HTTP method POST to deploy your service to make it available for use.

HTTP method

POST

URI

/zosConnect/services

If the service name contains forward slashes, they must be escaped by using %2F. For example, if the service name is MyService/v1, it must be provided as MyService%2Fv1.

Note: Added in V1.1.0 of the administration interface.

Description

Deploys a service into z/OS Connect EE.

Security

Users with Admin or Operations authority can deploy a service, users with Invoke or Reader authority cannot. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Note: A user must be a member of a global group to be able to deploy a service.

Request body

The service archive file. The *Content-Type* for the request is application/zip.

Response body

Note: The enhanced administration interface V1.2.0 includes version information for services in the response body.

```
{  
  "zosConnect": {  
    "serviceName": "<service name>",  
    "serviceDescription": "<service description>",  
    "serviceProvider": "<service provider>",  
    "version": "<version>",  
    "serviceURL": "<service URL>",  
    "serviceInvokeURL": "<service invocation URL>",  
    "dataXformProvider": "<data transformation provider>",  
    "serviceStatus": "<service status>"  
  },  
  "<service name>": {  
    .. <service provider-specific information>  
  }  
}
```

Example response body

```
{  
  "zosConnect": {  
    "serviceName": "patient",  
    "serviceDescription": "Patient lookup service",  
    "serviceProvider": "imsmobile-2.0",  
    "version": "1.1.0"  
    "ServiceURL": "https://server1.mycom.com:53633/zosConnect/services/patientService",  
    "serviceInvokeURL": "https://server1.mycom.com:53633/zosConnect/services/  
patientService?action=invoke",  
    "dataXformProvider": "DATA_UNAVAILABLE",  
    "serviceStatus": "Started"  
  },  
  "patient": {  
    "imsServiceType": "ims-sar",  
    "serviceDescription": "Patient lookup service",  
    "id": "patient",  
    "tranCode": "IVTNO",  
    "serviceProviderName": "imsmobile-2.0",  
    "status": "Started"  
  }  
}
```

Errors

The following errors can occur:

400 Bad request
Invalid or missing service archive file

409 Conflict
z/OS Connect service install failed. The service name {service name} is already in use.

415 Unsupported media type
An unsupported media type of application/json was specified under request URL {service URL}.

500 Internal Server Error
Server issue, might require administrator intervention.

503 Service unavailable

A 503 error could mean that the service provider that is specified in the service archive is not available on the server, or the **zosconnect_services** element in not present in the `server.xml` configuration file.

Setting the initial status of a new service

When you deploy a new service, the default initial status is set to Started. You can optionally set the initial status to Stopped by appending a query string to the HTTP POST method:

HTTP method

POST

URI

```
/zosConnect/services?status=stopped
```

Description

Deploys a new service into z/OS Connect EE and sets the status to Stopped. When a service is stopped, requests invoking the service will fail. Administration requests to `/zosConnect/services/{serviceName}` will still function as normal and the service will appear in the list returned by a GET request to `/zosConnect/services`. To start the service after it has been deployed, see [“Change the status of a service” on page 667](#).

Note: The enhanced administration interface V1.2.0 includes version information for services in the response body.

Security

Users with Admin or Operations authority can set the initial status of an API, users with Invoke or Reader authority cannot. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Request body

Service archive file to be deployed.

Example request

To deploy a service and set the initial status to stopped:

```
/zosConnect/services?status=stopped
```

Response body

```
{  
  "zosConnect": {  
    "serviceName": "<service name>",  
    "serviceDescription": "<service description>",  
    "serviceProvider": "<service provider>",  
    "version": "1.1.0",  
    "serviceURL": "<service URL>",  
    "serviceInvokeURL": "<service invocation URL>",  
    "dataXformProvider": "<data transformation provider>",  
    "serviceStatus": "<service status>"  
  },  
  "<service name>": {  
    "imsServiceType": "<service type>",  
    "serviceDescription": "<service description>",  
    "id": "<service ID>",  
    "tranCode": "<IMS transaction code invoked by the service if service provider is  
IMS>",  
    "serviceProviderName": "<service provider>",  
    "status": "<service status>"  
  }  
}
```

```
}
```

Errors

The following error can occur:

```
400 Bad request  
Unknown status specified
```

Set the initial state of a service

Use the HTTP method POST and the **status** query string option to set the initial status of a service. If the service archive file is not usable, the operation fails.

The status of a service can be either started, or stopped. If the status information is not available from the service provider, the service would be in an unknown status.

HTTP method

POST

URI

```
/zosConnect/services/{serviceName}?status=started|stopped
```

If the service name contains forward slashes, they must be escaped by using %2F. For example, if the service name is MyService/v1, it must be provided as MyService%2Fv1.

Description

Sets the initial status of a service upon service deployment or service updates. When a service is stopped, new requests will fail but existing requests will be allowed to complete. Administration requests to /zosConnect/services/{serviceName} will still function normally and the service will appear in the list returned by the /zosConnect/services request.

Note:

The Service Provider SPI is called by z/OS Connect EE to notify it that these actions were requested. The method names that are in the SPI for these actions are **stop()**, **start()**, and **status()**.

z/OS Connect EE does not persist any state that is related to the service and instead delegates this state to the service provider.

Security

If the authorization interceptor provided by z/OS Connect EE is enabled in the service provider, users with Admin or Operations authority can change the status of a service. Users with Invoke or Reader authority cannot. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Request body

To change the status of a service, the request body should have no content. If a service archive is in the request body, the request is to update the service and set the initial status after the update. For more information, see [“Update a service” on page 668](#).

Response body

Note: The enhanced administration interface V1.2.0 includes version information for services in the response body.

```
{  
    "zosConnect": {  
        "serviceName": "<service name>",  
        "serviceDescription": "<service description>",  
        "serviceProvider": "<service provider>",  
        "version": "<version>",  
        "serviceURL": "<service URL>",  
        "serviceInvokeURL": "<service invocation URL>",  
        "dataXformProvider": "<data transformation provider>",  
        "serviceStatus": "<service status>"  
    }  
}
```

Errors

400 Bad request
Unknown status specified

Change the status of a service

Use the HTTP method PUT and the **status** query string option to change the status of a service. The PUT method would also update the service if the service definition has changed.

The status of a service can be either started, or stopped. If the status information is not available from the service provider, the service would be in an unknown status.

HTTP method

PUT

URI

/zosConnect/services/{*serviceName*}?status=started|stopped

If the service name contains forward slashes, they must be escaped by using %2F. For example, if the service name is MyService/v1, it must be provided as MyService%2Fv1.

Description

Starts or stops the existing service. When a service is stopped, new requests will fail but existing requests will be allowed to complete. Administration requests to /zosConnect/services/{*serviceName*} will still function normally and the service will appear in the list returned by the /zosConnect/services request.

Note:

1. The Service Provider SPI is called by z/OS Connect EE to notify it that these actions were requested. The method names that are in the SPI for these actions are **stop()**, **start()**, and **status()**.
2. z/OS Connect EE does not persist any state that is related to the service and instead delegates this state to the service provider.
3. The enhanced administration interface V1.2.0 includes version information for services in the response body.

Security

If the authorization interceptor provided by z/OS Connect EE is enabled in the service provider, users with Admin or Operations authority can change the status of a service. Users with Invoke or Reader authority cannot. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Request body

To change the status of a service, the request body should have no content. If a service archive is in the request body, the request is to update the service and set the initial status after the update. For more information, see [“Update a service” on page 668](#).

Response body

```
{  
  "zosConnect": {  
    "serviceName": "<service name>",  
    "serviceDescription": "<service description>",  
    "serviceProvider": "<service provider>",  
    "version": "<version>",  
    "serviceURL": "<service URL>",  
    "serviceInvokeURL": "<service invocation URL>",  
    "dataXformProvider": "<data transformation provider>",  
    "serviceStatus": "<service status>"  
  }  
}
```

Note: In V1.0.0 of the administration interface, setting the service status requires the use of HTTP POST or PUT method with the **action=start** or **action=stop** query string, as shown in the following example:

```
https://myhost:myport/zosConnect/services/myService?action=start|stop
```

Errors

400 Bad request
Unknown status specified

Related information

[“Get the status of a service” on page 659](#)

Update a service

Use the HTTP method PUT to update a z/OS Connect EE service.

By default, a service has the status started. However, it can be set to a specific initial status by appending a query string to the URI with a status. For example:

```
/zosConnect/services/{serviceName}?status=stopped
```

HTTP method

PUT

URI

```
/zosConnect/services/{serviceName}
```

If the service name contains forward slashes, they must be escaped by using %2F. For example, if the service name is MyService/v1, it must be provided as MyService%2Fv1.

Note: Added in V1.1.0 of the administration interface.

Description

Updates the named service in z/OS Connect EE.

Note: The service needs to be stopped before updating.

Security

Users with Admin or Operations authority can update a service, users with Invoke or Reader authority cannot. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Request body

The service archive file. The content type for the request is application/zip.

Response body

Note: The enhanced administration interface V1.2.0 includes version information for services in the response body.

```
{
  "zosConnect": {
    "serviceName": "<service name>",
    "serviceDescription": "<service description>",
    "serviceProvider": "<service provider>",
    "version": "<version>",
    "serviceURL": "<service URL>",
    "serviceInvokeURL": "<service invocation URL>",
    "dataXformProvider": "<data transformation provider>",
    "serviceStatus": "<service status>"
  },
  "<service name>": {
    .. <service provider-specific information>
  }
}
```

Example response body

```
{  
  "zosConnect": {  
    "serviceName": "patient",  
    "serviceDescription": "Patient lookup service",  
    "serviceProvider": "imsmobile-2.0",  
    "version": "1.1.0"  
    "ServiceURL": "https://server1.mycom.com:53633/zosConnect/services/patientService",  
    "serviceInvokeURL": "https://server1.mycom.com:53633/zosConnect/services/  
patientService?action=invoke",  
    "dataXformProvider": "DATA_UNAVAILABLE",  
    "serviceStatus": "Started"  
  },  
  "patient": {  
    "imsServiceType": "ims-sar",  
    "serviceDescription": "Patient lookup service",  
    "id": "patient",  
    "tranCode": "IVTNO",  
    "serviceProviderName": "imsmobile-2.0",  
    "status": "Started"  
  }  
}
```

Errors

The following errors can occur:

400 Bad request
Invalid or missing service archive

409 Conflict
A z/OS Connect service must be stopped before it can be updated.

415 Unsupported Media Type
Content-Type is not application/zip:

500 Internal Server Error
Server issue, might require administrator intervention.

Delete a service

Use the HTTP method DELETE to delete a z/OS Connect EE service.

HTTP method

DELETE

URI

/zosConnect/services/{*serviceName*}

If the service name contains forward slashes, they must be escaped by using %2F. For example, if the service name is MyService/v1, it must be provided as MyService%2Fv1.

Note: Added in V1.1.0 of the administration interface.

Description

Uninstalls the named service from z/OS Connect EE, and deletes the service archive from the file system. A service must be in stopped state to be deleted.

Security

Users with Admin or Operations authority can delete a service, users with Invoke or Reader authority cannot. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Response body

```
{  
  "name": "{service name}"  
}
```

Example response body

```
{  
  "name": "patient"  
}
```

Errors

The following errors can occur:

409 Conflict

The z/OS Connect Service status is started. The z/OS Connect Service cannot be deleted unless it is stopped.

500 Internal Server Error

Server issue, might require administrator intervention.

Administering service archives

Use this information to learn how to manage your service archives.

Service archive (.sar) files are deployed to a z/OS Connect EE server by copying them to a location that is defined in the `server.xml` configuration file.

Automated service archive management

Deploy and undeploy your service archives automatically when you add or remove them from the services directory.

You can configure z/OS Connect EE to deploy a service archive automatically when you copy your service archive file to the location where your services archives are stored. For example, `resources/zosconnect/services`. When you remove the service archive file from the services directory, the service is automatically undeployed.

To deploy a service, you add the .sar archive file into the services location directory. Your user ID must have the following permissions:

- Read, write, and execute permission to the services location directory.
- Read and execute permission to all parent directories of the services location directory.

To update an existing service, you replace the .sar archive file in the services location directory. To delete an existing service, you delete the .sar archive file from the services location directory. To perform these tasks, your user ID must have the following permissions:

- Read, write, and execute permission to the services location directory.
- Read and execute permission to all parent directories of the services location directory.
- Write access to the .sar archive file.

The z/OS Connect EE server's user ID must have read and write access to the .sar archive file. Use the UNIX System Services `chmod` command to assign the appropriate permissions to the directories and archive files.

The services directory is monitored for changes to the service archives. You can control how and when z/OS Connect EE reacts to these changes by setting the `updateTrigger` and `pollingRate` attributes of the `zosconnect_services` element of the `server.xml` configuration file. The default value for `pollingRate` is 5000 mSecs. For more information, see ["zosconnect_services"](#) on page 774.

Note: Service definitions that contain override attributes remain in the `server.xml` configuration file after you delete or uninstall a service archive file.

When you copy a service archive file to the services directory, ensure that the service name is unique within the folder. For example, if `test1.sar` exists in the folder and contains a service named `test`, it must be removed before you copy `test2.sar`, which also contains a service named `test`. If both files are present when z/OS Connect EE starts, the service that is deployed is that of the first file that is read. When the other file is read, an error message is returned that the service is already deployed.

To update a service, replace the service archive file in the services directory. The file must have the same name. If the *updateTrigger* and *pollingRate* attributes of the **zosconnect_services** element are set, the service is installed automatically when the folder is polled.

If you copy a service archive to the services directory where a service of the same name is already configured, the new service will not be installed. When the server is restarted, the new service might install first, but this is unpredictable.

Overriding service archive properties

Control the properties of a service archive when you deploy it.

When you deploy a service archive that was supplied by a service provider, certain properties can be overridden.

Properties are specified as nested elements inside a service element. For example:

```
<service name="myService" requireAuth="true" requireSecure="true" runGlobalInterceptors="false">
  <property name="name1" value="value1"/>
  <property name='name2' value='value2' />
  ....
  <property name='namex' value='valuex' />
</service>
```

You must manually edit properties in the `server.xml` configuration file. For more information about these properties, see [Service archive override properties](#).

If the service archive file is deleted or moved, the properties remain in the configuration file and must be manually removed.

Chapter 23. How to manage APIs

Use this information to learn how to manage your APIs.

APIs can be deployed to a z/OS Connect EE server by copying the API archive (.aar) file into the APIs directory, or by using the [API toolkit](#), the [RESTful administration interface](#), or the [API deployment utility](#). The API toolkit and RESTful administration interface can also be used to get information on the APIs that have been deployed and to stop an API from accepting requests.

Automated API management

Deploy and undeploy your APIs automatically when you add or remove them from the API directory.

This task is primarily applicable to a development environment. In a production environment, you should disable polling to improve performance, and use the MODIFY refresh command to trigger updates.

You can configure z/OS Connect EE to deploy an API automatically when you copy your API archive (.aar file) to the location where your APIs are stored. For example, `resources/zosconnect/apis`.

To deploy an API, you add the .aar archive file into the API location directory. Your user ID must have the following permissions:

- Read, write, and execute permission to the API location directory.
- Read and execute permission to all parent directories of the API location directory.

To update an existing API, you replace the .aar archive file in the API location directory. To delete an existing API, you delete the .aar archive file from the API location directory. To perform these tasks, your user ID must have the following permissions:

- Read, write, and execute permission to the API location directory.
- Read and execute permission to all parent directories of the API location directory.
- Write access to the .aar archive file.

The z/OS Connect EE server's user ID must have read and write access to the .aar archive file. Use the UNIX System Services **chmod** command to assign the appropriate permissions to the directories and archive files.

When you remove the API from the directory, it is automatically undeployed.

The API location is monitored for changes to the API. You can control how and when z/OS Connect EE reacts to these changes by setting the *updateTrigger* and *pollingRate* attributes of the **zosconnect_zosConnectAPIs** element of the configuration file `server.xml`. The default value for *pollingRate* is 5000 mSecs. For more information, see [“zosconnect_zosConnectAPIs” on page 780](#).

Note: API definitions remain in the `server.xml` configuration file after you delete or uninstall an API. This situation causes a warning message to be written to the log until the API is reinstalled.

When you copy an .aar file to the apis directory, ensure that the API name is unique within the directory. For example, if `test1.aar` exists in the directory and contains the API `test`, it must be removed before you copy `test2.aar`, which also contains an API named `test`. If both files are present when z/OS Connect EE starts, one of the APIs is deployed. When the other file is read, an error message is returned that the API is already deployed.

To update an API, replace the .aar file in the apis directory. The file must have the same name. If the *updateTrigger* and *pollingRate* attributes of the **zosconnect_zosConnectAPIs** element are set, the API is installed automatically when the directory is polled.

Using the API Deployment utility

Use this information to learn how to deploy and maintain your APIs.

Related reference

[“apideploy command syntax” on page 792](#)

Deploy or undeploy APIs

Deploying an API

You must deploy an API to make it available to users.

Before you begin

Note:

- To issue the API Deployment utility commands, you must be a z/OS Connect EE server administrator with access to the OMVS shell.
- You can also deploy an API directly from the z/OS Connect EE API toolkit. See instructions in [“Deploying an API in the API toolkit” on page 595](#).

Follow these instructions to deploy an API using the API Deployment utility

- The API archive file must already reside on a UNIX System Services file system on the same z/OS LPAR where z/OS Connect EE is installed.
- When transferring the API archive file to the file system, use binary mode.
- After the API archive file is transferred, ensure that the owner of the file has read and write permission (a minimum of 644 by using the UNIX System Services **chmod** command).
- You must be a z/OS Connect EE server administrator to issue the API Deployment utility commands.
- The API deployment directory must already exist. The user that issues the API Deployment utility must have the permission to write to the API deployment directory.
- The **apideploy** command is a supplied z/OS UNIX command, so the administrator must have access to the OMVS shell to use the command.

Procedure

1. Go to the *<installation_path>/bin* directory.

2. Issue the following command:

```
apideploy -deploy -a <path_to_apiPackage.aar> -p <path_to_api_location>
```

Specify the relative or absolute path to the API archive file (-a) and to the API deployment location (-p). For example:

```
apideploy -deploy -a ./myApi/goodHealth.aar -p <WLP_USER_DIR>/servers/<serverName>/resources/zosconnect/apis
```

Results

The API that is defined in the API archive file is saved to the z/OS Connect EE server directory. A directory that is based on the API name is created in the API deployment directory.

What to do next

Depending on the configuration of your server, you might need to restart the server before the server is ready to process HTTP requests for the related z/OS Connect EE services from a REST client.

Related reference

[“apideploy command syntax” on page 792](#)

Deploy or undeploy APIs

Undeploying an API

When you undeploy an API, you remove it from the system and make it unavailable to users.

Before you begin

Follow these instructions to undeploy an API using the API Deployment utility.

Note:

1. To issue the API Deployment utility commands, you must be a z/OS Connect EE server administrator with access to the OMVS shell.
2. You can also undeploy an API directly from the z/OS Connect EE API toolkit. In the **z/OS Connect EE Servers** view in the editor, right-click the API and select **Stop API** to stop the API first. Then right-click the API and select **Remove API**.

Procedure

1. Go to the <installation_path>/bin directory.
2. Enter the following command:

```
apideploy -undeploy -a <path_to_apiPackage.aar> -p <path_to_deploy_location>
```

Specify the relative or absolute path to the API archive file (-a) and to the API deployment location (-p). For example:

```
apideploy -undeploy -a ./myApi/goodHealth.aar -p <WLP_USER_DIR>/servers/<serverName>/resources/zosconnect/apis
```

Results

The API that is defined in the API archive file is undeployed. The directory with the name of the API is removed from the API deployment directory.

What to do next

Depending on the configuration of your server, you might need to restart the server for the changes to take effect.

Related reference

[“apideploy command syntax” on page 792](#)

[Deploy or undeploy APIs](#)

Using the RESTful administration interface

The administration interface for APIs is available in paths under /zosConnect/apis and provides metadata for the APIs.

A Swagger document describing the RESTful administration interface for z/OS Connect EE is available on the URI /zosConnect/api-docs.

Notes for all calls

- All successful calls return HTTP status code 200 unless otherwise specified.
- URLs returned by any of the calls contain the protocol, server, and port from the request.
- Any call using a HTTP method with a URI not mentioned in these topics will return an error 405 Method Not Allowed.
- If an error occurs while processing a request, an appropriate HTTP status code and the following JSON will be returned:

```
{  
    "errorMessage": "{message}",  
    "errorDetails": "{details}"  
}
```

Note: `errorDetails` is optional and will only be returned for some error scenarios.

GET swagger documents

You can use the HTTP method GET to obtain a list of swagger documents.

HTTP method

GET

URI

/zosConnect/api-docs

Description

Gets a list of swagger documents for the z/OS Connect EE APIs installed in the runtime.

Security

Users with Admin, Operations or Reader authority can get a list of swagger documents, users with Invoke authority cannot. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Return body

```
{  
    "swagger": "2.0",  
    "info": {  
        "title": "z/OS Connect administration API",  
        "description": "Interface providing meta-data and life-cycle operations for z/OS Connect services, APIs and API requesters.",  
        "version": "1.1.0"  
    },  
    "schemes": [  
        "http",  
        "https"  
    ],  
    "consumes": [],  
    "produces": [  
        "application/json"  
    ],  
    "basePath": "/zosConnect",  
    "paths": {  
        "/apis": {  
            "get": {  
                "tags": [  
                    "APIs"  
                ],  
                "summary": "Returns a list of all the deployed z/OS Connect APIs",  
                "operationId": "getApis",  
                "responses": {  
                    "200": {  
                        "$ref": "#/responses/apis"  
                    },  
                    "default": {  
                        "$ref": "#/responses/defaultError"  
                    }  
                }  
            },  
            "post": {  
                "tags": [  
                    "APIs"  
                ],  
                "summary": "Deploys a new API into z/OS Connect",  
                "operationId": "createApi",  
                "consumes": [  
                    "application/zip"  
                ],  
                "parameters": [  
                    {  
                        "name": "apiPackage",  
                        "in": "body",  
                        "description": "API package.",  
                        "required": true,  
                        "schema": {  
                            "type": "string",  
                            "format": "binary"  
                        }  
                    },  
                    {  
                        "name": "status",  
                        "in": "query",  
                        "type": "string",  
                        "enum": ["PENDING", "DEPLOYED"]  
                    }  
                ]  
            }  
        }  
    }  
}
```

```

        "description": "The initial status of the API.",
        "type": "string",
        "enum": [
            "started",
            "stopped"
        ]
    },
    "responses": {
        "201": {
            "$ref": "#/responses/createApiDetail"
        },
        "400": {
            "$ref": "#/responses/400BadRequestCreate"
        },
        "409": {
            "$ref": "#/responses/409ConflictCreate"
        },
        "default": {
            "$ref": "#/responses/defaultError"
        }
    }
},
...
...
...
"tags": [
{
    "name": "APIs",
    "description": "Operations for working with APIs"
},
{
    "name": "Services",
    "description": "Operations for working with services"
},
{
    "name": "API Requesters",
    "description": "Operations that work with API Requesters."
}
]
}

```

GET a list of APIs

You can use the HTTP method GET to obtain a list of the z/OS Connect EE APIs installed in the runtime.

HTTP method

GET

URI

/zosConnect/apis

Description

Gets a list of the z/OS Connect EE APIs installed in the runtime, including some basic information and a URL for more detailed information. APIs are included in the list whether they are started or stopped.

Security

Users with Admin, Operations or Reader authority can get a list of APIs, users with Invoke authority cannot. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Note: A user must be a member of a global group to be able to see a list of all APIs.

Return body

```
{
  "apis": [
    {
        "name": "{name}",
        "version": "{version}",
        "description": "{API description}",
        "adminUrl": "http(s)://'{host}:{port}/zosConnect/apis/{name}"
    },
    ... repeats
  ]
}
```

```
    ]  
}
```

where **adminUrl** contains the URL where you can get more details for the specific API. For more information, see [“GET details of an API” on page 678](#).

Example body

```
{  
  "apis": [  
    {  
      "name": "HospitalAPI",  
      "version": "1.0",  
      "description": "API for Hospital app",  
      "adminUrl": "http://localhost:9080/zosConnect/apis/HospitalAPI"  
    }  
  ]  
}
```

GET details of an API

You can use the HTTP method GET to obtain information about a specific z/OS Connect EE API.

HTTP method

GET

URI

/zosConnect/apis/{APIname}

Description

Gets the details of the requested z/OS Connect EE API.

Security

Users with Admin, Operations or Reader authority can get details of an API, users with Invoke authority cannot. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Return body

```
{  
  "name": "{name}",  
  "version": "{version}",  
  "description": "{API description}",  
  "status": "Started|Stopped",  
  "apiUrl": "http(s)://{host}:{port}/{basePath}"  
  "documentation": {  
    "swagger": "http(s)://{host}:{port}/{basePath}/api-docs"  
  },  
  "services": [  
    {  
      "name": "{service name}",  
      "uri": "http(s)://{host}:{port}/zosConnect/services/{service name}"  
    }  
  ]  
}
```

where:

- apiUrl is the URL you use to invoke the API.
- swagger contains the URL where the Swagger document for the API can be obtained. For more details see [“Documenting your API using Swagger” on page 601](#)

Example body

```
{  
  "name": "HospitalAPI",  
  "version": "1.0.0",  
  "description": "API for Hospital app",  
  "status": "Started",  
  "apiUrl": "http://localhost:9080/hospital",  
  "documentation": {  
    "swagger": "http://localhost:9080/hospital/api-docs"  
  }
```

```
    },
    "services": [
        {
            "name": "patient",
            "uri": "http://localhost:9080/zosConnect/services/patient"
        }
    ]
}
```

Errors

The following error can occur:

404 Not found
The API was not found.

Deploying an API

You must deploy your API to make it available for use.

You can use the HTTP method POST to deploy a z/OS Connect EE API.

HTTP method

POST

URI

/zosConnect/apis

Description

Deploys an API into z/OS Connect EE. The server will store the API in a file using the API name as the filename and a file extension of .aar.

Security

Users with Admin or Operations authority can deploy an API, users with Invoke or Reader authority cannot. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Note: A user must be a member of a global group to be able to deploy an API.

Request body

The API archive file file. The *Content-Type* for the request is application/zip.

Response body

```
{
    "name": "{name}",
    "version": "{version}",
    "description": "{API description}",
    "status": "{Started|Stopped}",
    "apiUrl": "http(s)://{{host}}:{port}/{basePath}",
    "documentation": {
        "swagger": "http(s)://{{host}}:{port}/{basePath}/api-docs"
    }
}
```

Example response body

```
{
    "name": "testHealthApi2",
    "version": "1.0.0",
    "description": "Health API",
    "status": "Started",
    "apiUrl": "http://192.168.99.100:9080/testHealthApi2",
    "documentation": {
        "swagger": "http://192.168.99.100:9080/testHealthApi2/api-docs"
    }
}
```

Errors

The following errors can occur:

400 Bad request
Invalid or missing API archive file

```
409 Conflict
Package conflicts with the existing z/OS Connect configuration

415 Unsupported media type
Content-Type is not application/zip

500 Internal Server Error
Server issue, might require administrator intervention.
```

Setting the initial status of a new API

When you deploy a new API, the default initial status is set to Started. You can optionally set the initial status to Stopped by appending a query string to the HTTP POST method:

HTTP method

POST

URI

```
/zosConnect/apis?status=stopped
```

Description

Deploys a new API into z/OS Connect EE and sets the status to Stopped. When an API is stopped, requests invoking the API will fail. Administration requests to /zosConnect/apis/{*apiName*} will still function as normal and the API will appear in the list returned by a GET request to /zosConnect/apis. To start the API after it has been deployed, see “[Change the status of an API](#)” on [page 682](#)

Security

Users with Admin or Operations authority can set the initial status of an API, users with Invoke or Reader authority cannot. For more information about user authorization, see “[Overview of z/OS Connect EE security](#)” on page 331.

Request body

API archive file to be deployed.

Example request

To deploy an API and set the initial status to stopped:

```
/zosConnect/apis?status=stopped
```

Response body

```
{
  "name": "{name}",
  "version": "{version}",
  "description": "{API description}",
  "status": "{Started|Stopped}",
  "apiUrl": "http(s)://{{host}}:{portport
```

Errors

The following error can occur:

```
400 Bad request
Unknown status specified
```

Update an API

You can use the HTTP method PUT to update a z/OS Connect EE API.

Note:

By default, an API will have the status `started`. However, it can be set to a specific initial status by appending a query string to the URI with a status. For example:

```
/zosConnect/apis/{apiName}?status=stopped
```

HTTP method

PUT

URI

/zosConnect/apis/{apiName}

Description

Updates the named API in z/OS Connect EE using the new API archive file file.

Note:

1. The API needs to be stopped before updating.
2. The package being updated needs to be for the same API.

Security

Users with Admin or Operations authority can update an API, users with Invoke or Reader authority cannot. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Request body

The API archive file file. The content type for the request is application/zip.

Response body

This function returns the Location header pointing to protocol://server:port basePath

```
{  
  "name": "{name}",  
  "version": "{version}",  
  "description": "{API description}",  
  "status": "Started|Stopped",  
  "apiUrl": "http(s)://{host}:{port}/{basePath}"  
  "documentation": {  
    "swagger": "http(s)://{host}:{port}/{basePath}/api-docs"  
  }  
}
```

where:

- `apiUrl` is the URL you use to invoke the API.
- `swagger` contains the URL where the Swagger document for the API can be obtained. For more details see [“Documenting your API using Swagger” on page 601](#)

Example response body

```
{  
  "name": "HospitalAPI",  
  "version": "1.0.0",  
  "description": "API for Hospital app",  
  "status": "Started",  
  "apiUrl": "http://localhost:9080/hospital",  
  "documentation": {  
    "swagger": "http://localhost:9080/hospital/api-docs"  
  }  
}
```

Errors

The following errors can occur:

400 Bad request
Invalid or missing API archive file

409 Conflict
A z/OS Connect API must be stopped before it can be updated.

415 Unsupported Media Type

```
Content-Type is not application/zip:  
500 Internal Server Error  
Server issue, might require administrator intervention.  
503 Service Unavailable  
A z/OS Connect API must complete all outstanding requests before it can  
be updated.
```

Change the status of an API

A query string option can be used to set the status of an API. If the API archive file, is not usable, the operation fails.

Note: When an API is stopped, the API will not invoke interceptors for new requests, and returns the following message:

```
503 Service Unavailable
```

The error response body contains:

```
{ "errorMessage": "The API has been stopped" }
```

Note: The status of an API can be either started, or stopped. You can change the status with the HTTP PUT method.

HTTP method

PUT

URI

```
/zosConnect/apis/{apiName}?status=started|stopped
```

Description

Starts or stops the existing API. When an API is stopped, new requests fail but existing requests will run to completion. Administration requests to /zosConnect/apis/{apiName} still function normally and the API appears in the list that is returned by the /zosConnect/apis request.

Security

Users with Admin or Operations authority can change the status of an API, users with Invoke or Reader authority cannot. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Request body

To change the status of an API, the request body should have no content. If an API archive file is in the request body, the request is to update the API and set the initial status after the update. For more information, see [“Update an API” on page 680](#).

Example request

To change the status of the API called *apiName* to started:

```
/zosConnect/apis/apiName?status=started
```

Response body

```
{  
  "name": "{name}",  
  "version": "{version}",  
  "description": "{API description}",  
  "status": "Started|Stopped",  
  "apiUrl": "http(s)://{{host}}:{port}/{basePath}",  
  "documentation": {  
    "swagger": "http(s)://{{host}}:{port}/{basePath}/api-docs"  
  }  
}
```

Note: Responses from GET requests, such as GET /zosConnect/apis/{apiName}?status=started, include the status property. For example:

```
{  
  "name": "API name",  
  "version": "version",  
  "description": "API description",  
  "status": "Started|Stopped",  
  "apiUrl": "http(s)://{{host}}:{port}/{basePath}",  
  "documentation": {  
    "swagger": "http(s)://{{host}}:{port}/{basePath}/api-docs"  
  }  
}
```

```
"status": "Started|Stopped",
"apiUrl": "API base path",
"documentation": {
    "swagger": "API Swagger document location"
}
```

Errors

400 Bad request
Unknown status specified

Delete an API

You can use the HTTP method DELETE to delete a z/OS Connect EE API.

HTTP method

DELETE

URI

/zosConnect/apis/{*apiName*}

Description

Uninstalls the named API from z/OS Connect EE, and deletes the API archive file from the file system.
An API must be in stopped state to be deleted.

Security

Users with Admin or Operations authority can delete an API, users with Invoke or Reader authority cannot.

Return body

```
{ "name": "{API name}" }
```

Example body

```
{ "name": "HospitalAPI" }
```

Errors

The following errors can occur:

409 Conflict
A z/OS Connect API must be stopped before it can be deleted.

500 Internal Server Error
Server issue, might require administrator intervention.

503 Service Unavailable
A z/OS Connect API must complete all outstanding requests before it can be deleted.

Administering z/OS Connect EE policies

Use z/OS Connect EE policies to adjust how an API request is processed in z/OS Connect EE, based on the HTTP header values sent in by the client.

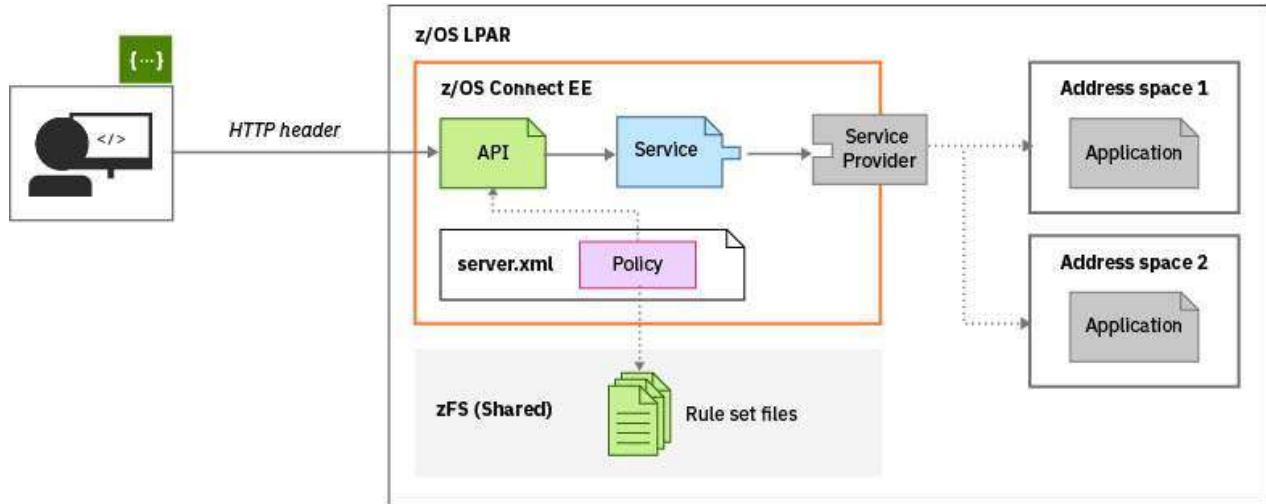
There are two types of policy, but only one policy is applied to an API.

1. *Specific* policies are applied only to selected API requests. You can define only one specific policy per API. Specific policies are defined in the `zosConnectAPI` element.
2. *Global* policies are applied to requests for all APIs with no specific policy defined. Global policies are defined in the `zosconnect_zosConnectAPIs` element.

Note:

1. The terms *specific* and *global* refer to the way the policy is used, not the way it is defined.
2. A specific policy can be applied only to requests for an API that is configured in `server.xml`.
3. If a specific policy fails to load, then no policy is applied to requests for that API.

You create rule sets to define the condition and actions, then enable z/OS Connect EE policies to apply those actions to API requests. A rule set contains one or more rules that define the condition and actions.



For a list of the properties that you can modify, see [“Valid properties for use in policy rules” on page 687](#).

Related concepts

[“Defining a rule set” on page 684](#)

Rule sets are XML files that describe actions to be performed when a condition is met. The condition is met when a property on the request header has a matching value.

[“How rule sets are applied” on page 689](#)

How z/OS Connect EE applies rule sets to requests.

Related tasks

[“Configuring z/OS Connect EE policies” on page 690](#)

How to configure z/OS Connect EE policies.

Defining a rule set

Rule sets are XML files that describe actions to be performed when a condition is met. The condition is met when a property on the request header has a matching value.

Create a rule set file by using the format and specifications that are described in [“Format of a rule set” on page 685](#).

Each rule consists of a `conditions` element that contains at least one header name-value pair. If the value of the property in the HTTP header matches any of the values that are defined in the condition, z/OS Connect EE performs the actions for that rule. If more than one header name-value pair is defined, then all conditions must be met for the action to be performed. If `match="ANY_VALUE"` is specified on the `header` element, z/OS Connect EE checks only that the named header is present on the request, regardless of the value.

You can also use variable substitution to modify the values according to the value given in the header property.

Note: Rules must not contradict each other. Contradictions cause inconsistent results.

For rule set examples, see [“Format of a rule set” on page 685](#).

Format of a rule set

A rule set must be written in a specific format.

A rule set consists of one or more rules. Each rule contains one or more conditions and one or more actions. If the conditions are met, z/OS Connect EE performs the related actions.

Rule specifications

A rule consists of a name attribute and the following sub elements:

conditions

Only one conditions element is allowed in each rule. The conditions element has one or more header elements with the attribute's name and value. Multiple header elements have a Boolean AND relationship, so all header conditions must be met for the actions to be performed. If more than one value is defined, then any one of those values can match.

actions

An action or list of actions to be performed if the condition is met. For example, the set action can be used to change the value of a property.

Note:

1. Leading and trailing white space for each comma-separated value is ignored.
2. If **match="ANY_VALUE"** is specified, the header condition is satisfied when an HTTP header with the correct name is included in the API request, regardless of the HTTP header value.
3. Rules are not validated by z/OS Connect EE. If the header name, property, or value is not entered correctly, results are unpredictable.

Example 1

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ruleset>
    <rule name="groupOne">
        <conditions>
            <header name="group" value="A,B,C"/>
        </conditions>
        <actions>
            <set property="imsConnectionRef" value="groupOneConnection"/>
            <set property="imsInteractionRef" value="groupOneInteraction"/>
        </actions>
    </rule>
    <rule name="groupTwo">
        <conditions>
            <header name="group" value="X,Y,Z"/>
        </conditions>
        <actions>
            <set property="imsConnectionRef" value="groupTwoConnection"/>
            <set property="imsInteractionRef" value="groupTwoInteraction"/>
        </actions>
    </rule>
    <rule name="setTrancode">
        <conditions>
            <header name="trancode" value="update"/>
        </conditions>
        <actions>
            <set property="imsTranCode" value="IVTNO"/>
        </actions>
    </rule>
</ruleset>
```

This sample rule set defines three rules. Multiple rules have a Boolean OR relationship. Each rule is checked separately. If for example, a request is received where the value of the HTTP header property group is B and trancode is update, then z/OS Connect EE performs the following actions:

1. <rule name="groupOne">. The condition is met and the value of imsConnectionRef is changed to groupOneConnection and imsInteractionRef groupOneInteraction.
2. <rule name="groupTwo">. The condition is not met, so the actions are ignored.

3. <rule name="setTrancode">. The condition is met, so the imsTrancode property is changed to IVTNO.

Variables

You can use variable substitution to target a specific transaction according to the header value in the request.

Example 2

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ruleset name="variableSubstitution">
    <rule name="connectionInteraction">
        <conditions>
            <header name="connectionType" value="A,B,C"/>
        </conditions>
        <actions>
            <set property="imsConnectionRef" value="${connectionType}Connection"/>
            <set property="imsInteractionRef" value="${connectionType}Interaction"/>
        </actions>
    </rule>
</ruleset>
```

This rule set has only one rule, but performs as if there are many rules. For example, if a request is received where the header property connectionType contains the value A, then the imsConnectionRef property is changed to AConnection and imsInteractionRef to AInteraction.

Variable substitution can be placed at any location in the value string, provided it is in the correct format. The substitution can also be used with no hardcoded value.

Example 3

Example 3 illustrates the use of multiple headers.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ruleset>
    <rule name="groupOne">
        <conditions>
            <header name="groupA" value="A"/>
        </conditions>
        <actions>
            <set property="imsConnectionRef" value="groupOneConnection"/>
            <set property="imsInteractionRef" value="groupOneInteraction"/>
        </actions>
    </rule>
    <rule name="groupTwo">
        <conditions>
            <header name="groupA" value="A"/>
            <header name="groupB" value="B"/>
        </conditions>
        <actions>
            <set property="imsConnectionRef" value="groupTwoConnection"/>
            <set property="imsInteractionRef" value="groupTwoInteraction"/>
        </actions>
    </rule>
</ruleset>
```

When multiple headers are defined in the same conditions element, they have a Boolean AND relationship so that all conditions must be met for the actions to be performed. In this example, the rule groupTwo has multiple conditions. The header **groupA** must match the value A and the header **groupB** must match the value B. If either condition is not met, then no actions occur.

In this example, the header named **groupA** appears in both rules, but the rule groupTwo has two headers, so it is considered to be more specific than rule groupOne and is checked first, even though it appears later in the ruleset than rule groupOne. If both conditions are met, then rule groupOne is not

checked. It is good practice to write your ruleset so that the most specific rules appear first. This format is easier to understand when diagnosing problems.

Example 4

Example 4 illustrates how a rule can set properties for multiple service providers and any properties not relevant to the service provider in use are ignored by the service provider.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ruleset>
    <rule name="groupOne">
        <conditions>
            <header name="groupA" value="A"/>
        </conditions>
        <actions>
            <set property="cicsConnectionRef" value="cicsOneconnection" />
            <set property="imsTranCode" value="IVTNO"/>
        </actions>
    </rule>
</ruleset>
```

Each service checks the actions that are applicable to it and ignores the others. In this example, the CICS service acts on the `cicsConnectionRef` property and the IMS service acts on the `imsTranCode` property.

Example 5

Example 5 illustrates how a rule can be configured to perform an action when a specific header is present on the API request, regardless of the value of that header. This method eliminates the need to include a long list of acceptable values in the rule set.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ruleset>
    <rule name="cicsTransId">
        <conditions>
            <header name="target" value="" match="ANY_VALUE"/>
        </conditions>
        <actions>
            <set property="cicsTransId" value="${target}" />
        </actions>
    </rule>
</ruleset>
```

If an API request header contains **target**, then the value of that header is substituted into `cicsTransId`.

Note: If you use the `match` attribute, you should always specify `value=""` as this attribute is required but is not used when the request header is checked for a matching name.

Valid properties for use in policy rules

An active policy rule can alter the value of certain properties if the condition defined in the rule is met.

The following properties can be altered by policy rules:

cicsCcsid

Specifies the CCSID that is used at run time to encode character data in COMMAREA and BIT container application data structures. This property is ignored for COMMAREA SARs that were generated by the build toolkit.

cicsConnectionRef

The ID of the `zosconnect_cicsIpicConnection` element that defines the connection to CICS.

cicsTransId

A CICS transaction name. This value overrides the value of `transid` that is set on the connection for this service.

db2CollectionID

Overrides the value of the collection ID that is set in the service project editor.

Note: A collection ID with a value that contains the # character is not supported.

db2ConnectionRef

The ID of the zosconnect_zosConnectServiceRestClientConnection element that defines the connection to Db2. This value overrides the connection reference that is specified in the service project editor during service archive file generation.

imsConnectionRef

The ID of the imsmobile_imsConnection element that defines the connection to IMS. This value overrides the connection profile that is specified in the API toolkit during service archive file generation.

imsInteractionRef

The ID of the imsmobile_interaction element that defines the connection to IMS. This value overrides the interaction profile that is specified in the API toolkit during service creation.

imsInteractionTimeout

Overrides the interactionTimeout attribute of the imsmobile_interaction element in the ims-interactions.xml configuration file. This is the value that is set in the IMS TM Resource Adapter method IMSInteractionSpec.setExecutionTimeout() at runtime. This policy property sets the TMRA IMSInteractionSpec property executionTimeout.

imsLtermOverrideName

Overrides the ltermOverrideName attribute of the imsmobile_interaction element in the ims-interactions.xml configuration file. This policy property sets the TMRA IMSInteractionSpec property ltermName.

imsTranCode

Overrides the transaction code that the service invokes at run time.

imsTranExpiration

Overrides the tranExpiration attribute of the imsmobile_interaction element in the ims-interactions.xml configuration file. Accepted values for the value attribute are “true” or “false”. This policy property sets the TMRA IMSInteractionSpec property transExpiration.

Note: The values specified for the imsTranExpiration, imsLtermOverrideName, and imsInteractionTimeout properties are set directly into the TMRA IMSInteractionSpec properties. To learn what these properties control, see the TMRA section of the *IMS documentation*.

mqConnectionFactory

Overrides the JNDI name of the connection factory that is used by the service to connect to IBM MQ.

mqDestination

- For a two-way service, this value overrides the JNDI name of the IBM MQ queue to which the service sends request messages.
- For a one-way service for receiving messages, this value overrides the JNDI name of the IBM MQ queue from which the service receives messages.
- For a one-way service for sending messages, this value overrides the JNDI name of the IBM MQ queue or topic, to which the service sends or publishes messages.

mqReplyDestination

Overrides the JNDI name of the IBM MQ queue from which a two-way service receives reply messages.

Note: This property does not apply to one-way services.

Related concepts

[“Format of a rule set” on page 685](#)

A rule set must be written in a specific format.

Validating a rule set

The following XML schema definition can be used to validate rule set files.

Use this schema with your organization's procedure to validate an XML file. This schema is included in the <install_path>/dev directory as file zosconnectruleset.xsd.

Rule set schema

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xsschema version="1.0" xmlns:xss="http://www.w3.org/2001/XMLSchema">

<xselement name="rule" type="rule"/>

<xselement name="ruleset" type="ruleSet"/>

<xsccomplexType name="ruleSet">
  <xsssequence>
    <xselement ref="rule" maxOccurs="unbounded"/>
  </xsssequence>
  <xseattribute name="name" type="xs:string"/>
</xsccomplexType>

<xsccomplexType name="rule">
  <xssall>
    <xselement name="conditions" type="conditions"/>
    <xselement name="actions" type="actions"/>
  </xssall>
  <xseattribute name="name" type="xs:string"/>
</xsccomplexType>

<xsccomplexType name="conditions">
  <xsssequence>
    <xselement name="header" type="condition" maxOccurs="unbounded"/>
  </xsssequence>
</xsccomplexType>

<xsccomplexType name="condition">
  <xsssequence>
    <xseattribute name="name" type="xs:string" use="required"/>
    <xseattribute name="value" type="xs:string" use="required"/>
    <xseattribute name="match" type="match"/>
  </xsssequence>
</xsccomplexType>

<xsccomplexType name="actions">
  <xsssequence>
    <xselement name="set" type="action" maxOccurs="unbounded"/>
  </xsssequence>
</xsccomplexType>

<xsccomplexType name="action">
  <xsssequence>
    <xseattribute name="property" type="xs:string" use="required"/>
    <xseattribute name="value" type="xs:string" use="required"/>
  </xsssequence>
</xsccomplexType>

<xssimpleType name="match">
  <xssrestriction base="xs:string">
    <xssenumeration value="ANY_VALUE"/>
  </xssrestriction>
</xssimpleType>
</xsschema>
```

How rule sets are applied

How z/OS Connect EE applies rule sets to requests.

When creating your rule sets, be aware of the following

- A rule can have multiple conditions and actions.
- An action can have multiple variable substitutions.

- A condition can have an action that affects multiple service provider types
- A variable substitution can be used independently.
- Only headers used in the conditions element of the rule can be used in variable substitution.
- Rules are not applied in the same order as they appear in a rule set. If you have conflicting rules, results will be inconsistent.
- If a property is changed by two rules in the same ruleset, message BAQR1204W is issued.
- When more than one condition is specified in a single rule, all conditions must be met for the actions to be performed
- When **match="ANY_VALUE"** is used in a condition, the value attribute is still required on that condition element (as **value=""**). The condition will be met when the named header is present on an API request, regardless of the headers value.
- Leading and trailing white space for each comma-separated value is ignored.
- Property names are case sensitive.
- The value of the **imsTranCode** property is not checked. You must validate the value in your application.
- If you update a valid ruleset with an invalid ruleset, the old rules are removed.

Configuring z/OS Connect EE policies

How to configure z/OS Connect EE policies.

Before you begin

Ensure that you have at least one rule set defined by completing the task [“Defining a rule set” on page 684](#).

About this task

Follow these steps to configure z/OS Connect EE to apply policies to incoming requests. You can configure multiple policies in the `server.xml` configuration file, but the maximum number that can be active is one policy per configured API plus one global policy. z/OS Connect EE attempts to load the rule set file for each policy and writes a confirmation message if the file is loaded successfully or an error message if problems were found. If you configure multiple policies, a message is written to the log for each rule set. These messages can be found in the file `messages.log`.

Procedure

1. Add a `zosconnect_policy` element in the `server.xml` configuration file.

This element defines the location and name of the rule set. You can declare any number of policy elements but they become active only when referenced by the `zosconnect_zosConnectAPIs` or `zosConnectAPI` elements. There can be only one nested `ruleset` element for each policy.

For example,

```
<zosconnect_policy id="production" location="/var/zosconnect/rules/">
  <ruleset file="rulefile1.xml"/>
</zosconnect_policy>
```

Note: The default location is `#{server.config.dir}/resources/zosconnect/rules`, but you can define another location in the `zosconnect_policy` element. The location must be a fully qualified path.

2. Choose whether this policy is to be used as a global or specific policy.

- a) To use the policy that was defined in step 1 as a global policy, define it in the `policyRef` attribute of the `zosconnect_zosConnectAPIs` element

```
<zosconnect_zosConnectAPIs policyRef="production"/>
```

The policy `production` is applied to all APIs.

- b) To define a specific policy, edit the `zosConnectAPI` element to add a **policyRef** attribute that references the `zosconnect_policy` element.

In this example, a second `zosconnect_policy` element is defined with an ID of `specialPolicy`.

```
<zosconnect_zosConnectAPIs policyRef="production">
  <zosConnectAPI name="mySpecialAPI" policyRef="specialPolicy"/>
</zosconnect_zosConnectAPIs>
```

In this example, two policies are configured, but `specialPolicy` is applied only to `mySpecialAPI`. The global policy `production` is applied to all other APIs.

Note: If a specific policy is defined for an API element, the global policy is not applied to that API even if the specific policy is not available.

What to do next

`z/OS Connect EE` is now configured to apply your policies to incoming API requests. You must now create your rule set file and store it in the location you defined. See [“Defining a rule set” on page 684](#).

How to enable `z/OS Connect EE` rule sets and policies

You must enable `z/OS Connect EE` policies to have `z/OS Connect EE` perform the actions that are defined in your rule sets.

You can manually enable your policies and rule sets either by restarting the server or by using the **Refresh Modify** command. The **Refresh Modify** command refreshes all server artifacts, including bind files, policies, .aar, .sar, and .ara files, and the `server.xml` configuration file. For more information, see [“MODIFY command syntax” on page 794](#) in the *Reference* section.

When the server starts, it enables any rule set files in the location defined in the **location** parameter of the `zosconnect_policy` element in the `server.xml` configuration file.

The following message indicates that a global policy is active.

BAQR7056I `z/OS Connect EE` policy `<policy name>` is active.

The following message indicates that an API-specific policy is active:

BAQR7082I: `z/OS Connect EE` policy `<policy name>` is active for API `<API name>`

You can also use MBeans to update your policies and rule sets on demand. For more information, see [“Using an MBean to trigger updates” on page 301](#).

How to apply rule set changes dynamically

Before you begin

Follow the instructions in [“Defining a rule set” on page 684](#) to create your rule set files.

About this task

Follow these steps to configure `z/OS Connect EE` to enable changes to your rule sets as soon as the file is saved to the defined location. With this configuration, you do not need to restart the server to enable the rule sets.

Procedure

1. Ensure that the rule set files are in the location that you defined in the **location** parameter of the `zosconnect_policy` element in the `server.xml` configuration file.

2. In the `zosconnect_policy` element, set the **pollingRate** to a suitable value for your installation. The default is 1 minute.
3. In the same element, set the **updateTrigger** value to polled. The default is disabled. For example,

```
<zosconnect_policy id="production" location="/var/zosconnect/rules/"  
    updateTrigger="polled" pollingRate="120s">  
    <ruleset file="rulefile1.xml"/>  
</zosconnect_policy>
```

With this configuration, the server checks for changes to rule sets every 2 minutes.

4. Restart the z/OS Connect EE server to enable this configuration.

This step is only necessary if the **updateTrigger** parameter of the `config` element is set to disabled.

5. From now on, when you make a change to a rule set and save the file to the defined location, z/OS Connect EE will automatically enable that rule set.

How to apply policy changes dynamically

Before you begin

Follow the instructions in [“Configuring z/OS Connect EE policies” on page 690](#) to configure z/OS Connect EE to use policies.

About this task

Policies are defined in the `server.xml` configuration file. You can configure z/OS Connect EE to dynamically apply changes to the policy configuration by following this procedure.

Note: This change will also automatically update other artifacts defined in the configuration file, such as API archive files, service archive files, and API requester archive files. For more information, see [“Configuration updates on demand” on page 300](#).

Procedure

1. In the `server.xml` configuration file, set the **updateTrigger** parameter of the `config` element to polled.
You can also set this parameter to mbean to enable updates on demand. For more information, see [“Using an MBean to trigger updates” on page 301](#).
2. Save the configuration file.
If the **updateTrigger** parameter was set to disabled, restart the server or issue the [**Refresh Modify**](#) command to activate this change.
3. From now on, changes to the `server.xml` configuration file, will be enabled automatically without restarting the server.

Chapter 24. How to manage API requesters

Use this information to learn how to manage your API requesters.

You must first deploy your API requester to make it available for use. API requesters can be deployed to a z/OS Connect EE server by copying the API requester archive to the API requester directory, or using the RESTful administration interface. The RESTful administration interface can also be used to get information on the API requesters that have been deployed and to stop an API requester from accepting requests.

How to deploy an API requester automatically

API requesters can be deployed or undeployed automatically when you add or remove them from the API requester directory.

You can configure z/OS Connect EE to deploy an API requester automatically when you copy your API requester archive (.ara file) to the location where your API requesters are stored. For example, resources/zosconnect/apiRequesters.

To deploy an API Requester, you add the .ara archive file into the API Requester location directory. Your user ID must have the following permissions:

- Read, write, and execute permission to the API Requester location directory.
- Read and execute permission to all parent directories of the API Requester location directory.

To update an existing API Requester, you replace the .ara archive file in the API Requester location directory. To delete an existing API Requester, you delete the .ara archive file from the API Requester location directory. When you perform these tasks, your user ID must have the following permissions:

- Read, write, and execute permission to the API Requester location directory.
- Read and execute permission to all parent directories of the API Requester location directory.
- Write access to the .ara archive file.

The z/OS Connect EE server's user ID must have read and write access to the .ara archive file. Use the UNIX System Services **chmod** command to assign the appropriate permissions to the directories and archive files.

When you remove the API requester from the directory, it is automatically undeployed.

The API requester location is monitored for changes to the API requester. You can control how and when z/OS Connect EE reacts to these changes by setting the *updateTrigger* and *pollingRate* attributes of the **zosconnect_apiRequesters** element in the configuration file *server.xml*. The default value for *pollingRate* is 5000 mSecs. For more information about the element, see “[zosconnect_apiRequesters](#)” on page 754.

Note: API requester definitions remain in the *server.xml* configuration file after you delete or uninstall an API requester. This situation causes a warning message to be written to the log until the API requester is reinstalled.

When you copy an .ara file to the *apiRequesters* directory, ensure that the API requester name is unique within the directory. For example, if *apiRequester1.ara* exists in the directory and contains an API requester named *testApiRequester*, the *apiRequester1.ara* file must be removed before you copy *apiRequester2.ara*, which also contains an API requester named *testApiRequester*. If both .ara files are present when z/OS Connect EE starts, one of the API requesters is deployed. When the other file is read, an error message is returned that the API requester is already deployed.

To update an API requester, replace the .ara file in the *apiRequesters* directory. The file must have the same name. If the *updateTrigger* and *pollingRate* attributes of the **zosconnect_apiRequesters** element are set, the API requester is installed automatically when the directory is polled.

How to use the RESTful administration interface to manage API requesters

You can use the RESTful administration interface to manage API requesters.

The RESTful administration interface for API requesters is available in paths under /zosConnect/apiRequesters. A Swagger document describing the RESTful administration interface for z/OS Connect EE API requesters is available on the URI /zosConnect/api-docs once the API requester feature is enabled.

Supported operations

The following table lists the methods and administrative tasks that are supported in the RESTful administration interface for API requesters.

Table 98. Methods and tasks supported in the administration interface	
Method	Administration tasks supported
GET	<ul style="list-style-type: none">• “How to get a list of API requesters” on page 694• “How to get details of an API requester” on page 695
POST	“How to deploy an API requester using the administration interface” on page 697
PUT	<ul style="list-style-type: none">• “How to change the status of an API requester” on page 696• “How to update an API requester” on page 698
DELETE	“How to delete an API requester” on page 699

Notes for all calls

- All successful calls return HTTP status code 200 unless otherwise specified.
- URLs returned by any of the calls contain the protocol, server, and port from the request.
- Any call using an HTTP method with a URI that is not mentioned in the topics in [Table 1](#) will return an error 405 Method Not Allowed.
- If an error occurs while a request is being processed, an appropriate HTTP status code and the following JSON will be returned:

```
{  
    "errorMessage": "{message}",  
    "errorDetails": "{details}"  
}
```

Note: errorDetails is optional and will only be returned for some error scenarios.

How to get a list of API requesters

You can use the HTTP method GET to obtain a list of the z/OS Connect EE API requesters installed in the runtime.

HTTP method

GET

URI

/zosConnect/apiRequesters

Description

Gets a list of the z/OS Connect EE API requesters installed in the runtime, including some basic information and a URL for more detailed information.

Security

Users with Admin, Operations or Reader authority can get a list of API requesters; users with Invoke authority cannot.

Note: A user must be a member of a global group to be able to see all the resources. For more information about user authorization, see [“Overview of z/OS Connect EE security” on page 331](#).

Response body

```
{ "apiRequesters": [ { "name": "{name}", "version": "{version}", "description": "{API requester description}", "status": "Started|Stopped", "apiRequesterUrl": "http(s)://{{host}}:{port}/zosConnect/apiRequesters/{name}" "connection": "{Endpoint Connection id}" }, ... repeats ] }
```

where **apiRequesterUrl** contains the URL where you can get more details for the specific API requester. For more information, see [“How to get details of an API requester” on page 695](#).

Example response body

```
{ "apiRequesters": [ { "name": "Book_Inventory", "version": "1.0.0", "description": "API requester for Book_Inventory app", "status": "Started", "apiRequesterUrl": "http://localhost:9080/zosConnect/apiRequesters/Book_Inventory" "connection": "BookConnref" } ] }
```

How to get details of an API requester

You can use the HTTP method GET to obtain information about a specific z/OS Connect EE API requester.

HTTP method

GET

URI

/zosConnect/apiRequesters/{apiRequesterName}

Description

Gets the details of the requested z/OS Connect EE API requester.

Security

Users with Admin, Operations or Reader authority can get details of an API requester; users with Invoke authority cannot.

Response body

```
{ "name": "{apiRequesterName}", "version": "{version}", "description": "{API requester description}", "status": "Started|Stopped", "apiRequesterUrl": "http(s)://{{host}}:{port}/zosConnect/apiRequesters/{name}" "connection": "{Endpoint Connection id}" }
```

Example response body

```
{  
  "name": "Book_Inventory",  
  "version": "1.0.0",  
  "description": "API requester for Book_Inventory app",  
  "status": "Started"  
  "apiRequesterUrl": "http://localhost:9080/zosConnect/apiRequesters/Book_Inventory"  
  "connection": "BookConnref"  
}
```

Errors

The following error can occur:

```
404 Not found  
The API requester is not registered.
```

How to change the status of an API requester

You can change the status of an API requester through a RESTful interface by using HTTP PUT method.

The status of an API requester can be set to either started or stopped.

Note: When an API requester is stopped, it can not be invoked from the backend z/OS application and the following message returns:

```
503 Service Unavailable
```

The error response body contains:

```
{ "errorMessage": "API requester {apiRequesterName} has been stopped" }
```

HTTP method

PUT

URI

```
/zosConnect/apiRequesters/{apiRequesterName}?status=started|stopped
```

Description

Starts or stops the existing API requester. When an API requester is stopped, new requests will fail but the existing requests will be allowed to complete. Administration requests to /zosConnect/apiRequesters/{apiRequesterName} will still function normally and the API requester will appear in the list returned by the /zosConnect/apiRequesters request.

Security

Users with the Admin or Operations authority can change the status of an API; users with the Invoke or Reader authority cannot.

Request body

To change the status of an API request, the request body should have no content. If an API requester archive file is in the request body, the request is to update the API requester and set the initial status after the update. For more information, see [“How to update an API requester” on page 698](#).

Example request

To change the status of the API requester called *apiRequesterName* to started:

```
/zosConnect/apiRequesters/apiRequesterName?status=started
```

Response body

```
{  
  "name": "{name}",  
  "version": "{version}",  
  "description": "{API requester description}",  
  "status": "Started|Stopped",  
  "adminUrl": "http(s)://{{host}}:{port}/{basePath}/name",  
  "connection": "{ConnectionRefID}"  
}
```

Note:

Responses from GET requests, such as GET /zosConnect/apiRequesters/{apiRequesterName}?status=started, include the status property. For example:

```
{  
    "name": "{name}",  
    "version": "{version}",  
    "description": "{API requester description}",  
    "status": "Started|Stopped",  
    "adminUrl": "http(s)://{{host}}:{port}/{basePath}/name",  
    "connection": "{ConnectionRefID}"  
}
```

Errors

400 Bad request
Unknown status specified

How to deploy an API requester using the administration interface

You can use the HTTP method POST to deploy a z/OS Connect EE API requester.

HTTP method

POST

URI

/zosConnect/apiRequesters

Description

Deploys an API requester into z/OS Connect EE.

Security

Users with the Admin or Operations authority can deploy an API requester; users with the Invoke or Reader authority cannot.

Request body

The API requester archive file. The *Content-Type* for the request is application/zip.

Response body

```
{  
    "name": "{apiRequesterName}",  
    "version": "{version}",  
    "description": "{API requester description}",  
    "status": "Started|Stopped",  
    "apiRequesterUrl": "http(s)://{{host}}:{port}/{basePath}",  
    "connection": "{Endpoint Connection id}"  
}
```

Example response body

```
{  
    "name": "bookmanagementAPIrequester_1.0.0",  
    "version": "1.0.0",  
    "description": "Multiple operations on book management",  
    "status": "Started",  
    "apiRequesterUrl": "http://localhost:9080/zosConnect/apiRequesters/  
bookmanagementAPIrequester_1.0.0",  
    "connection": "bookmanagementAPIrequesterConn"  
}
```

Errors

The following errors can occur:

400 Bad request
Invalid or missing API requester archive

409 Conflict
Archive conflicts with the existing z/OS Connect configuration

415 Unsupported media type
Content-Type is not application/zip

500 Internal Server Error
Server issue, might require administrator intervention.

Setting the initial status of a new API requester

When you deploy a new API requester, the default initial status is set to `started`. You can optionally set the initial status to `stopped` by appending a query string to the HTTP POST method:

HTTP method

POST

URI

```
/zosConnect/apiRequesters?status=stopped
```

Description

Deploys a new API requester into z/OS Connect EE and sets the status to `stopped`. When an API requester is stopped, requests invoking the API requester will fail. Administration requests to `/zosConnect/apiRequesters/{API requester Name}` will still function as normal and the API requester will appear in the list returned by a GET request to `/zosConnect/apiRequesters`. To start the API requester after it has been deployed, see [“How to change the status of an API requester” on page 696](#).

Security

Users with the Admin or Operations authority can set the initial status of an API requester; users with the Invoke or Reader authority cannot.

Request body

The API requester archive file to be deployed.

Example request

To deploy an API requester and set the initial status to `stopped`:

```
/zosConnect/apiRequesters?status=stopped
```

Response body

```
{  
  "name": "{apiRequesterName}",  
  "version": "{version}",  
  "description": "{API requester description}",  
  "status": "Stopped",  
  "apiRequesterUrl": "http(s)://{{host}}:{port}/{basePath}",  
  "connectionRef": "{Endpoint Connection id}"  
}
```

Errors

The following error can occur:

400 Bad request
Unknown status specified

How to update an API requester

You can use the HTTP method PUT to update a z/OS Connect EE API requester.

Note:

By default, an API requester has the status `started`. It can be set to a specific initial status by appending a query string to the URI with a status. For example:

```
/zosConnect/apiRequesters/{apiRequesterName}?status=stopped
```

HTTP method

PUT

URI`/zosConnect/apiRequesters/{apiRequesterName}`**Description**

Updates the named API requester in z/OS Connect EE using the new API requester archive file.

Note:

1. The API requester needs to be stopped before updating.
2. The API requester archive being updated needs to be for the same API requester.

Security

Users with the Admin or Operations authority can update an API requester; users with the Invoke or Reader authority cannot.

Request body

The API requester archive file. The *Content-Type* for the request is application/zip.

Response body

```
{  
    "name": "{apiRequesterName}",  
    "version": "{version}",  
    "description": "{API requester description}",  
    "status": "Stopped",  
    "apiRequesterUrl": "http(s)://{{host}}:{port}/{basePath}",  
    "connection": "{Endpoint Connection id}"  
}
```

Example response body

```
{  
    "name": "bookmanagementAPIrequester_1.0.0",  
    "version": "1.0.0",  
    "description": "Multiple operations on book management",  
    "status": "Started",  
    "apiRequesterUrl": "http://localhost:9080/zosConnect/apiRequesters/  
bookmanagementAPIrequester_1.0.0",  
    "connection": "bookmanagementAPIrequesterConn"  
}
```

Errors

The following errors can occur:

400 Bad request
Invalid or missing API requester archive.

409 Conflict
A z/OS Connect EE API requester must be stopped before it can be updated.

415 Unsupported Media Type
Content-Type is not application/zip.

500 Internal Server Error
Server issue, might require administrator intervention.

503 Service Unavailable
A z/OS Connect EE API requester must complete all outstanding requests before it can be updated.

How to delete an API requester

You can use the HTTP method DELETE to delete a z/OS Connect EE API requester.

HTTP method

DELETE

URI

/zosConnect/apiRequesters/{apiRequesterName}

Description

Uninstalls the named API requester from z/OS Connect EE, and deletes the API requester archive from the file system. An API requester must be in the stopped status to be deleted.

Security

Users with the Admin or Operations authority can delete an API requester; users with the Invoke or Reader authority cannot.

Response body

```
{  
  "name": "{apiRequesterName}"  
}
```

Example response body

```
{  
  "name": "bookmanagementAPIrequester"  
}
```

Errors

The following errors can occur:

409 Conflict

A z/OS Connect EE API requester must be stopped before it can be deleted.

500 Internal Server Error

Server issue, might require administrator intervention.

503 Service Unavailable

A z/OS Connect EE API requester must complete all outstanding requests before it can be deleted.

Chapter 25. Extending z/OS Connect EE

Creating a z/OS Connect EE service provider

z/OS Connect EE provides the `com.ibm.zosconnect.spi.Service` SPI that supports the creation of service providers that you can use to process requests arriving via z/OS Connect EE.

About this task

For more information about the `com.ibm.zosconnect.spi.Service` SPI, see [Interface Service class](#).

z/OS Connect EE service providers can be written and delivered by any component to plug into the framework. Both WOLA and IMS service providers are included with z/OS Connect EE.

Service providers from third party products can be deployed with z/OS Connect EE. These service providers must supply a properties file describing the product features and install the properties file into the *z/OS Connect Product feature directory*. For more information, see [“Setting up the product extensions directory” on page 197](#).

A service provider implemented for z/OS Connect EE is an OSGi service that connects and interacts with z/OS Connect EE through the OSGi framework.

Procedure

1. Create a service provider that implements the z/OS Connect EE `com.ibm.zosconnect.spi.Service` SPI.
2. Edit the service metatype configuration so that it specifies the following attribute definition:

```
<OCD id="user.service.ID" ibm:alias="userServiceAlias" name="serviceName"
description="zOSConnect User ServiceProvider" ibm:objectClass="com.ibm.zosconnect.serviceType">
```

where the `ibm:objectClass` attribute indicates that this object is a type service provider.

Java API documentation for the z/OS Connect EE SPIs can be found in the [z/OS Connect EE SPI reference](#), and as a separate file at `<installation_path>/doc/javadoc.zip`.

The Java API documentation for each Liberty profile SPI can be found in the *Liberty* product documentation and is also available in a separate .zip file in one of the javadoc directories of the `<installation_path>/wlp/dev` directory.

Creating a z/OS Connect EE service at run time

To process incoming requests, you can create a z/OS Connect EE service at run time by using the `com.ibm.zosconnect.spi.ServiceController` service programming interface (SPI).

About this task

You can dynamically create a z/OS Connect EE service provider at run time based on the configuration that is stored in an external repository.

Procedure

1. Create a service provider that implements the z/OS Connect EE `com.ibm.zosconnect.spi.ServiceController` SPI.
2. At run time, register the service with the OSGi framework that is using the `registerService` method on the `BundleContext` attribute.

```

Dictionary<String, Object> dynamicServiceProps = new Hashtable<String, Object>();
dynamicServiceProps.put(ServiceControllerConstants.SERVICE_NAME, "myNewService");
dynamicServiceProps.put(ServiceControllerConstants.INVOKE_URI, new String[] { "/u/my/url1",
    "/u/myurl2", "/u/my/url3*" });
ServiceRegistration<ServiceController> dynamicServiceReg =
    bundleContext.registerService(com.ibm.zosconnect.spi.ServiceController.class, new
MyServiceController(), dynamicServiceProps);

```

Note: The Java API documentation for each Liberty profile SPI can be found in the *Liberty* product documentation and is also available in a separate .zip file in one of the javadoc directories of the <installation_path>/wlp/dev directory.

Creating an interceptor

You can use interceptors for various purposes, such as monitoring, recording, or validation. Some interceptors are included with z/OS Connect EE, but you can also use interceptors that are supplied by other products or you can create your own.

Interceptors are OSGi services that are called by z/OS Connect EE through the OSGi framework. Design your interceptors to run quickly and maintain acceptable performance of the request flow.

z/OS Connect EE provides three interceptors:

- An *audit interceptor* is used to write SMF records. For more information, see “[Using SMF records to monitor requests](#)” on page 475.
- An *authorization interceptor* performs authorization checks to control users' access to resources. For more information, see “[API provider authorization](#)” on page 373.
- A *file system logger interceptor* logs request information to a file. For more information, see “[Configuring the file system logger interceptor](#)” on page 290.

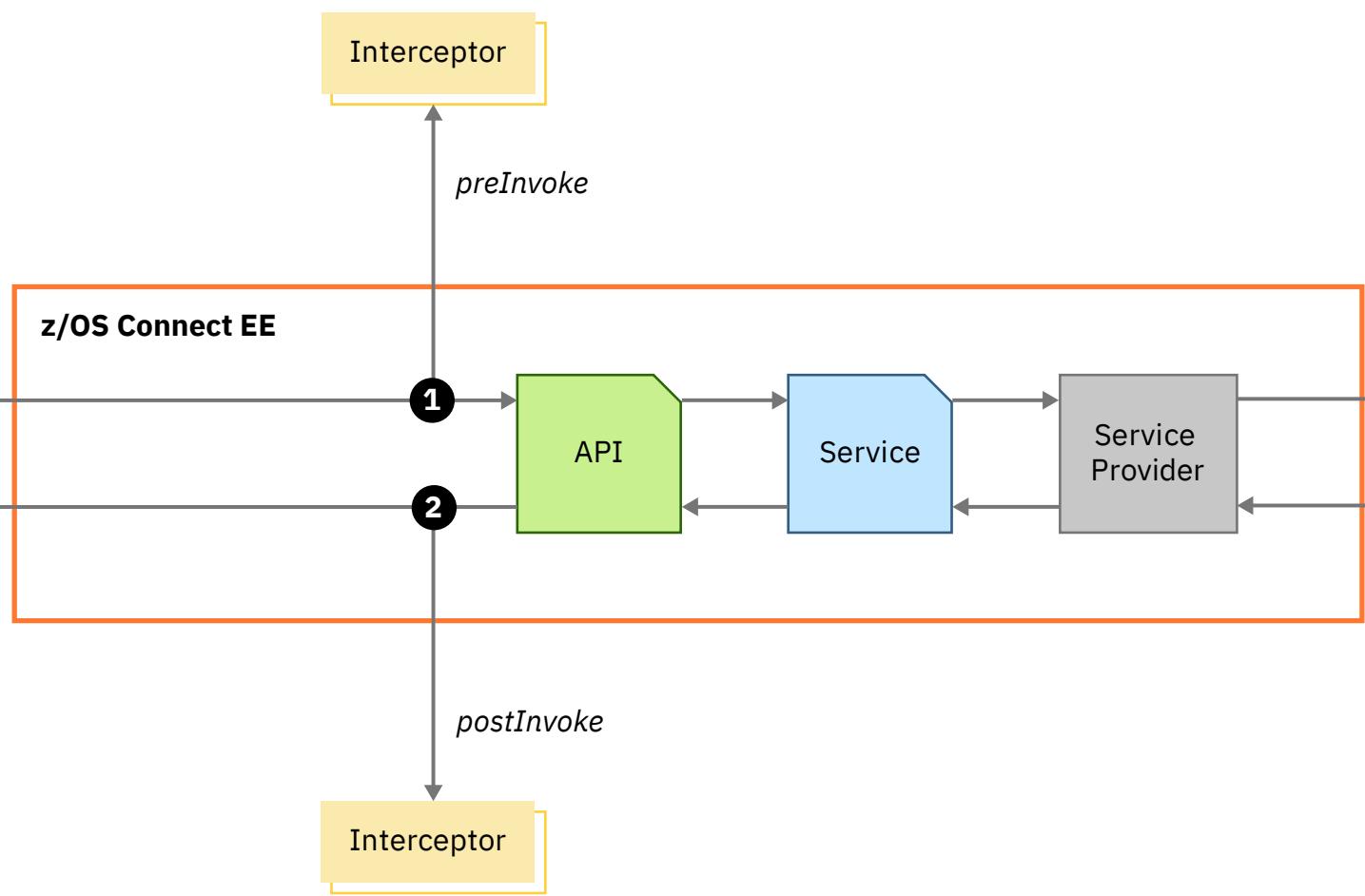
Intercepting API provider calls

Create an interceptor for monitoring, recording, or validation of API requests sent to a System of Record (SoR) and z/OS Connect EE administration requests, by implementing the `Interceptor` interface.

The `Interceptor` interface declares four methods: `getName`, `getSequence`, `preInvoke`, and `postInvoke`. For more information, see `com.ibm.zosconnect.spi.Interceptor`. The Java API documentation for the [z/OS Connect SPI](#) is also available as a separate file at <installation_path>/doc/javadoc.zip.

The `getName` method enables an interceptor to supply the name by which it is referenced in messages. The `getSequence` method enables an interceptor to return a configured sequence number that is used to specify when this interceptor is run relative to other configured interceptors. For more information, see “[Configuring interceptors](#)” on page 288.

The `preInvoke` and `postInvoke` methods are called for API, direct service, and all administration requests. They receive a copy of the request and information about the request. The type of request can be determined by calling the `Data.getRequestType()` method. An interceptor cannot alter a request in any way. If an interceptor's `preInvoke` method is called, its `postInvoke` method is always called. The points at which these methods are called is illustrated in [Figure 117 on page 703](#).



• preInvoke

Required interface: *Interceptor*

The `preInvoke` method is called when a request is received from the REST client and initial validation of the request is complete. This method can throw an interceptor exception to reject the request. If a request fails initial validation, the request is rejected before the `preInvoke` method is called. A request is considered invalid if, for example, it contains an invalid URL or fails user authentication. To capture these failing requests, see “[Creating a monitoring interceptor](#)” on page 706.

• postInvoke

Required interface: *Interceptor*

The `postInvoke` method is called just before a response is returned to the REST client.

The OSGI service metatype definition for the interceptor must set the object class definition (OCD) element attribute `ibm:objectClass` to `com.ibm.zosconnect.interceptorType` to indicate the object is an *Interceptor*. For example: definition:

```
<OCD id="user.interceptor.ID" ibm:alias="userInterceptorAlias" name="userInterceptorName"
description="zOSConnect User Interceptor" ibm:objectClass="com.ibm.zosconnect.interceptorType">
```

For an example of building and deploying an interceptor, see the `readme.md` file in the [zosconnect-sample-interceptor](#) repository of the GitHub project [zosconnect](#).

For information about interfaces that provide additional interceptor points, see “[Creating a monitoring interceptor](#)” on page 706.

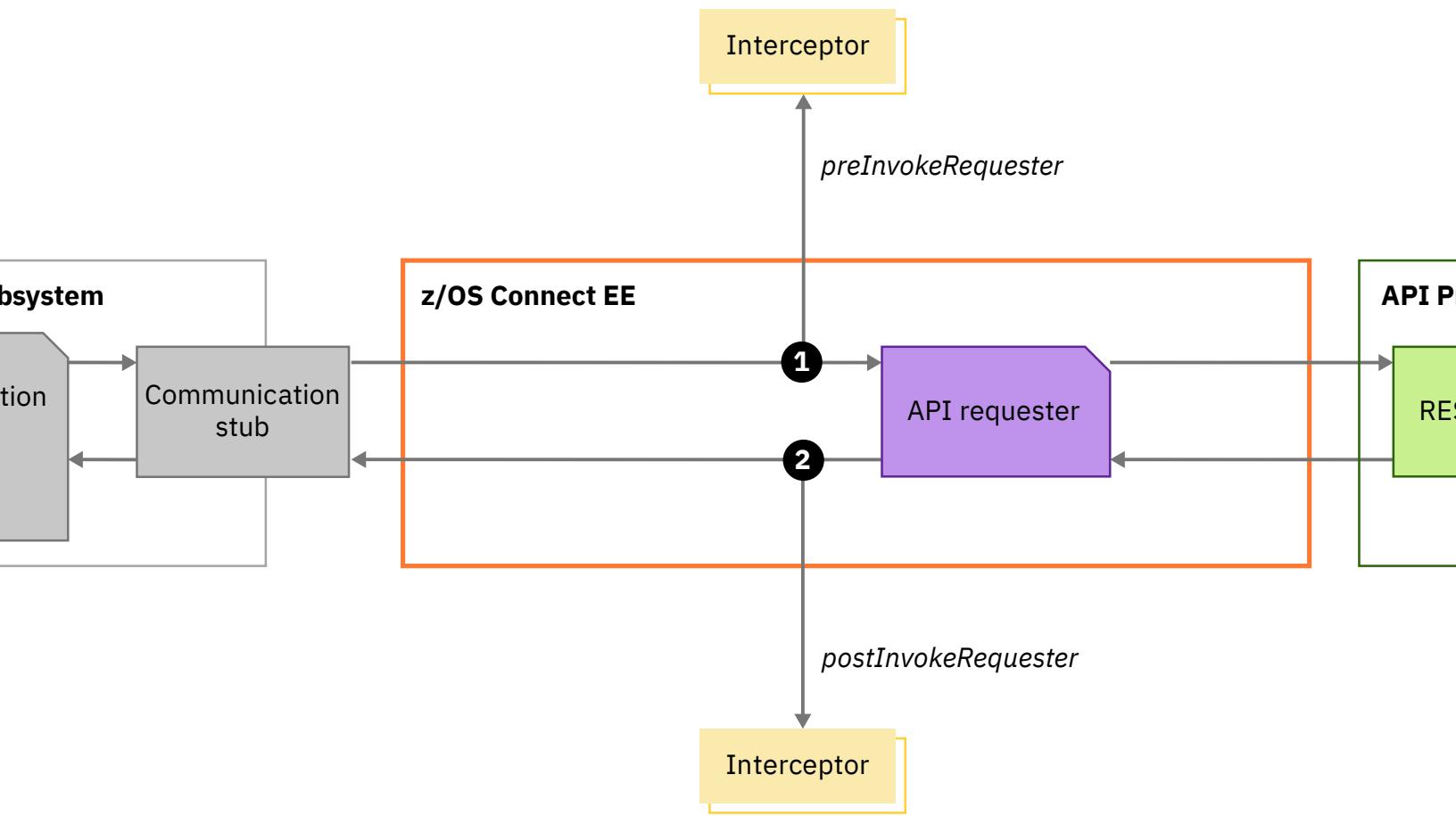
Intercepting API requester calls

Create an interceptor for monitoring, recording, or validation of API requests sent from a z/OS application to a REST endpoint, by implementing the *InterceptorRequester* interface.

The *InterceptorRequester* interface extends the *Interceptor* interface and is applicable only to API requester. API requester administrator requests are handled by the *Interceptor* interface, see “[Intercepting API provider calls](#)” on page 702.

The *InterceptorRequester* interface inherits the four methods from the *Interceptor* interface: `getName`, `getSequence`, `preInvoke`, and `postInvoke` and adds two more methods: `preInvokeRequester`, and `postInvokeRequester`. For more information, see [com.ibm.zosconnect.spi.InterceptorRequester](#). The Java API documentation for the [z/OS Connect SPI](#) is also available as a separate file at `<installation_path>/doc/javadoc.zip`.

The `preInvokeRequester` and `postInvokeRequester` methods are called for each request, and receive information about the request. An interceptor cannot alter a request in any way. If an interceptor's `preInvokeRequester` method is called, its `postInvokeRequester` method is always called. The points at which these methods are called is illustrated in [Figure 118](#) on page 705.



• preInvokeRequester

Required interface: *InterceptorRequester*

The `preInvokeRequester` method is called when a request is received from the z/OS application and initial validation of the request is complete. This method can throw an interceptor exception to reject the request. If a request fails initial validation, the request is rejected before the `preInvokeRequester` method is called. A request is considered invalid if, for example, it contains an invalid URL or fails user authentication. To capture these failing requests, see “[Creating a monitoring interceptor](#)” on page 706.

• postInvokeRequester

Required interface: *InterceptorRequester*

The `postInvokeRequester` method is called just before a response is returned to the z/OS application.

The OSGI service metatype definition for the interceptor must set the object class definition (OCD) element attribute `ibm:objectClass` to `com.ibm.zosconnect.interceptorType` to indicate the object is an Interceptor. For example:

```
<OCD id="user.interceptor.ID" ibm:alias="userInterceptorAlias" name="userInterceptorName"
description="zOSConnect User Interceptor" ibm:objectClass="com.ibm.zosconnect.interceptorType">
```

For an example of building and deploying an interceptor, see the `readme.md` file in the [zosconnect-sample-interceptor](#) repository of the GitHub project [zosconnect](#).

For information about interfaces that provide additional interceptor points, see “[Creating a monitoring interceptor](#)” on page 706.

Creating a monitoring interceptor

You create a monitoring interceptor by implementing the `Interceptor` interface and optionally, one or more of its subinterfaces.

Before you study this topic, you should first be familiar with the information in “[Creating an interceptor](#)” on page 702.

Implement the `com.ibm.zosconnect.spi.Interceptor` interface to monitor API requests that invoke a service in a System of Record (SoR) and all z/OS Connect EE administration requests.

The z/OS Connect EE SPI Interceptor subinterfaces are:

`com.ibm.zosconnect.spi.ServiceProviderInterceptor`

Implement this interface to obtain SoR specific monitoring data for API requests.

`com.ibm.zosconnect.spi.EarlyFailureInterceptor`

Implement this interface to monitor API requests that fail authentication or are badly formed.

`com.ibm.zosconnect.spi.InterceptorRequester`

Implement this interface to monitor API requests from a z/OS application that invoke a service in a REST endpoint.

`com.ibm.zosconnect.spi.EndpointInterceptor`

Implement this interface to obtain API endpoint-specific monitoring data for requests from a z/OS application.

`com.ibm.zosconnect.spi.EarlyFailureInterceptorRequester`

Implement this interface to monitor API requests from a z/OS application that fail authentication or are badly formed.

`com.ibm.zosconnect.spi.TrackingInterceptor`

Implement this interface to selectively track API requests that invoke a service in an SoR. It enables a stakeholder to add their stakeholder token to the tracking token sent to the SoR.

The Java API documentation for the `z/OS Connect SPI` is also available as a separate file at `<installation_path>/doc/javadoc.zip`.

z/OS Connect EE provides five points where interceptors can monitor requests for the API provider, as illustrated in [Figure 119 on page 707](#).

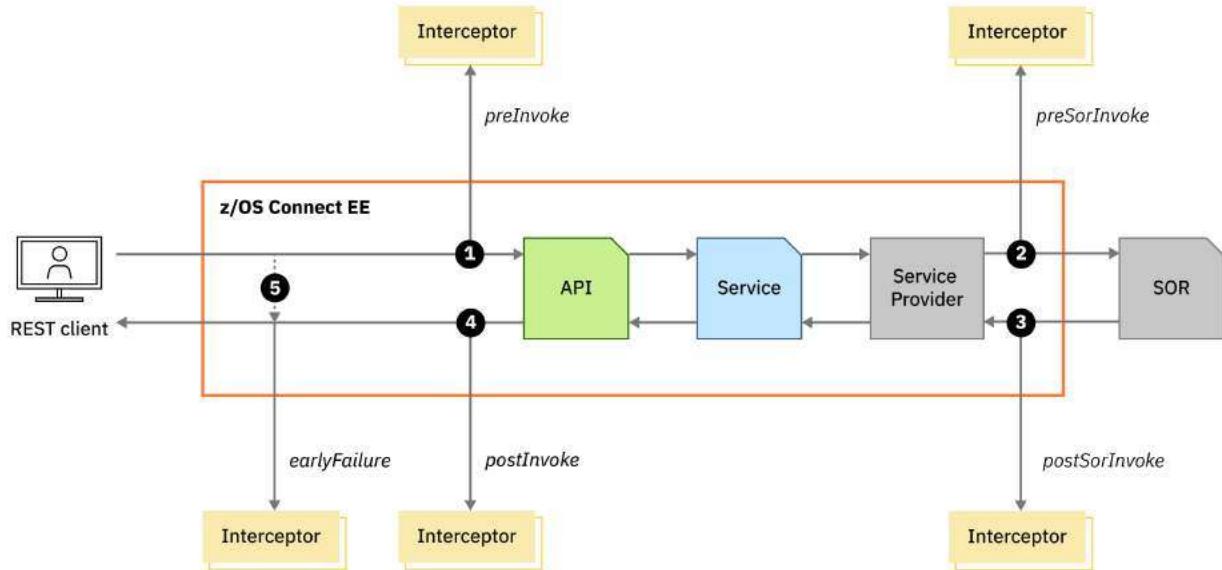


Figure 119. Monitoring interceptor points for the API provider

• **preInvoke**

Required interface: *Interceptor*

The *preInvoke* method is called when a request is received from the client and initial validation of the request is complete. This method can throw an interceptor exception to reject the request at this point.

• **preSoRInvoke**

Required interface: *ServiceProviderInterceptor*

The *preSoRInvoke* method is called when a request is sent to the SoR.

• **postSoRInvoke**

Required interface: *ServiceProviderInterceptor*

The *postSoRInvoke* method is called when a response is received from the SoR.

• **postInvoke**

Required interface: *Interceptor*

The *postInvoke* method is called just before a response is returned to the client.

• **earlyFailure**

Required interface: *EarlyFailureInterceptor*

The *earlyFailure* method is called when a request fails validation before the *preInvoke* method is called. No other interceptors are called. Early failures can be caused by errors such as incorrect URLs or authentication failures.

Note: The *earlyFailure* method is invoked only for interceptors configured at the global level.

z/OS Connect EE also provides five monitoring points for the API requester, as illustrated in [Figure 120 on page 708](#).

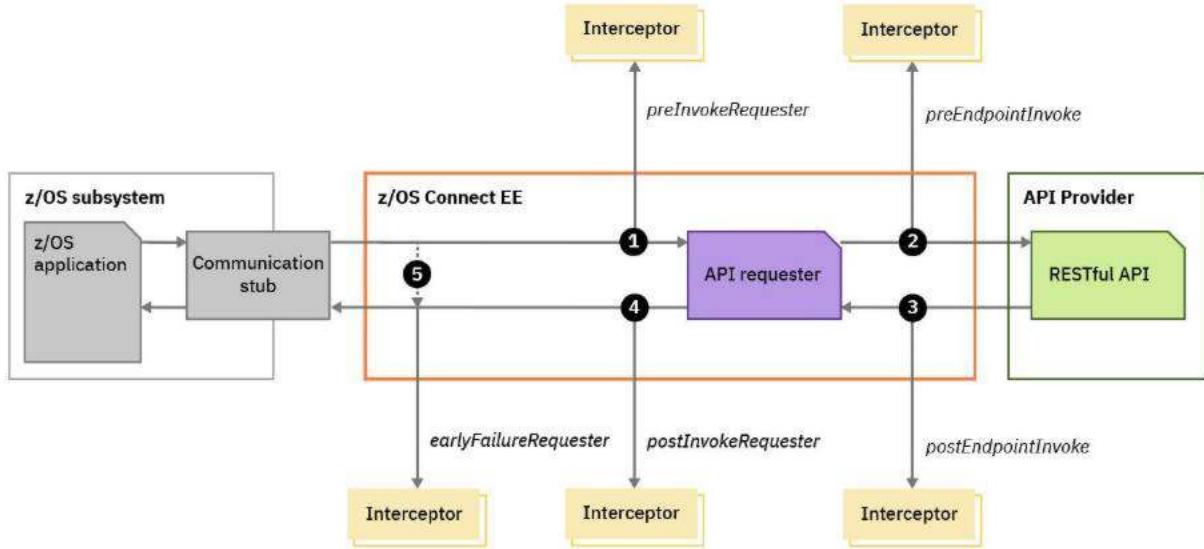


Figure 120. Monitoring interceptor points for the API requester

• **preInvokeRequester**

Required interface: *Interceptor*

The `preInvokeRequester` method is called when a request is received from the z/OS application and initial validation of the request is complete. This method can throw an interceptor exception to reject the request.

• **preEndpointInvoke**

Required interface: *EndpointInterceptor*

The `preEndpointInvoke` method is called when a request is sent to the API endpoint.

• **postEndpointInvoke**

Required interface: *EndpointInterceptor*

The `postEndpointInvoke` method is called when a response is received from the API endpoint.

• **postInvokeRequester**

Required interface: *Interceptor*

The `postInvokeRequester` method is called just before a response is returned to the z/OS application.

• **earlyFailureRequester**

Required interface: *EarlyFailureInterceptor*

The `earlyFailureRequester` method is called when a request fails validation before the `preInvokeRequester` method is called. No other interceptors are called. Early failures can be caused by errors such as incorrect URLs or authentication failures.

Note: The `earlyFailureRequester` method is invoked only for interceptors configured at the global level.

Related concepts

[“Tracking selected requests” on page 710](#)
Monitoring selected requests

Interpreting time data

API request time data is provided for both API provider and API requester to enable problem determination.

API provider timeouts

When running z/OS Connect EE in asynchronous mode and an API or service request times out, the client receives a 503 HTTP status response but request processing can continue asynchronously. Interceptors are called for the asynchronous request processing. When the `asyncRequestTimeout` expires for a request, the `Data.REQUEST_TIMED_OUT` flag is set and the `Data.HTTP_RESPONSE_CODE` is set to the result of the asynchronous request processing, which might have been successful. The CICS service provider includes support for the `asyncRequestTimeout`, so when the `asyncRequestTimeout` expires for a request, asynchronous processing of the request ends and a request is sent to CICS to purge the transaction, if appropriate.

API requester timestamp data

Table 99 on page 709 shows how the timestamps are set that enable the calculation of the time spent in z/OS Connect EE, the time to obtain one or more access tokens for the request and the time for the API endpoint to process the request. For requests that do not use access tokens, `DataRequester.TIME_TOKEN_GET_START` and `DataRequester.TIME_TOKEN_GET_FINISH` are not set.

If the request response is a 401 authentication failure, the request is retried once with one or more new tokens. The time between `DataRequester.TIME_ENDPOINT_SENT` and `DataRequester.TIME_ENDPOINT_RECEIVED` then includes the time to make both the first and second request to the endpoint and the time to get the new token(s) from the authorization server.

Table 99. Setting of timestamps in API requester flow

Step	Timestamp setting and request flow
1	Request sent from communications stub to z/OS Connect EE. <ul style="list-style-type: none">• <code>DataRequester.TIME_STUB_SENT</code> set.
2	Request enters z/OS Connect EE server. <ul style="list-style-type: none">• <code>DataRequester.TIME_ZOS_CONNECT_ENTRY</code> set.
3	Request authentication, authorization and data transformation.
4 - if access token(s) required	If access token(s) required for request, start acquisition. <ul style="list-style-type: none">• <code>DataRequester.TIME_TOKEN_GET_START</code> set.
5 - if access token(s) required	Access token(s) obtained from cache or authorization server. <ul style="list-style-type: none">• <code>DataRequester.TIME_TOKEN_GET_FINISH</code> set.
6	Request sent to endpoint. <ul style="list-style-type: none">• <code>DataRequester.TIME_ENDPOINT_SENT</code> set.
7	Response received from endpoint. <ul style="list-style-type: none">• <code>DataRequester.TIME_ENDPOINT_RECEIVED</code> set.
8 - if 403 failure	If request received an authentication failure response, new access token(s) obtained from authorization server.

Table 99. Setting of timestamps in API requester flow (continued)

Step	Timestamp setting and request flow
9 - if 403 failure	Request sent to endpoint. <ul style="list-style-type: none">• DataRequester.TIME_ENDPOINT_RECEIVED cleared.
10 - if 403 failure	Response received from endpoint. <ul style="list-style-type: none">• DataRequester.TIME_ENDPOINT_RECEIVED set.
11	Response data transformation.
12	Response returned from z/OS Connect EE server. <ul style="list-style-type: none">• DataRequester.TIME_ZOS_CONNECT_EXIT set.
1	Response received in communications stub.

Tracking selected requests

You can monitor selected requests in the System of Record (SoR) by creating an interceptor that implements the `TrackingInterceptor` interface. This interface is only applicable to API provider requests.

Before you study this topic, you should be familiar with the information in [“Intercepting API provider calls”](#) on page 702 and [“Creating a monitoring interceptor”](#) on page 706.

z/OS Connect EE generates a system-wide unique correlator for each API and service request and sends it to the target system of record, embedded in a tracking token. Tracking tokens are currently supported by the CICS, IMS and REST client service providers.

- For CICS, up to z/OS Connect EE V3.0.29.0, the tracking token is available in the user correlation data field `USERCORRDATA` of the task association data.

From V3.0.30.0, the tracking token is available in the adapter data field `ODADPTRDATA1` of the task association data, and the adapter field `ODADPTRID` contains the z/OS Connect EE product identifier in the format `IBM_zOS_Connect_CICS_SP <product_version><product_buildlevel>`.

CICS makes this data available to applications through the **INQUIRE ASSOCIATION** system command. CICS also records this information in the `OADID` and `OADATA1` fields of the DFHCICS group in the PERFORMANCE class of SMF 110 SubType 1 records.

- For IMS, the tracking token is available in the user data fields in the OTMA header. The `OMUSR_TRCKID_OFF` field in the OTMA user data provides the offset to the tracking token. The tracking token is then recorded by IMS in the X'01' type log record under the MVS APPC message prefix segment.
- For REST endpoints, the first 4 bytes of the tracking token are converted to ASCII and the remaining bytes are Base64 encoded. The result is sent in the `X-Correlation-ID` HTTP header on each request.
- For Db2, the first 4 bytes of the tracking token are converted to ASCII and the remaining bytes are Base64 encoded. The result is sent in the `X-Correlation-ID` HTTP header on each request.

Db2 takes the value in the `X-Correlation-ID` header and assigns it to the Db2 CURRENT CLIENT_CORR_TOKEN special register for access by application processes.

Db2 also records the value of `X-Correlation-ID` as the *client correlation token* in the accounting correlation header record of Db2 trace. The full length correlation token value is reported in the `QWHCCTKN_D` field of the trace record which is located using fields `QWHCCTKN_OFF`, `QWHCCTKN_LEN`, and `QWHCCTKN_VAR`. These trace records can be written to SMF 101 records or GTF.

For information about how to access the information in those fields, see the applicable system of record documentation. The format of tracking tokens is shown in [Table 101](#) on page 711 for CICS and IMS, and in [Table 102](#) on page 712 for REST endpoints.

Tokens

z/OS Connect EE provides a mechanism by which monitoring products, also referred to as *stakeholders*, can specify selective tracking of requests in the processing system of record. For each API or service request, z/OS Connect EE generates a unique correlator to identify the request. This correlator is embedded in a *tracking token* together with *stakeholder tokens* that are supplied by active tracking interceptors. The tracking token is sent with the request to the system of record, where the token can be programmatically inspected by the monitoring product's agent. The stakeholder tokens enable different monitoring products to supply their own indication of whether or not a request should be tracked in the system of record.

A stakeholder token contains a monitoring product's stakeholder identifier, which is a single character in the range A-Z or a-z. The format of stakeholder tokens is shown in [Table 100 on page 711](#). Monitoring products are encouraged to register their chosen identifier with IBM to ensure it is unique.

Use the [z/OS Connect SPI](#) to implement selective tracking:

1. Create an interceptor that implements the `TrackingInterceptor` interface and return your stakeholder identifier on the `TrackingInterceptor.getStakeholderIdentifier()` method, which is called when the interceptor is activated in the z/OS Connect EE server. This enables z/OS Connect EE to warn if two monitoring products are incorrectly using the same stakeholder identifier.
2. During the invocation of the `Interceptor.preInvoke` method, call the `Data.addStakeholderToken(char stakeholderIdentifier)` method to add a stakeholder token containing your stakeholder identifier to the tracking token associated with the request.
3. Deploy your monitoring agent in the system of record to inspect the stakeholder tokens and take the appropriate monitoring action.

Format of tracking tokens

Table 100. Format of a stakeholder token

Bytes	Field	Type	Description
0	Stakeholder identifier	Character	An EBCDIC character in the range A-Z or a-z, registered as the stakeholder's identifier.
1-n	Reserved	-	Reserved for future use.

Table 101. Format of a tracking token in CICS and IMS.

Bytes	Field	Type	Description
0-2	Product Identifier	Character	z/OS product identifier, 'BAQ' in EBCDIC.
3	Version	Integer	Version of tracking token.
4-5	Correlator length	Integer	Length of the correlator, in big-endian format.
6-n	Correlator	Byte array	System-wide unique correlator.
n+1	Stakeholder token count	Integer	Number of stakeholder tokens.
n+2	Stakeholder token length	Integer	Length of stakeholder tokens. Present only if the number of stakeholder tokens is not zero. (Use this length to traverse the list of stakeholder tokens to allow for a future increase in the length of stakeholder tokens.)

Table 101. Format of a tracking token in CICS and IMS. (continued)

Bytes	Field	Type	Description
n+3	Stakeholder token list	Stakeholder token array	List of stakeholder tokens. Present only if the number of stakeholder tokens is not zero.

Table 102. Format of a tracking token passed to REST endpoints, including Db2.

Characters	Field	Type	Description
0-2	Product Identifier	Character	z/OS product identifier, 'BAQ' in ASCII.
3	Version	Character	Version of tracking token in ASCII.
4-n	Encoded tracking information	Character	Base64 encoded bytes 4-(n+3) of tracking token in Table 101 on page 711 .

Registered stakeholder identifiers

The following products have registered stakeholder identifiers.

Table 103. Registered stakeholder identifiers

Stakeholder identifier	Product
S	CA SYSVIEW
Z	IBM Z APM Connect

To register your product for a stakeholder identifier, contact an IBM representative or submit a request in the RFE community.

Related information

Setting communications stub data

The DataRequester interface contains API requester data that is sent to the z/OS Connect EE server from the communications stub. Most data is automatically set but some items require specific action for the values to be set.

Setting the CICS_UOWID and CICS_NETUOWID values

Values are set for the CICS_UOWID and CICS_NETUOWID data items in the com.ibm.zosconnect.spi.DataRequester interface for requests from CICS applications if the BAQCTRU TRUE is configured in the CICS region. For more information, see [“Configuring enhanced monitoring for CICS tasks” on page 307](#).

Setting the IMS_EXIT_DATA value

Values can be set for the IMS_EXIT_DATA data item in the com.ibm.zosconnect.spi.DataRequester interface for requests from IMS applications by a monitoring product or application.

The IMS communications stub makes INQY calls to request information about the execution environment. These calls can be intercepted by the IMS Monitor user exit (IMSMON). The I/O area parameter on the INQY call specifies the data output area for the call and when the communications stub allocates data for this I/O area, it allocates extra storage before the I/O area as shown in [Table 104 on page 713](#). On the INQY call, before the API request is sent to the z/OS Connect EE server, the eye catcher is set to

ZOSCONNECT-V3->>. A monitoring product can set a correlator or other value in this data area before the I/O area.

Table 104. Format of data area on INQY call before the API request

Bytes	Field	Description
0-15	Eyecatcher	Set with the EBCDIC character string ZOSCONNECT-V3->> before an API request and ZOSCONNECT-V3-<< after an API request.
16-79	Exit data	Field that can be set by a monitoring product. The value is transmitted to the z/OS Connect EE and is set as the value of IMS_EXIT_DATA.
80+	I/O area	The I/O area that is passed on the call to INQY.

After the API request, the communications stub makes a second call to INQY with the eye catcher ZOSCONNECT-V3-<< set.

Creating a z/OS Connect EE data transformer

You can use the z/OS Connect EE SPI to create a custom data transformer. Data transformers are OSGi services that implement the com.ibm.zosconnect.spi.DataXform SPI that is provided by z/OS Connect EE.

About this task

z/OS Connect EE data transformers are written and delivered by any component to plug into the framework. A data transformer is included with z/OS Connect EE and provides both to and from JSON byte arrays that are consumable by COBOL, PL/I, and C programs on z/OS.

A data transformer that is implemented for z/OS Connect EE is an OSGi service that connects and interacts with z/OS Connect EE through the OSGi framework.

Procedure

1. Implement the z/OS Connect EE com.ibm.zosconnect.spi.DataXform SPI in the service.
2. Edit the metatype configuration so that it specifies the following attributes. This definition should be entered on one line:

```
<OCD id="user.DataTransform.ID" ibm:alias="userDataTransformAlias" name="userDataTransformName"
description="zOSConnect User DataTransform" ibm:objectClass="com.ibm.zosconnect.dataXformType">
```

where the ibm:objectClass attribute indicates that this object is a type dataTransform.

Java API documentation for the z/OS Connect EE SPIs can be found in the [z/OS Connect EE SPI reference](#), and as a separate file at <installation_path>/doc/javadoc.zip.

The Java API documentation for each Liberty profile SPI can be found in the *Liberty* product documentation and is also available in a separate .zip file in one of the javadoc directories of the <installation_path>/wlp/dev directory.

As an implementation of a z/OS Connect Enterprise Edition programming interface, Liberty profile API and SPI features can be used.

Using the DataxFormExt interface

You can use the methods provided by the `com.ibm.zosconnect.spi.DataXformExt` SPI to obtain information about the encoding of the data transformer.

When you create a data transformer, you implement the z/OS Connect EE `com.ibm.zosconnect.spi.DataXform` SPI in the service. You can also use the methods in the `com.ibm.zosconnect.spi.DataXformExt` SPI to return information about the encoding.

For more information on using the DataxFormExt interface, see [Interface DataxFormExt](#).

The `getEncoding()` method

When you invoke the service configured to use a `DataXform` that implements the `getEncoding()` method of the `DataXform` interface, the service will check the value of the `Service.IBM_ZOS_CONNECT_ENCODING` key in the `requestStateMap`.

Chapter 26. Resolving problems

Use this section to find solutions to problems that might occur.

You can ask questions, and find answers to common z/OS Connect EE problems, on the IBM Z Community website : <https://community.ibm.com/community/user/ibmz-and-linuxone/groups/topic-home?CommunityKey=53f7f9ef-d807-4db9-99ec-d3a45a7fc9f4>.

You can find which version of z/OS Connect EE you are running in message BAQR0000I in the <WLP_USER_DIR>/servers/{servername}/logs/messages.log file.

Enabling trace in z/OS Connect EE server

Learn how to enable the z/OS Connect EE server trace when you are asked to do so by IBM service.

The following trace component specifications are available:

- z/OS Connect: **zosConnect=all**
- z/OS Connect REST client:
zosConnectServiceRestClient=all:com.ibm.ws.jaxrs20.client=all
- z/OS Connect FS Logger interceptor: **zosConnectInterceptorLoggerFS=all**
- z/OS Connect automatic deployment file monitor: **zosConnectFileMonitor=all**
- z/OS Connect audit interceptor: **zosConnectAuditSMF=all**
- z/OS Connect SAF services and SAF SURROGAT profile checks: **zosConnectSaf=all**
- z/OS Connect policy: **zosConnectPolicy=all**
- CICS service provider: **zosConnectServiceCics=all**
- WOLA service provider: **zosConnectLocalAdapters=all**
- WOLA: **zosLocalAdapters=all**
- IMS service provider:
 - **com.ibm.ims.*=all**
 - **com.ibm.j2ca.RAIMSTM=all** (for IMS connection issues)
- IMS database service provider:
 - **zosConnectServiceDb=all**
 - **com.ibm.ims.opendb=all** (for IMS connection issues)
- IBM MQ service provider: **zosConnectServiceMq=all**
- API requester: **zosConnectApiRequester=all**
- Data transformation for CICS, IMS, and IBM MQ service providers:
com.ibm.zosconnect.wv*=FINEST
- Endpoint connection: **zosConnectEndpointConnection=all**
- OAuth 2.0 and JWT for API requester: **zosConnectApiRequesterToken=all**

You can enable trace with any of the following methods:

1. Configure the server.xml logging element. Separate multiple trace specifications with colons (":").
For example:

```
<logging traceSpecification="zosConnect=all:zosConnectLocalAdapters=all"/>
```

or

```
<logging traceSpecification="zosConnect=all:com.ibm.ims.*=all:com.ibm.j2ca.RAIMSTM=all"/>
```

2. Create a file called `bootstrap.properties` in the `{server.config.dir}` directory, with content that specifies the `com.ibm.ws.logging.trace.specification` property. For example:

```
com.ibm.ws.logging.trace.specification==info:zosConnect=all:zosConnectLocalAdapters=all
```

For this method, you must restart the server for the changes to take effect.

3. Use a Liberty modify command: **MODIFY**

[jobname.]identifier,LOGGING='<trace_specification>'. For example,

```
F  
[jobname.]identifier,LOGGING='*=info:zosConnectFileMonitor=all:com.ibm.zosconnect.internal.Sar  
ManagerImpl=all'
```

If you use CORS, include the Liberty trace specification:

`*=info:CorsService=all:GenericBNF=all`.

For more information, see [Modify commands on z/OS](#) in the *Websphere Application Server for z/OS Liberty* documentation.

In each case, the trace is written to a `trace.log` file in the `${server.config.dir}/logs` directory by default.

For more information about setting up trace in a Liberty server, see the technote [Setting up trace and getting a full dump in the WebSphere Application Server V8.5 Liberty profile](#), or the topic [Liberty: Logging and Trace](#) in the *WebSphere Application Server Liberty* documentation.

Enabling trace in communications stubs

You can enable application user trace in the communication stubs for application development and problem determination.

Enabling trace in the communication stub for CICS

The communication stub for CICS uses the trace facilities of the CICS region in which it runs. The trace is implemented as CICS application user trace. To turn on CICS trace, use the supplied CETR transaction. For more information, see [CETR - trace control](#).

To trace the communication stub for CICS, set the *master user trace* flag to on and ensure the Application Programming(AP) component standard trace is enabled. In addition to AP, it is good practice to also enable the standard trace for the Socket(SO) and Web(WB) components. After trace is generated, use the CICS trace utility program to format the output.

The communication stub for CICS uses trace point ID: **AP 00A1**.

The following example for formatting CICS auxiliary trace uses the trace utility program supplied by CICS Transaction Server for z/OS version 5.6. For more information, see [CICS Trace utility program DFHTU730](#).

```
//PRTRACE JOB MSGCLASS=H,REGION=500M
//PRINT EXEC PGM=DFHTU730
//STEPLIB DD DSN=CTS560.CICS730.SDFHLOAD,DISP=SHR
// DD DSN=CTS560.CICS730.SDFHLINK,DISP=SHR
//DFHAUXT DD DSN=<aux_trace_hlq>.DFHAUXT,DISP=SHR
//DFHAXPRT DD SYSOUT=A
//DFHAXPRM DD *
FULL,TYPETR=(AP00A1,WB0000-FFFF,S00000-FFFF)
/*
```

The formatted trace entries generated by the communication stub for CICS are identified by the text:

```
AP 00A1 USER EVENT APPLICATION-PROGRAM-ENTRY BAQCSTUB
```

Enabling trace in the communication stub for IMS and z/OS

The communication stub for IMS and z/OS provides different levels for the verbose diagnostic logging capability. You enable and configure the logging capability with the environment variable **BAQVERBOSE** followed by the required level.

Trace messages are directed to the standard output for the application environment. By default, verbose messages are turned off.



CAUTION: If you use the settings ON or ALL, the trace output might contain sensitive information.

Available levels and their usage are:

OFF

Turns off verbose trace messages (the default). No trace messages are produced.

ON

Turns on verbose trace messages. Most messages are written, including messages for entry, exit, errors, and auditing. To investigate any issues in the API requester stub code for IMS and batch, enable this level of trace.

ERROR

If an error occurs within the stub code, whether an internal error or by bad data being passed, messages are written to the output destination.

AUDIT

Records the entry and exit to stub and third-party libraries such as the HWT. This level is useful to track the timing of events such as entry and exit to the stub code, and timing of request and response through the HWT.

ALL

All diagnostic messages are written, including previous levels and intensive trace. This can include string conversions, setters, lookup tables etc. This level may cause a performance impact so it is not advisable unless there is an issue in one of the areas that writes at this level.

For example, **BAQVERBOSE=ON**.

Contacting IBM Software Support

IBM Software Support provides assistance with product defects.

Before contacting IBM Software Support, your company must have an active IBM software maintenance contract, and you must be authorized to submit problems to IBM.

Follow the steps in this topic to contact IBM Software Support:

1. Determine the business impact of your problem.
2. Describe your problem and gather background information.
3. Submit your problem to IBM Software Support using [the IBM Support Portal](#).

Determine the business impact of your problem

When you report a problem to IBM, you will be asked to supply a severity level. Therefore, you need to understand and assess the business impact of the problem you are reporting. Use the following criteria:

Severity	Impact	Characteristic
1	Critical	You are unable to use the program, resulting in a critical impact on operations. This condition requires an immediate solution.
2	Significant	The program is usable but is severely limited.
3	Moderate	The program is usable with less significant features (not critical to operations) unavailable.

Severity	Impact	Characteristic
4	Minimal	The problem causes little impact on operations, or a reasonable circumvention to the problem has been implemented.

Describe your problem and gather background information

When explaining a problem to IBM, be as specific as possible. Include all relevant background information so that IBM Software Support specialists can help you solve the problem efficiently. To save time, know the answers to these questions:

- What software versions were you running when the problem occurred?
- Do you have logs, traces, and messages that are related to the problem symptoms? IBM Software Support is likely to ask for this information.
- Can the problem be recreated? If so, what steps led to the failure?
- Have any changes been made to the system? For example, hardware, operating system, networking software, and so on.
- Are you currently using a workaround for this problem? If so, you might be asked to explain it when you report the problem.

To find out what information and files you will need to supply when opening a problem management record (PMR), see [Collecting troubleshooting data for z/OS Connect Enterprise Edition](#).

Submit your problem to IBM Software Support

You can submit your problem in one of two ways:

- Online: Go to the Submit and track problems page on the IBM Software Support site. Enter your information into the appropriate problem submission tool.
- By phone: For the phone number to call in your country, go to the contacts page of the IBM Software Support Handbook on the Web and click the name of your geographic region.

If the problem you submit is for a software defect or for missing or inaccurate documentation, IBM Software Support will create an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Software Support will provide a workaround for you to implement until the APAR is resolved and a fix is delivered.

IBM publishes resolved APARs on the IBM product support Web pages daily, so that other users who experience the same problem can benefit from the same resolutions.

Prepare z/OS Connect EE for FFDC

When you have installed your z/OS Connect EE server, prepare it to capture vital diagnostic information if a problem occurs.

If a problem occurs, IBM Software Support might ask you to provide diagnostic information in the form of Java dumps, z/OS dumps, or TDUMPS. Prepare your system now to ensure you can capture this information at the first occurrence of the problem. This is called *First Failure Data Capture* or FFDC.

Java dumps

To find out where different Java dumps are written, specify the following parameter in the z/OS Connect EE server's JVM options file, or in STDENV JVM_OPTIONS= of the server's job card:

-Xdump:what

This causes the JVM to write output details of the dump configuration to the z/OS Connect EE server job's STDERR:

```

Registered dump agents
-----
-Xdump:system:
  events=gpf+user+abort+traceassert+corruptcache,
  dsn=%uid.JVM.%job.D%y%m%d.T%H%M%S.X&DS,
  range=1..0,
  priority=999,
  request=serial
-----
-Xdump:system:
  events=systhrow,
  filter=java/lang/OutOfMemoryError,
  dsn=%uid.JVM.%job.D%y%m%d.T%H%M%S.X&DS,
  range=1..1,
  priority=999,
  request=exclusive+compact+prepwalk
-----
-Xdump:heap:
  events=systhrow,
  filter=java/lang/OutOfMemoryError,
  file=/u/<userID>/heaptump.%Y%m%d.%H%M%S.%pid.%seq.phd,
  range=1..4,
  priority=500,
  request=exclusive+compact+prepwalk,
  opts=PHD
-----
-Xdump:java:
  events=gpf+user+abort+traceassert+corruptcache,
  file=/u/<userID>/javacore.%Y%m%d.%H%M%S.%pid.%seq.txt,
  range=1..0,
  priority=400,
  request=exclusive
-----
-Xdump:java:
  events=systhrow,
  filter=java/lang/OutOfMemoryError,
  file=/u/<userID>/javacore.%Y%m%d.%H%M%S.%pid.%seq.txt,
  range=1..4,
  priority=400,
  request=exclusive
-----
-Xdump:snap:
  events=gpf+abort+traceassert+corruptcache,
  file=/u/<userID>/Snap.%Y%m%d.%H%M%S.%pid.%seq.trc,
  range=1..0,
  priority=300,
  request=serial
-----
-Xdump:snap:
  events=systhrow,
  filter=java/lang/OutOfMemoryError,
  file=/u/<userID>/Snap.%Y%m%d.%H%M%S.%pid.%seq.trc,
  range=1..4,
  priority=300,
  request=serial
-----
-Xdump:jit:
  events=gpf+abort,
  file=/u/<userID>/jitedump.%Y%m%d.%H%M%S.%pid.%seq.dmp,
  range=1..0,
  priority=200,
  request=serial
-----
```

The uid is the OMVS uid specified in the OMVS segment of the RACF user ID. Note the uid should be unique to a single user ID.

Using this information from your system, you can confirm that the user ID that the z/OS Connect EE server is running under, has the required permissions to write all the related Java dumps.

z/OS dumps

Specify SYSMDUMP DD, SYSABEND DD, and SYSUDUMP DD cards on the z/OS Connect EE server job so that system dumps are written.

TDUMPS

The default location for Java TDUMPs is %uid.JVM.TDUMP.%job.D%y%m%d.T%H%M%S

Confirm that dumps can be written

It is possible to invoke some types of dump that the z/OS Connect EE server might request from the SDSF command line. Use this method to confirm that the dumps can be written before a failure occurs. To confirm that the z/OS Connect EE server can take the dump, run these commands under the user ID that starts the z/OS Connect EE server job.

Ensure TDUMPs can be written

Enter the following SDSF command: **/F <job_name>,TDUMP**

Check the job's output for messages:

```
JVMDUMP034I User requested System dump using '<user_ID>.JVM.  
<job_name>.D200813.T153740.X&DS' through JVMRI  
IEATDUMP in progress with options SDATA=(LPA,GRSQ,LSQA,NUC,PSA,RGN,SQA,SUM,SWA,TRT)  
IEATDUMP success for DSN='<user_ID>.JVM.BAQ30JOB.D200813.T153740.X&DS'  
JVMDUMP040I System dump written to dataset(s) using name template  
<user_ID>.JVM.<job_name>.D200813.T153740.X&DS
```

Ensure Javacore dumps can be written

Enter the following SDSF command: **/F <job_name>,JAVACORE**

Check the job's output for the following messages:

```
JVMDUMP034I User requested Java dump using <userID>/  
javacore.20200805.160557.67305723.0002.txt' through com.ibm.jvm.Dump.javaDumpToFile  
JVMDUMP010I Java dump written to <userID>/javacore.20200805.160557.67305723.0002.txt
```

If the javacore dump cannot be written, the following messages might be seen:

```
JVMDUMP030W Cannot write dump to file <userID>/javacore.20200805.160557.67305723.0002.txt:  
EDC5141I Read-only file system.  
JVMDUMP030W Cannot write dump to file /javacore.20150713.155831.11044.0002.txt: Permission  
denied
```

Ensure Heap dumps can be written

Enter the following SDSF command: **/F <job_name>,HEAPDUMP**

Check the job's output for messages:

```
JVMDUMP034I User requested Heap dump using '/u/<user_ID>/heapdump.202008  
14.131628.16974145.0001.phd' through com.ibm.jvm.Dump.heapDumpToFile  
CWWKB0005I: COMMAND RESPONSES COMPLETED SUCCESSFULLY FROM Heapdump Command Handler.  
CWWKB0002I: MODIFY COMMAND HEAPDUMP COMPLETED SUCCESSFULLY.
```

Ensure that an MVS system dump can be written

Issue the following SDSF modify commands to request a system dump:

```
/DUMP COMM=(<dump_title>)  
/R n,JOBNAME=(<jobname>,OMVS),CONT  
/R n,SDATA=(LPA,GRSQ,LSQA,NUC,PSA,RGN,  
/R n,SQA,SUM,SWA,TRT),END
```

Check the system log for messages:

```
IEA794I SVC DUMP HAS CAPTURED:  
DUMPID=004 REQUESTED BY JOB (*MASTER*)  
DUMP TITLE=<dump_title>  
IEA611I COMPLETE DUMP ON <dump_dataset_name>  
DUMPID=004 REQUESTED BY JOB (*MASTER*)  
FOR ASIDS(nnnn,nnnn)
```

Dump related environment variables

Heap dumps

_CEE_DMPTARG specifies the location where the heap dump is written.

TDUMP

JAVA_DUMP_TDUMP_PATTERN can be used to change the naming pattern of the TDUMP.

IBM_JAVA_ZOS_TDUMP=NO prevents TDUMPS being written.

IBM_JAVA_ZOS_TDUMP_COUNT= limits the number of TDUMPS a job can request.

If IBM_JAVA_ZOS_TDUMP_PATTERN= specifies an invalid value, TDUMPs are written to STDERR.

Java dumps

Check JAVA_DUMP_OPTS, Command line, and -Xdump output.

DISABLE_JAVADUMP=TRUE suppresses the default production of Java dumps.

IBM_JAVADUMP_OUTOFMEMORY=FALSE disables Java dumps for an out-of-memory exception. If not set, a Java dump is generated when an out-of-memory exception is thrown but not caught and handled by the application. Set to TRUE to generate a dump when an out-of-memory exception is thrown,

The TMPDIR=<directory> variable specifies an alternative temporary directory. This directory is used only when Java dumps and Heap dumps cannot be written to their target directories, or the current working directory. The default is /tmp.

Resolving startup problems

When you start a z/OS Connect EE server, a problem might occur before z/OS Connect EE can display an error message.

These problems are usually caused by configuration errors, so they are reported by Liberty.

RC=256 and message "cannot run server"

An attempt to start a z/OS Connect EE ends with RC = 256 and the following messages:

```
Error: zosconnect can not run server 'ABCDESRV', the server does not exist.  
Error: Unable to access jarfile /bin/tools/ws-server.jar
```

In the reported case, the tmp directory was found to be at 100%. The server.xml configuration file could not be loaded, so the server failed to start.

Check the tmp directory and remove any obsolete files to make more space.

XML syntax error in the configuration file

The following message is seen in the job log:

```
[WARNING ] CWWKF0009W: The server has not been configured to install any features.
```

The messages.log file contains more information:

```
[11/26/20 11:45:18:550 GMT] 00000017 com.ibm.ws.config.xml.internal.XMLConfigParser  
E CWWKG0014E: The configuration parser detected an XML syntax error while parsing the root of the  
configuration and the referenced configuration documents. Error: The element type "server" does
```

```
not match the expected end-tag "</httpEndpoint>". File: file:/var/zosconnect//servers/server1/server.xml Line: 214 Column: 9
[11/26/20 11:45:18:981 GMT] 0000000b com.ibm.ws.kernel.launch.internal.FrameworkManager
I CWNKE0002I: The kernel started after 10.448 seconds
[11/26/20 11:45:19:932 GMT] 00000022 com.ibm.ws.kernel.feature.internal.FeatureManager
I CWWKF0007I: Feature update started.
[11/26/20 11:45:19:953 GMT] 00000022 com.ibm.ws.kernel.feature.internal.FeatureManager
W CWWKF0009W: The server has not been configured to install any features.
```

Check the `server.xml` configuration file and correct any errors.

Resolving problems with security

The resources listed in this topic might help you solve problems associated with setting up security for z/OS Connect EE.

[MustGather: Documentation for diagnosing SSL handshake failures on Liberty Application Server for z/OS](#)

[MustGather: SSL problems on WebSphere Liberty](#)

[Set up trace and get a full dump for WebSphere Liberty](#)

Resolving performance problems

z/OS Connect EE runs as a Liberty application so performance problems are analysed using the tools provided by Liberty.

To analyze performance problems in Liberty, see [Troubleshooting performance issues in the WebSphere Application Server for z/OS Liberty](#) documentation.

For information about configuring z/OS Connect EE for best performance, see [Chapter 6, “Performance considerations,” on page 205](#).

Resolving Java memory problems

A `java.lang.OutOfMemoryError` occurs when a Java virtual machine cannot allocate an object because it is out of memory and more memory cannot be made available by the garbage collector.

Symptoms include failure to create a thread and failure to load Java classes. This situation can be caused by, for example; exhaustion of the Java heap, exhaustion of Java stack, exhaustion of the native stack, or the JIT compiler ran out of space. To determine the cause, you must determine why the object cannot be created. The z/OS Connect EE job log records that an `OutOfMemoryError` occurred. Examine the job log to find the cause.

Identifying the cause

THREADS

`OutOfMemoryError` reports an `errno 112`. For example,

```
java.lang.OutOfMemoryError: Failed to create a thread: retVal -1073741830, errno 112
(0x70), errno2 189858397 (0xb51025d) received
```

It is likely that a thread limit was reached. The number of threads that can be created by the server is limited by the OMVS BPXPARM MAXTHREADS and MAXTHREADTASKS, whichever is lower. If the MAXTHREADS or MAXTHREADTASKS limit is reached, any attempt to create another thread fails and the `OutOfMemoryError` occurs.

MEMORY

`OutOfMemoryError` reports an `errno 132`. For example,

```
java/lang/OutOfMemoryError" "Failed to create a thread: retVal -1073741830, errno 132
(0x84), errno2 -1055784930 (0xfffffffffc112001e)" received
```

Either the MEMLIMIT or HEAP sizes were reached:

- **MEMLIMIT** - The memory available in the JVM can be controlled by setting the MEMLIMIT parameter in the z/OS Connect EE started task procedure. The sample BAQSTRT specifies 8G. Java documents that each Java thread created in the JVM requires at least 3 MB above the bar for its stack storage and control blocks.
- **HEAP** - The OutOfMemoryError reports "Java heap space". For example, :

```
JVMDUMP013I Processed dump event "systhrow", detail "java/lang/OutOfMemoryError". java/
lang/OutOfMemoryError: Java heap space
```

Heap is exhausted. Increase the specified Xmx.

Monitoring Usage

THREADS

Monitor the OMVS thread usage of the z/OS Connect EE server by locating the process ID for the server in its messages.log file.

```
*****
product = WAS FOR Z/OS 19.0.0.3, z/OS Connect 03.00.22 (wlp-1.0.26.cl190320190321-1636)
wlp.install.dir = /u/zosconnect/v3r0/wlp/
server.config.dir = /var/zosconnect/servers/V3Server/
java.home = /java/java80_bit64_sr5_fp36/J8.0_64
java.version = 1.8.0_211
java.runtime = Java(TM) SE Runtime Environment (8.0.5.36 - pmz6480sr5fp36-20190510_01(SR5
FP36))
os = z/OS (02.02.00; s390x) (en_US)
process = 50528665@WINMVS01
*****
```

Enter the SDSF DISPLAY OMVS command to show the status of the process:

```
/D OMVS,LIMITS,PID=50528665
RESPONSE=MV01
BPX0051I 16.31.06 DISPLAY OMVS 699
OMVS 0010 ACTIVE OMVS=(P1,X3,DB,JV,00,RE,22,TG)
USER JOBNAME ASID PID PPID STATE START CT_SECS
TESTUSR BAQ30J0B 003E 50528665 33751390 HK----- 12.05.15 3601.7
LATCHWAITPID= 0 CMD=/java/java80_64/J8.0_64/bin/java -javaagent
PROCESS LIMITS: LIMMSG=None
          CURRENT   HIGHWATER   PROCESS
          USAGE     USAGE      LIMIT
MAXFILEPROC    220        225      65535
MAXFILESIZE    ---        ---      NOLIMIT
MAXPROCUSER    5          11       512
MAXQUEUEDSIGS  1          1       100000
MAXTHREADS     300        360      1000
MAXTHREADTASKS 302        360      365 [See text]
IPCSHMNSEGS    0          0       1000
MAXCORESIZE    ---        ---      4194304
MAXMEMLIMIT    908M       976M      4096M
```

The display shows that the thread usage reached the maximum value as set in MAXTHREADTASKS and would benefit from an increased maximum.

MEMORY

Monitor memory usage with IBM OMEGAMON for JVM on z/OS and the IBM Monitoring and Diagnostic Tools - Health Center Eclipse plug-in.

Configuration considerations

THREADS

Set the value of OMVS BPXPARM MAXTHREADS higher than the expected number of Java threads created in the z/OS Connect EE server.

In the WebSphere Liberty Profile environment, the number of Java threads is managed dynamically depending on workload so it is difficult to determine the number of threads created. To limit the number of Java threads that can be generated, configure the maximum number of default executor

threads allowed in the z/OS Connect EE server. In the following `server.xml` definition example, the maximum number of default executor threads, which run most of the workload in the server, is limited to no more than 300:

```
<executor maxThreads="300" name="Default Executor"/>
```

Note: There are other types of threads in the JVM, so there are more than the limit of default executor threads present.

MEMLIMIT

The suggested initial setting for MEMLIMIT in the z/OS Connect EE server's job is 8 GB.

Every thread running AMODE64 requires a minimum of 3 MB of above-the-bar virtual storage for each thread's stack storage. If you specify a maximum stack size of less than 1 MB, the value is rounded up so that 3 MB of storage is used. For example, for a large application with 1000 threads, the MEMLIMIT would need to be increased by 3 GB ($1000 * 3$ MB) for thread stack virtual storage. So increasing the MAXTHREADS to a very high value might cause excessive usage of memory above the bar in a heavily used server.

HEAP

- The default Xmx value of 256 M is usually sufficient.
- Every thread occupies 1.6 KB of heap storage.
- Every IPIC session uses 300 KB of the heap for its send and receive buffers. The default number of sessions per IPIC connection is 100. So when an IPIC connection is established, with 100 sessions, 30 MB of Java heap is allocated.
- The IPIC payload data for in-flight requests is allocated from the heap.

Note:

1. The allocation for the Java heap is allocated out of the above the bar memory on JVM startup.
2. Increasing the heap reduces the available memory from the region MEMLIMIT. If the Xmx specification is increased, the MEMLIMIT specification should be increased by a similar amount.
3. If the problem is not related to heap exhaustion, increasing the heap can make the problem worse.

Chapter 27. Messages

z/OS Connect EE messages are grouped into these categories:

Runtime messages

Messages that are generated by the z/OS Connect EE runtime functions start with the prefix BAQR.

REST client messages

Messages that are generated by the z/OS Connect EE REST client have the prefix BAQR.

CICS service provider messages

Messages that are generated by the z/OS Connect EE CICS service provider have the prefix BAQR.

MQ service provider Messages

Messages that are generated by the z/OS Connect EE MQ service provider have the prefix BAQM.

Policy messages

Messages that are generated by the z/OS Connect EE Policy have the prefix BAQR.

WOLA service provider messages

Messages that are generated by the z/OS Connect EE WOLA service provider have the prefix BAQR.

API requester messages

Messages that are generated by the z/OS Connect EE API requester have the prefix BAQR.

Endpoint connection messages

Messages that are generated by the z/OS Connect EE endpoint connection have the prefix BAQR.

SAF Messages

Messages that are generated by the z/OS Connect EE SAF interface have the prefix BAQR.

Audit interceptor messages

Messages that are generated by the z/OS Connect EE audit interceptor have the prefix BAQR.

File system logger interceptor messages

Messages that are generated by the z/OS Connect EE File system logger interceptor have the prefix BAQR.

Deployment utility messages

Messages that are generated by the z/OS Connect EE deployment utility have the prefix BAQD.

Utility messages

Messages that are generated by the z/OS Connect EE utility functions have the prefix BAQU.

Build toolkit messages

Messages that are generated by the z/OS Connect EE build toolkit have the prefix BAQB.

Build Toolkit MQ Messages

Messages that are generated by the z/OS Connect EE build toolkit for MQ have the prefix BAQB.

Communication stub (CICS) messages

Messages that are generated by the z/OS Connect EE communication stub for CICS start with the prefix BAQT.

These messages include a *[date time]* insert where the *date* format is yyyy/MM/dd and the *time* format is HH:mm:ss. The date and time are taken from the system time-of-day clock, which is adjusted for leap seconds, and apply the local time zone offset, including Daylight Saving Time. Hours represent the 24-hour clock and seconds are truncated.

BAQT0001E *[date time]* URIMAP(*mapname*) is not found.

Explanation The named URIMAP resource was not found.

Action Install URIMAP (*mapname*), or verify the resource installation in the CICS region, and then rerun the application.

BAQT0002E [date time] URIMAP(*mapname*) is not available.

- Explanation** The named URIMAP resource is not available.
- Action** Verify that URIMAP (*mapname*) is in ENABLED status, and rerun the application.

BAQT0003E [date time] Hostname in URIMAP(*mapname*) cannot be resolved.

- Explanation** The hostname cannot be resolved, or the format of the host option is incorrect.
- Action** Correct the hostname that is configured in the named UIRMAP resource.

BAQT0004E [date time] Timeout occurred before a connection to the z/OS Connect EE server could be established.

- Explanation** A connection to the z/OS Connect EE server could not be established likely due to a network issue.
- Action** Examine the network path from your subsystem to the z/OS Connect EE server to identify the failure point.

BAQT0005E [date time] Timeout occurred before the request could be sent to the z/OS Connect EE server.

- Explanation** The communication stub was not able to send the request to the z/OS Connect EE server before the connection timed out, likely due to a network issue.
- Action** Examine the network path from your subsystem to the z/OS Connect EE server to identify the failure point.

BAQT0006E [date time] Timeout occurred before a response could be received from the z/OS Connect EE server.

- Explanation** A timeout occurred when the communication stub attempted to receive a response from the z/OS Connect EE server, likely due to a network issue.
- Action** Examine the network path from your subsystem to the z/OS Connect EE server to identify the failure point.

BAQT0007E [date time] The length of the response message is invalid.

- Explanation** The response message exceeds the length that is specified by **BAQ-RESPONSE-LEN**, and the excess part is discarded. This error might occur because the language specified in the build toolkit properties file is not the same as the language used with the client z/OS application or the response data structure is not correct. This error can also occur if **BAQ-RESPONSE-PTR** is NULL or **BAQ-RESPONSE-LEN** is 0.
- Action** Check the language specification in the build toolkit properties file and ensure it is consistent with the one that is used with your client z/OS application. Ensure that the response data structure and the value that is specified by **BAQ-RESPONSE-LEN** are correct. Check that when the communication stub is called, the values for **BAQ-RESPONSE-PTR** and **BAQ-RESPONSE-LEN** are valid.

BAQT0008E [date time] Socket error.

Explanation	A socket error occurred, which might be caused by one of the following errors:
	<ul style="list-style-type: none">The system initialization parameter TCPIP is set or defaults to NO for this CICS region.The port that is configured in CICS region does not match the port on which the z/OS Connect EE server runs.
Action	Check the CICS log to investigate this error.

BAQT0009E [date time] Incorrect client certificate.

Explanation	The client certificate that is defined in URIMAP(<i>mapname</i>) is incorrect.
Action	Correct the client certificate and rerun the application.

BAQT0010E [date time] All requested cipher codes were rejected.

Explanation	The cipher codes might be incorrectly generated or corrupted.
Action	Correct the cipher codes and rerun the application.

BAQT0011E [date time] An unexpected error occurred when BAQCSTUB processed a request for REST API [REST API name]. EXEC CICS [CICS API name] gave RESP [RESP value] and [RESP2 value].

Explanation	An unexpected error occurred when the communication stub uses the CICS API to process the request for the REST API.
Action	Check the CICS product documentation to further identify the error reason with the name and response code of the CICS API.

BAQT0012E [date time] The communication stub failed to connect the z/OS Connect EE server.

Explanation	The communication stub failed to connect the z/OS Connect EE server. A possible cause is the z/OS Connect EE server was stopped.
Action	Check the status of the z/OS Connect EE server and ensure it is started.

BAQT0013E [date time] The size of the storage for the request message is invalid.

Explanation	The size of the storage for the request message that is specified by BAQ_REQUEST_LEN is not greater than zero.
Action	Ensure the value of BAQ_REQUEST_LEN is greater than zero.

BAQT0014E [date time] OAuth parameter username length is invalid.

Explanation	The username length that is specified by the BAQ-OAUTH-USERNAME-LEN parameter does not match the actual length of username that is specified by the BAQ-OAUTH-USERNAME parameter or is longer than the maximum length allowed. The maximum value is 256.
Action	Correct the BAQ-OAUTH-USERNAME-LEN value. Ensure that the input value is in the range 1 - 256 characters and matches the actual length of the username that is specified by the BAQ-OAUTH-USERNAME parameter.

BAQT0015E [date time] OAuth parameter password length is invalid.

Explanation	The password length that is specified by the BAQ-OAUTH-PASSWORD-LEN parameter does not match the actual length of the password that is specified by the BAQ-OAUTH-PASSWORD parameter or is longer than the maximum length allowed. The maximum value is 256.
Action	Correct the BAQ-OAUTH-PASSWORD-LEN value. Ensure that the input value is in the range 1 - 256 characters and matches the actual length of the password that is specified by BAQ-OAUTH-PASSWORD .

BAQT0016E [date time] OAuth parameter client ID length is invalid.

Explanation	The client ID length that is specified by the BAQ-OAUTH-CLIENTID-LEN parameter does not match the actual length of the client ID that is specified by the BAQ-OAUTH-CLIENTID parameter or is longer than the maximum length allowed. The maximum value is 256.
Action	Correct the BAQ-OAUTH-CLIENTID-LEN value. Ensure that the input value is in the range 1 - 256 characters and matches the actual length of client ID specified by BAQ-OAUTH-CLIENTID .

BAQT0017E [date time] OAuth parameter client secret length is invalid.

Explanation	The client secret length that is specified by the BAQ-OAUTH-CLIENT-SECRET-LEN parameter does not match the actual length of the client secret that is specified by the BAQ-OAUTH-CLIENT-SECRET parameter or is longer than the maximum length allowed. The maximum value is 256.
Action	Correct the BAQ-OAUTH-CLIENT-SECRET-LEN value. Ensure that the input value is in the range 1 - 256 characters and matches the actual length of client secret that is specified by BAQ-OAUTH-CLIENT-SECRET .

BAQT0018E [date time] The communication stub is misused.

Explanation	The incorrect communication stub was used for the environment.
Action	Check the version of the communication stub configured in your CICS region. Ensure that the communication stub program BAQCSTUB from the hlq.SBAQLIB1 load library is used.

BAQT0019E [date time] An error occurred in the communication stub [reason].

Explanation	An unexpected error occurred in the communication stub.
Action	Contact IBM software support.

BAQT0020E [date time] The communication stub currently in use cannot process the [name] data structure because [element] has an invalid value '[value]'.

Explanation	The communication stub currently in use cannot process the named data structure because the named element has a value that is not supported by the communication stub.
--------------------	--

Action	Ensure the build toolkit that is used to generate the data structure, the communication stub, and the z/OS Connect EE server are compatible with each other. Where possible, use the build toolkit, the communication stub, and the z/OS Connect EE server from the same installation package to avoid incompatibility problems.
---------------	--

BAQT0021E [date time] TOKEN parameter username length is invalid.

Explanation	The username length that is specified by the BAQ-TOKEN-USERNAME-LEN parameter does not match the actual length of the username that is specified by the BAQ-TOKEN-USERNAME parameter or is longer than the maximum length allowed. The maximum value is 256.
Action	Correct the value of the BAQ-TOKEN-USERNAME-LEN parameter. Ensure that the input value is in the range 1 - 256 characters and matches the actual length of the username that is specified by the BAQ-TOKEN-USERNAME parameter.

BAQT0022E [date time] TOKEN parameter password length is invalid.

Explanation	The password length that is specified by the BAQ-TOKEN-PASSWORD-LEN parameter does not match the actual length of the password that is specified by the BAQ-TOKEN-PASSWORD parameter or is longer than the maximum length allowed. The maximum value is 256.
Action	Correct the value of the BAQ-TOKEN-PASSWORD-LEN parameter. Ensure that the input value is in the range 1 - 256 characters and matches the actual length of the password that is specified by the BAQ-TOKEN-PASSWORD parameter.

BAQT0023E [date time] The communication stub cannot process the [name] data structure because the pointer is null.

Explanation	The communication stub cannot process the named data structure because the pointer to the structure has a null value.
Action	Ensure the pointer passed to BAQCSTUB contains the address of the data structure.

BAQT0024E [date time] The communication stub cannot process the [name] data structure because the [element] element is not defined.

Explanation	The communication stub cannot process the named data structure because the value for the named element is null or has a length of zero.
Action	Ensure the value of the element for the named data structure is correct when passed to BAQCSTUB.

BAQT0025E [date time] The communication stub cannot write message [number] because the message text is too long.

Explanation	The communication stub cannot write message <i>number</i> to CICS TDQUEUE BAQQ because the message text is too large for the defined TD QUEUE record size.
Action	Ensure that the CICS definition for TDQUEUE BAQQ is defined with a RECORDSIZE of 508 and a BLOCKSIZE of 512.

BAQT0026E [date time] BAQ-ZCON-SERVER-URI URIMAP *urimap* does not match CICS resource naming standards.

Explanation	The URIMAP resource name does not match the CICS TS resource naming standards and is invalid.
Action	Ensure that the URIMAP resource name matches the CICS URIMAP definition.

BAQT0027E [date time] BAQ-ZCON-SERVER-URI URIMAP *urimap* is too long. A maximum of 8 characters is allowed.

Explanation	CICS TS URIMAP resource names are a maximum of 8 characters in length.
Action	Ensure that the URIMAP resource name matches the CICS URIMAP definition.

BAQT0028I Attempting to enable the TRUE.

Explanation	The program BAQCTENA is started and attempting to enable the TRUE.
Action	No further action is needed.

BAQT0029W The TRUE is already enabled.

Explanation	The program BAQCTENA found that the TRUE is already enabled and will not attempt to enable the TRUE.
Action	The TRUE is found to be enabled. If the TRUE needs to be restarted, run the BAQD transaction to first disable the TRUE before you attempt to enable it again.

BAQT0030E CICS could not initialize the TRUE resp=[RESP value] resp2=[RESP2 value].

Explanation	BAQCTENA called EXEC CICS ENABLE PROGRAM , but it did not receive a normal response code. The <code>resp</code> and <code>resp2</code> values from CICS are shown.
Action	Ensure that the TRUE is correctly configured in CICS. If this problem persists, contact IBM Software Support.

BAQT0031E CICS could not set the TRUE global work area resp=[RESP value] resp2=[RESP2 value].

Explanation	BAQCTENA called EXEC CICS EXTRACT EXIT PROGRAM , but it did not receive a normal response code. The <code>resp</code> and <code>resp2</code> values from CICS are shown.
Action	Ensure that the TRUE is correctly configured in CICS. If this problem persists, contact IBM Software Support.

BAQT0032E CICS could not load BAQCTRUA for the TRUE resp=[RESP value] resp2=[RESP2 value].

Explanation	BAQCTENA called EXEC CICS LOAD PROGRAM , but it did not receive a normal response code. The <code>resp</code> and <code>resp2</code> values from CICS are shown.
Action	Ensure that the TRUE is correctly configured in CICS. If this problem persists, contact IBM Software Support.

BAQT0033E CICS could not enable the TRUE resp=[RESP value] resp2=[RESP2 value].

Explanation	BAQCTENA called EXEC CICS ENABLE PROGRAM , but it did not receive a normal response code. The <code>resp</code> and <code>resp2</code> values from CICS are shown.
Action	Ensure that the TRUE is correctly configured in CICS. If this problem persists, contact IBM Software Support.

BAQT0034I The TRUE was successfully enabled.

Explanation	BAQCTENA successfully enabled the TRUE.
Action	No further action is needed.

BAQT0035I Attempting to disable the TRUE.

Explanation	The program BAQCTDIS is started and attempting to disable the TRUE.
Action	No further action is needed.

BAQT0036W The TRUE is not enabled resp=[RESP value] resp2=[RESP2 value].

Explanation	The program BAQCTDIS called EXEC CICS EXTRACT EXIT and found that the TRUE is not enabled.
Action	The TRUE is not enabled and the <code>resp</code> and <code>resp2</code> values from CICS are shown. If the TRUE needs to be enabled, run the BAQE transaction.

BAQT0037E CICS could not disable the TRUE resp=[RESP value] resp2=[RESP2 value].

Explanation	BAQCTDIS called EXEC CICS DISABLE PROGRAM , but it did not receive a normal response code. The <code>resp</code> and <code>resp2</code> values from CICS are shown.
Action	Ensure that the TRUE is correctly configured in CICS. If this problem persists, contact IBM Software Support.

BAQT0038E CICS could not release BAQCTRUA for the TRUE resp=[RESP value] resp2=[RESP2 value].

Explanation	BAQCTDIS called EXEC CICS RELEASE PROGRAM , but it did not receive a normal response code. The <code>resp</code> and <code>resp2</code> values from CICS are shown.
Action	Ensure that the TRUE is correctly configured in CICS. If this problem persists, contact IBM Software Support.

BAQT0039I The TRUE was successfully disabled.

Explanation	BAQCTDIS successfully disabled the TRUE.
Action	No further action is needed.

BAQT0040E Another program is already attempting to enable or disable the TRUE

Explanation	The program was unable to run exclusively to configure the TRUE.
--------------------	--

Action	Check for another task that is running either the BAQCTENA or BAQCTDIS programs and wait for this task to complete before you try again. If this problem persists, contact IBM Software Support.
---------------	--

BAQT0041E It was not possible to complete the enabling or disabling of the TRUE

Explanation	The program was unable to tidy up its resources as it was ending.
--------------------	---

Action	Attempt to run the program again and if this problem persists, contact IBM Software Support.
---------------	--

Communication stub (IMS and z/OS) messages

Messages that are generated by the z/OS Connect EE communication stub for IMS and other z/OS applications (not CICS) start with the prefix BAQI.

BAQI0001E: z/OS Connect EE server URI [*uri*] is invalid. [*error_detail*]

Explanation	The server URI that is specified in the BAQURI environment variable is invalid. The reported [<i>error_detail</i>] indicates the reason. If an HWTHXXXX error is reported, the RSN value of the HWTHXXXX error message provides additional information.
--------------------	---

Action	Check the BAQURI specification in the CEEOPTS DD statement.
---------------	---

BAQI0002E: z/OS Connect EE port number [*port*] is invalid. [HWTHXXXX_error]

Explanation	The server port number that is specified in the BAQURI environment variable is invalid. The RSN value in the HWTHXXXX error message provides additional information.
--------------------	--

Action	Check the BAQPORT specification in the CEEOPTS DD statement. Valid port numbers are from 0 to 65536.
---------------	--

BAQI0003E: Unable to connect to the z/OS Connect EE server URI [*uri*] on port [*port*]. [HWTHXXXX_error]

Explanation	The URI cannot be resolved, the port number is incorrect, or a network issue occurs. The RSN value in the HWTHXXXX error message provides additional information. For example, if the hostname does not resolve, the HWTHCONN error would be as follows: HWTHCONN RC=262 RSC=1 RSN=EDC9501I The name does not resolve for the supplied parameters.
--------------------	--

Action	Check and correct the URI (BAQURI) and port number (BAQPORT) in the CEEOPTS DD statement. Review the log for any network issue.
---------------	---

BAQI0004E: Configuration property [*property_name*] was not specified.

Explanation	The value for the required property [<i>property_name</i>] was not specified in the CEEOPTS DD statement.
--------------------	---

Action	Ensure that the URI (BAQURI) and port number (BAQPORT) in the CEEOPTS DD statement are properly specified.
---------------	--

BAQI0005E: Unable to send request to or receive response from the z/OS Connect EE server. [HWTHXXXX_error]**Explanation**

A connection to the z/OS Connect EE server cannot be established. There are a number of possible causes for this error. The RSN value in the HWTHXXXX error message provides additional information. A common cause is a connection timeout, for which you would see the following error:

```
BAQ return code type: 3
BAQ status code: 5
BAQ status message: BAQI0005E: Unable to send request to or receive
response from the z/OS Connect EE server. HWTHRQST RC=262 RSC=1102
RSN=EDC8102I Operation would block.
```

Action

Review the RSN value in the HWTHXXXX error message for additional information and examine the network path from your subsystem to the z/OS Connect EE server to identify the failure point. For a connection timeout, you might need to set BAQTIMEOUT option to a larger value.

BAQI0006E: Unable to set property [property_name] to value [value]. [HWTHXXXX_error]**Explanation**

The communication stub was unable to complete the request for the specified reason.

Action

Correct the IMS or batch application configuration and retry the request.

BAQI0007E: The specified pointer to the storage for the response message is null.**Explanation**

The response message contains payload but the pointer is set to null. Response data cannot be copied to the storage specified in the application.

Action

If the response message is expected or likely to contain payload data, ensure that the pointer to the storage for the response message is not set to null.

If your application MYAPIREQ1, as shown in the following COBOL sample, uses the variable reference BAQ-RESPONSE-PTR as the pointer to the storage for the response message, ensure that the pointer is not null and points to a valid address. Ensure the storage is large enough to hold the response message.

```
CALL MYAPIREQ1 USING
      BY REFERENCE    API-INFO
      BY REFERENCE    BAQ-REQUEST-INFO
      BY REFERENCE    BAQ-REQUEST-PTR
      BY REFERENCE    BAQ-REQUEST-LEN
      BY REFERENCE    BAQ-RESPONSE-INFO
      BY REFERENCE    BAQ-RESPONSE-PTR
      BY REFERENCE    BAQ-RESPONSE-LEN.
```

BAQI0008E: z/OS Connect EE server URI [uri] is greater than 255 characters. [HWTHXXXX_error]**Explanation**

The given URI is longer than can be handled by the communication stub

Action

Retry the request using a name that is shorter than the maximum given or use a dotted IP address.

BAQI0009E: The name of the API is not specified or the length is invalid.

Explanation	The name of the API is not specified or the length is invalid likely because the generated API information file was altered.
Action	Regenerate the API information file.

BAQI0010E: The path to the API is not specified or the length is invalid.

Explanation	The path to the API is not specified or the length is invalid. A possible cause is that the generated API information file was altered.
Action	Regenerate the API information file.

BAQI0011E: The HTTP method to invoke the API is not specified or the length is invalid.

Explanation	The HTTP method to invoke the API is not specified or the length is invalid likely because the generated API information file was altered.
Action	Regenerate the API information file.

BAQI0012E: The specified storage size for the request message is invalid.

Explanation	The storage size specified for the request message is not valid. The size cannot be a negative value.
Action	Correct the size in your application.

BAQI0013E: The specified storage size for the response message is invalid.

Explanation	The storage size specified for the response message is not valid. The size cannot be a negative value.
Action	Correct the size in your application.

BAQI0014E: Verbose value [value] is invalid.

Explanation	The BAQVERBOSE option is not specified correctly in the CEEOPTS DD statement.
Action	Correct the value for the BAQVERBOSE option in your CEEOPTS DD statement. For more information about this option, see “Configuring IMS to access z/OS Connect EE for API calls” on page 309 .

BAQI0015E: Value for BAQVERBOSE_DEST_HWT [value] is invalid.

Explanation	The BAQVERBOSE_DEST_HWT option is not specified correctly in the CEEOPTS DD statement.
Action	Correct the value for the BAQVERBOSE_DEST_HWT option in your CEEOPTS DD statement. For more information about this option, see “Configuring IMS to access z/OS Connect EE for API calls” on page 309 .

BAQI0016E: Timeout value [value] is invalid. [HWTHXXXX_error]

Explanation	Valid BAQTIMEOUT values are 1 - 2,678,400 seconds. The RSN value in the HWTHXXXX error message provides additional information.
Action	Correct the timeout value in your CEEOPTS DD statement.

BAQI0017E: Username [user_name] is invalid. Length is greater than the maximum of [number] characters. [error_detail]

Explanation	The specified username cannot be properly set before it is sent to the z/OS Connect EE server for validation. The reported <i>[error_detail]</i> indicates the reason. If an HWTHXXXX error is reported, the RSN value of the HWTHXXXX error message provides additional information.
Action	Correct the BAQUSERNAME value in your CEEOPTS DD statement.

BAQI0018E: Password is invalid. Length is greater than the maximum of [number] characters. [error_detail]

Explanation	The specified password cannot be properly set before it is sent to the z/OS Connect EE for validation. The reported <i>[error_detail]</i> indicates the reason. If an HWTHXXXX error is reported, the RSN value of the HWTHXXXX error message provides additional information.
Action	Correct the BAQPASSWORD value in your CEEOPTS DD statement.

BAQI0019E: Unable to disconnect and close the socket to the z/OS Connect EE server. [HWTHXXXX_error]

Explanation	The communication stub is unable to disconnect and close the socket connection to the z/OS Connect EE. The RSN value in the HWTHXXXX error message provides additional information.
Action	Contact IBM software support and provide the HWTH error detail.

BAQI0020E: Unable to terminate the connection to the z/OS Connect EE server. [HWTHXXXX_error]

Explanation	Resources that were allocated for the connection cannot be cleaned up. The RSN value in the HWTHXXXX error message provides additional information.
Action	Contact IBM software support and provide the HWTH error detail.

BAQI0021E: OAuth parameter username length is invalid.

Explanation	The username length specified by the BAQ-OAUTH-USERNAME-LEN parameter does not match the actual length of username specified by the BAQ-OAUTH-USERNAME parameter or is longer than the maximum length allowed. The maximum value is 256.
Action	Correct the BAQ-OAUTH-USERNAME-LEN value. Ensure the input value is between 1 and 256 characters and matches the actual length of username specified by BAQ-OAUTH-USERNAME.

BAQI0022E: OAuth parameter password length is invalid.

Explanation	The password length specified by the BAQ-OAUTH-PASSWORD-LEN parameter does not match the actual length of password specified by the BAQ-OAUTH-PASSWORD parameter or is longer than the maximum length allowed. The maximum value is 256.
Action	Correct the BAQ-OAUTH-PASSWORD-LEN value. Ensure the input value is between 1 and 256 characters and matches the actual length of password specified by BAQ-OAUTH-PASSWORD.

BAQI0023E: OAuth parameter client ID length is invalid.

Explanation The client ID length specified by the BAQ-OAUTH-CLIENTID-LEN parameter does not match the actual length of client ID specified by the BAQ-OAUTH-CLIENTID parameter or is longer than the maximum length allowed. The maximum value is 256.

Action Correct the BAQ-OAUTH-CLIENTID-LEN value. Ensure the input value is between 1 and 256 characters and matches the actual length of client ID specified by BAQ-OAUTH-CLIENTID.

BAQI0024E: OAuth parameter client secret length is invalid.

Explanation The client secret length specified by the BAQ-OAUTH-CLIENT-SECRET-LEN parameter does not match the actual length of client secret specified by the BAQ-OAUTH-CLIENT-SECRET parameter or is longer than the maximum length allowed. The maximum value is 256.

Action Correct the BAQ-OAUTH-CLIENT-SECRET-LEN value. Ensure the input value is between 1 and 256 characters and matches the actual length of client secret specified by BAQ-OAUTH-CLIENT-SECRET.

BAQI0025E: The communication stub is misused.

Explanation The incorrect communication stub has been used for the environment.

Action Check the version of the communication stub configured in your IMS region (or z/OS application). Ensure that the communication stub program BAQWEBT alias BAQCSTUB from the hlq .SBAQLIB load library is used.

BAQI0026E: Unable to request additional memory storage..

Explanation The communication stub was unable to allocate additional memory to complete the request.

Action Ensure that the IMS or batch application has been configured correctly. If the problem persists, Contact IBM software support.

BAQI0027E: The communication stub currently in use cannot process the BAQ* data structure.

Explanation The communication stub currently in use cannot process the BAQ* data structure because the compatibility level defined in the data structure is not a value that is supported by the communication stub.

Action Check the compatibility level that is specified in the request or the response data structure is correct, check the values of BAQ-REQUEST-INFO-COMP-LEVEL and BAQ-RESPONSE-INFO-COMP-LEVEL. Ensure the build toolkit that is used to generate the API-INFO data structure, the communication stub and the z/OS Connect EE server are compatible with each other. It is recommended that you use the build toolkit, the communication stub and the z/OS Connect EE server that are installed from the same installation package to avoid the incompatibility problem.

BAQI0028E: Parameter error - a NULL value is defined for [parameter]

Explanation A NULL value was specified for the *parameter* when the associated length value is not zero.

Action	Correct the <i>parameter</i> value or change the associated length value to zero.
---------------	---

BAQI0029E: Internal error - Storage allocation error.

Explanation	The communication stub was unable to allocate additional memory to complete the request.
--------------------	--

Action	Ensure the IMS or batch application is configured correctly. If the problem persists, contact IBM software support.
---------------	---

BAQI0030E: Internal error - Unable to get job owner id.

Explanation	The communication stub was unable to get job owner id.
--------------------	--

Action	Contact IBM software support and provide the communication stub trace.
---------------	--

BAQI0031E: TOKEN parameter username length is invalid.

Explanation	The username length specified for the BAQ-TOKEN-USERNAME-LEN parameter does not match the actual length of username specified for the BAQ-TOKEN-USERNAME parameter or is longer than the maximum length allowed. The maximum value is 256.
--------------------	--

Action	Correct the value of the BAQ-TOKEN-USERNAME-LEN parameter. Ensure the input value is between 1 and 256 characters and matches the actual length of the username specified for the BAQ-TOKEN-USERNAME parameter.
---------------	---

BAQI0032E: TOKEN parameter password length is invalid.

Explanation	The password length specified for the BAQ-TOKEN-PASSWORD-LEN parameter does not match the actual length of password specified for the BAQ-TOKEN-PASSWORD parameter or is longer than the maximum length allowed. The maximum value is 256.
--------------------	--

Action	Correct the value of the BAQ-TOKEN-PASSWORD-LEN parameter. Ensure the input value is between 1 and 256 characters and matches the actual length of the password specified for the BAQ-TOKEN-PASSWORD parameter.
---------------	---

BAQI0033E: The specified pointer to API-INFO is null.

Explanation	The parameters passed to BAQCSTUB contain a null value for API-INFO.
--------------------	--

Action	Ensure the pointer passed to BAQCSTUB contains the address of API-INFO.
---------------	---

BAQI0034E: The specified pointer to BAQ-REQUEST-INFO is null.

Explanation	The parameters passed to BAQCSTUB contain a null value for BAQ-REQUEST-INFO.
--------------------	--

Action	Ensure the pointer passed to BAQCSTUB contains the address of BAQ-REQUEST-INFO.
---------------	---

BAQI0035E: Web Toolkit verbose failed to set with HSet. [HWTHXXXX_error]

Explanation	Setting of HWTH_OPT_VERBOSE option for the HTTP request failed. The RSN value in the HWTHXXXX error message provides additional information.
--------------------	--

Action	Contact IBM software support and provide the HWTH error detail.
---------------	---

BAQI0901E: HTTP connection initiation failed with the Init Connection type [type].

Explanation	Failed to initiate an HTTP connection with the specified Init Connection type.
Action	Contact IBM software support and provide the communication stub trace.

BAQI0902E: HTTP connection initiation failed with the Init request type [type].

Explanation	Failed to initiate an HTTP connection with the specified Init request type.
Action	Contact IBM software support and provide the communication stub trace.

BAQI0903E: Failed to set the value [value] for the request option [option_name].

Explanation	Setting of the specified option value in the HTTP request failed.
Action	Contact IBM software support and provide the communication stub trace.

BAQI0904E: The [HWTH_OPT_URI] value of the request has a length of [length] that exceeds [maximum_length].

Explanation	The request length is greater than the specified maximum value.
Action	Ensure the request length is less than the specified maximum value.

BAQI0905E: Failed to set the value [value] for the HTTP header option [option_name].

Explanation	Setting of the specified HTTP header value for the HTTP request failed.
Action	Contact IBM software support and provide the communication stub trace.

GMOM messages - API toolkit data transformation diagnostic

The z/OS Connect EE API toolkit includes a data transformation function that converts between byte arrays and JSON. This function provides additional information about the error symptoms and reports them as GMOMW messages.

GMOMW0005E A data type conversion error occurred while the leaf field *field_name* of service interface *service_interface_name* was converted: *Error_details*.

Explanation

Check the data schema. Most likely the data type does not match the schema, overflows, is too long, is out of range, or is uninitialized.

Programmer response

Use the HTTP method GET to call the administration interface to get the correct data schema:

```
/zosConnect/services/{serviceName}/schema/  
request  
/zosConnect/services/{serviceName}/schema/  
response
```

GMOMW0006E The byte array was exhausted while the leaf field *field_name* of service interface *service_interface_name* was read and processed.

Explanation:

Check the data schema. Most likely the data type does not match the schema, overflows, is too long, or is out of range.

Programmer response

Use the HTTP method GET to call the administration interface to get the correct data schema:

```
/zosConnect/services/{serviceName}/schema/  
request  
/zosConnect/services/{serviceName}/schema/  
response
```

GMOMW0007E	The field <i>field_name</i> of service interface <i>service_interface_name</i> is not a composite.
-------------------	---

Explanation:

The named field was expected to be a composite.

Programmer response

Use the HTTP method GET to call the administration interface to get the correct data schema:

```
/zosConnect/services/{serviceName}/schema/  
request  
/zosConnect/services/{serviceName}/schema/  
response
```

GMOMW0008E	The field <i>field_name</i> of service interface <i>service_interface_name</i> is not a composite.
-------------------	---

Explanation:

The named field was expected to be a composite.

Programmer response

Use the HTTP method GET to call the administration interface to get the correct data schema:

```
/zosConnect/services/{serviceName}/schema/  
request  
/zosConnect/services/{serviceName}/schema/  
response
```

GMOMW0009E	The field <i>field_name</i> of service interface <i>service_interface_name</i> is not a leaf.
-------------------	--

Explanation:

The named field was expected to be a leaf.

Programmer response

Use the HTTP method GET to call the administration interface to get the correct data schema:

```
/zosConnect/services/{serviceName}/schema/  
request  
/zosConnect/services/{serviceName}/schema/  
response
```

GMOMW0010E	The field <i>field_name</i> of service interface <i>service_interface_name</i> is not a leaf array.
-------------------	--

Explanation:

The named field was expected to be a leaf array.

Programmer response

Use the HTTP method GET to call the administration interface to get the correct data schema:

```
/zosConnect/services/{serviceName}/schema/  
request
```

```
/zosConnect/services/{serviceName}/schema/  
response
```

GMOMW0011E	The maximum of <i>max_number_of_entries</i> entries for the leaf array <i>field_name</i> of service interface <i>service_interface_name</i> was exceeded. The number of entries found is <i>number_of_entries</i>.
-------------------	---

Explanation:

The number of the entries for the leaf array exceeds what was expected.

Programmer response

Use the HTTP method GET to call the administration interface to get the correct data schema:

```
/zosConnect/services/{serviceName}/schema/  
request  
/zosConnect/services/{serviceName}/schema/  
response
```

GMOMW0012E	The maximum of <i>max_number_of_entries</i> entries for the composite array <i>field_name</i> of service interface <i>service_interface_name</i> was exceeded. The number of entries found is <i>number_of_entries</i>.
-------------------	--

Explanation:

The number of the entries for the composite array exceeds what was expected.

Programmer response

Use the HTTP method GET to call the administration interface to get the correct data schema:

```
/zosConnect/services/{serviceName}/schema/  
request  
/zosConnect/services/{serviceName}/schema/  
response
```

GMOMW0013E	Leaf field <i>field_name</i> with the occurrence count of array <i>array_name</i> of service interface <i>service_interface_name</i> cannot be updated: <i>error_details</i>
-------------------	---

Explanation:

Examine the error details for more information about the error. Most likely the data type of the leaf field is not an integer or not compatible with integer.

Programmer response:

Verify that the data type of the leaf field is compatible with integer and try again.

GMOMW0014E	An error occurred when leaf field <i>field_name</i> was read to obtain the
-------------------	---

**occurrence count of array
array_name of service interface
service_interface_name:
error_details**

Explanation:

Examine the error details for more information about the error. Most likely the data type of the leaf field array count is not an integer or not compatible with integer.

System action:

An HTTP 500 error is returned.

Programmer response:

Verify that the data type of the array count is compatible with integer and try again.

GMOMW0015E A mismatch exists between the service interface and the overrides overlay of the service. Any time a data structure is changed in the transaction editor you must delete and recreate the service.

Explanation:

Anytime a data structure is changed in the service interface definition through the z/OS Connect EE API toolkit, you must delete and re-create the service.

Programmer response:

Drop and re-create the service to cause a new matching interface to be created.

GMOMW0016E A minimum of {0} entries for the composite array {1} of service interface {2} is expected, but only {3} entries were found.

Explanation:

The minimum number of entries defined for a composite field is not met.

Programmer response

Use the HTTP method GET to call the administration interface to get the correct data schema:

```
/zosConnect/services/{serviceName}/schema/  
request  
/zosConnect/services/{serviceName}/schema/  
response
```

GMOMW0017E A minimum of {0} entries for the leaf array {1} of service interface

{2} is expected, but only {3} entries were found.

Explanation:

The minimum number of entries defined for a leaf field is not met.

Programmer response

Use the HTTP method GET to call the administration interface to get the correct data schema:

```
/zosConnect/services/{serviceName}/schema/  
request  
/zosConnect/services/{serviceName}/schema/  
response
```

GMOMW0018E Field {0} that redefines field {1} could not be processed because redefines group member {2} has already been processed.

Explanation:

Another REDEFINES group member has been processed.

Programmer response:

Correct the JSON to specify at most one redefining field per redefined item. Use different services, one for each REDEFINES group, to ensure the correct values in correct data types are passed through.

GMOMW0019E Expected key {0} in JSON payload. Found key(s) {1}.

Explanation:

The key that is expected based on the service schema is not found. Another key or a different set of keys are found instead.

Programmer response:

Check the schema for the service against the client implementation and correct the problem.

GMOMW0027E Cannot convert value \"{0}\" from numeric field \"{1}\" using host date pattern \"{2}\" because the value is negative.

Explanation:

Negative values are not supported for numeric fields that are overridden as type Date.

Programmer response:

Ensure that the Date value sent to the service is a positive number.

GMOx messages - IMS service provider (IMS service provider)

Error messages for the IMS service provider have the prefix GMO.

GMOBA messages

GMOBA messages are related to communications with IMS Connect through the built-in backend adapter.

GMOBA0108E	The IMS gateway server is unable to establish a connection with IMS Connect: Failed to connect to host [host_name], port [port_number]. [java_exception]
-------------------	---

Explanation:

A connection with IMS Connect could not be established, most likely due to incorrect connection property specification, a network issue, or the unavailability of the host system. *java_exception* indicates the reason for the failure to connect.

System action:

An HTTP 500 status code is returned.

Programmer response

Examine the *java_exception* to determine the reason for the failure to connect to the host. Some possible values for *java_exception* are described in the following table.

Table 105. Java exceptions for GMOBA0108E
(continued)

Exception	Description
	<p>use the IMS Connect OPENPORT dddd command, where <i>dddd</i> is the specified port number.</p> <ul style="list-style-type: none">IMS Connect on the specified host is not running. Start IMS Connect on the host system.TCP/IP was restarted without either canceling and restarting IMS Connect, or issuing STOPPORT followed by OPENPORT on the host.

Table 105. Java exceptions for GMOBA0108E

Exception	Description
java.net.UnknownHostException: <i>hostname</i>	The host name that you specified when you configured the connection profile is invalid. Check that the configuration is still valid. You might need to use the fully-qualified path for the host name or the IP address.
java.net.ConnectException: Connection refused	Some possible reasons for the exception are: <ul style="list-style-type: none">The port number is invalid. Ensure that you are using a valid port number for the IMS Connect that is indicated by <i>host_name</i>.The specified port is stopped. Check the status of the port by using the IMS Connect VIEWHWS command. If the port status is NOT ACTIVE, the port is stopped. To start the port,

Table 105. Java exceptions for GMOBA0108E
(continued)

Exception	Description
java.net.SocketException: connect (code=10051)	<p>Some possible reasons for the exception are:</p> <ul style="list-style-type: none"> • The system with the specified host name is unreachable on the Internet Protocol network. Verify that the host system is accessible from the Internet Protocol network by issuing the ping command to the specified host system. Enter the ping command on the system on which the z/OS Connect EE server is running. Start TCP/IP on the host, if it is not started. • TCP/IP was restarted but the status of the port that is used by the application was NOT ACTIVE. To correct this situation, take one of the following actions: <ul style="list-style-type: none"> – Use the IMS Connect command OPENPORT dddd, where <i>dddd</i> is the port number, to activate the port. – Restart IMS Connect.

GMOBA0109E	The interaction requests could not be processed. <i>Error_details</i>.
------------	---

Explanation:

An error occurred when the interaction requests were being processed by IMS Connect or OTMA.

System action:

An HTTP 500 status code is returned.

Programmer response

Examine the IMS Connect return and reason codes and the OTMA sense codes to determine the appropriate corrective action.

The *return_code*, *reason_code*, and *sense_code* are reported in decimal. For example, if you receive an OTMA sense code of 27, the hexadecimal value is 1B, or 001B. Use the hexadecimal value when you look up the IMS Connect and OTMA return code, reason code, and sense code in IMS messages and code information.

For OTMA, a reason code of 0 indicates that the sense code does not have an associated reason code.

Related information

[IMS Connect return and reason codes](#)
[OTMA codes](#)

GMOBA0110E	The transaction could not be executed. <i>[server_error_detail]</i>.
-------------------	---

Explanation:

An error occurred when the requested transaction was executed on IMS Connect.

System action:

An HTTP 500 status code is returned.

Programmer response

Examine the *server_error_detail*. Traffic from the z/OS Connect EE server to IMS has a GMP prefix for the client ID. Use the client ID to help analyze the trace data from the different components that are involved. If a Java exception are reported, use the information to determine the reason for the failure. Some possible values for the Java exception are described in the following table.

Table 106. Java exceptions for GMOBA0110E

Java exceptions	Description
java.io.IOException	<p>Some possible reasons for the exception are:</p> <ul style="list-style-type: none"> The timeout value that is specified in the IMS Connect configuration member is exceeded before IMS Connect receives a response from IMS. Exceeding a timeout value typically occurs when a region is not available in IMS to run the IMS transaction that processes the client request. If so, ensure that an appropriate region is started and available to process the request. Exceeding a timeout value can also occur if the IMS application program that is associated with the transaction is stopped. If so, use the IMS type-2 UPDATE PGM NAME(pgm_name) START(SCHD) command or type-1 /START PROGRAM command to start the IMS application program. A Java client tries to use a previously active client (for example, a connection from the pool) for which an IMS Connect STOPCLNT command has been issued.
java.net.SocketException	<p>Some possible reasons for the exception are:</p>

Table 106. Java exceptions for GMOBA0110E (continued)

Java exceptions	Description
Connection reset by peer: socket write error	<ul style="list-style-type: none"> A Java client attempts to use a connection for which the underlying socket is no longer connected to IMS Connect. The socket connection might be lost if IMS Connect is recycled, but the application server is not. After IMS Connect is restarted, the connections that were formerly successfully connected to IMS Connect are still in the connection pool. As clients attempt to reuse each of these connections, the exception <code>java.net.SocketException</code> is thrown, and the connection object is removed from the connection pool. TCP/IP on the host is coming down.

GMOBA0113E Unable to instantiate a ConnectionFactory instance *instance_name*.

Explanation:

The IMS service provider cannot create a connection to connect to IMS. The connection profile might be corrupted.

System action:

An HTTP 500 status code is returned.

Programmer response:

Recreate the connection profile.

GMOCR messages

GMOCR messages are related to connection resources.

GMOCR0004E The specified connection profile name *conn_profile_name* cannot be located in the registry.

Explanation:

Most likely the specified connection profile name is not entered correctly. Profile names are case-sensitive. Allowed characters are A-Z, a-z, 0-9, and "-" (hyphen), ":" (period), "_" (underscore), and "~" (tilde).

System action:

An HTTP 404 status code is returned.

Programmer response:

Check that the connection profile name is specified correctly.

GMOCR0008E A connection named *conn_profile_name* already exists in the connection registry.

Explanation:

The specified connection profile name already exists in the registry.

System action:

No action was taken by the system. An HTTP 409 status code is returned.

Programmer response:

Specify a different connection name.

GMOCR0021E	The creation or change to the {0} configuration element {1} was not processed by the server.	Explanation: The configuration element might be connection, service, or interaction properties related. The server was not able to process the configuration changes in a reasonable amount of time. The server might be busy due to heavy workload. System action: An HTTP 500 status code is returned. Programmer response: Stop and start the server to process the configuration changes.	Explanation: This is an informational message. System action: An HTTP 200 status code is returned. Programmer response: No action is required.
GMOCR0201I	The connection profile with name <i>conn_profile_name</i> was successfully created.	Explanation: This is an informational message. System action: An HTTP 201 status code is returned. Programmer response: No action is required.	Explanation: This is an informational message. System action: An HTTP 200 status code is returned. Programmer response: No action is required.
GMOCR0202I	The connection profile with name <i>conn_profile_name</i> was successfully deleted.	Explanation: This is an informational message. System action: An HTTP 200 status code is returned. Programmer response: No action is required.	Explanation: This is an informational response to queries for available connection profiles in the registry. System action: An HTTP 404 status code is returned. Programmer response: No action is required.

GMOIG messages

GMOIG messages are general server communication-related messages.

GMOIG0004E	The entity <i>entity_name</i> was not found in the <i>entity_type</i> registry.	Explanation: The specified entity name does not exist in the registry for the identified entity type. System action: An HTTP 404 error is returned. Programmer response: Specify a valid name and try again.	This error message was returned only when all likely error cases were exhausted. System action: An HTTP 500 status code is returned. Programmer response: Check the trace output for possible causes. Restart the server. If the problem persists, contact IBM Software Support and provide the trace output.
GMOIG0008E	The <i>entity_name</i> name was empty.	Explanation: The required entity name was not specified. System action: No action was taken by the system. An HTTP 400 status code is returned. Programmer response: Specify a name and resubmit the request.	Explanation: GMOIG0101E The API call is not compatible with the IMS service feature that runs on the gateway server. Version <i>version_number</i> was specified for resource <i>resource_name</i>, but the valid version is (1.1). System action: An HTTP 412 status code is returned.
GMOIG0100E	An internal server error occurred.	Explanation:	

Programmer response:**Programmer response:**

Check and verify the required version of IMS Explorer for Development.

GMOIG0102E The data store could not be initialized.**Explanation:**

The data store might be corrupted or the data store location was modified.

System action:

An HTTP 500 status code is returned.

Programmer response:

The data store might be corrupted or the data store location was modified.

GMOIG0103E An error occurred during registry initialization. Error details: *details*.**Explanation:**

The registry for storing service information could not be initialized.

System action:

An HTTP 500 status code is returned.

Programmer response:

Examine the error details to determine the appropriate corrective action.

GMOIG0108I The *resource_type* in the cache was successfully logged in *length* milliseconds.**Explanation:**

This is an informational message that information stored in the cache for the *resource_type* is logged. The resource type might be connection profiles, interaction properties profiles, services, or transaction messages.

System action:

An HTTP 200 status code is returned.

Programmer response:

No action is required.

GMOIG0109I All resource types in the cache were successfully logged in *length* milliseconds.**Explanation:**

This is an informational message that information stored in the cache for all resource types is logged.

System action:

An HTTP 200 status code is returned.

Programmer response:

No action is required.

GMOIG0110I The *resource_type* in the cache was successfully cleared in *length* milliseconds.**Explanation:**

This is an informational message that information stored in the cache for the *resource_type* is cleared. The resource type might be connection profiles, interaction properties profiles, services, or transaction messages.

System action:

An HTTP 200 status code is returned.

Programmer response:

No action is required.

GMOIG0111I All resource types in the cache were successfully cleared in *length* milliseconds.**Explanation:**

This is an informational message that information stored in the cache for all resource types is cleared.

System action:

An HTTP 200 status code is returned.

Programmer response:

No action is required.

GMOIG0112E Deletion of the transaction directory {0} failed.**Explanation:**

The operation to delete a transaction failed, most likely due to a permission issue.

System action:

An HTTP 500 status code is returned.

Programmer response:

Ensure that the server administrator has the permission to the transaction directory in the service registry.

GMOIG0113E Deletion of the transaction message file {0} failed.**Explanation:**

The operation to delete a transaction message file failed, most likely due to a permission issue.

System action:

An HTTP 500 status code is returned.

Programmer response:

Ensure that the server administrator has the permission to the transaction messages files in the service registry.

GMOIG0114E Input payload is missing.**Explanation**

The JSON payload is missing. An outmost, top-level element that matches level 01 of the COBOL data

structure is required. For example, if the input copybook has the following structure:

```
01 IVTNO-INPUT-MSG.  
  02 IN-LL          PICTURE S9(3) COMP.  
  02 IN-ZZ          PICTURE S9(3) COMP.  
  02 IN-TRANCDE    PICTURE X(10).  
  . . .
```

The outmost element for the input message must be IVTNO-INPUT-MSG:

```
{  
  "IVTNO-INPUT-MSG": {  
    "IN_TRANCDE": "IVTNO",  
    . . .  
  }  
}
```

System action:

An HTTP 400 status code is returned.

Programmer response:

Ensure that the input message has an outmost element that matches level 01 of the COBOL input data structure and the XML is well-formed.

GMOIG0115E Unsupported media type.

Explanation:

The request is attempting to invoke a service by using an unsupported media type. The support media type is application/json.

System action:

An HTTP 415 status code is returned.

GMOIG0300W Information in the cache cannot be logged. Logging must be enabled first by adding the com.ibm.ims.gateway package to the logging element in server.xml.

Explanation:

Logging must be enabled in order to dump the information in the cache to the log.

System action:

An HTTP 412 status code is returned. No action was taken by the system.

Programmer response:

Enable logging and tracking by adding the com.ibm.ims.* package to the server.xml file. Specify the trace file name, maximum file size, maximum number of files, trace format, and trace level.

Related information

["Enabling trace in z/OS Connect EE server" on page 715](#)

Learn how to enable the z/OS Connect EE server trace when you are asked to do so by IBM service.

GMOIG7777I IMS service provider (*build_timestamp*) for z/OS Connect Enterprise Edition initialized successfully.

Explanation:

This is an informational message that indicates the server is up and running with the IMS service provider properly initialized.

Programmer response:

No action is required.

GMOIG7786I The IMS Mobile feature released *number of connections* connections to IMS.

Explanation:

This is an informational message.

System action:

An HTTP 404 not found status code is returned.

Programmer response:

No action is required.

GMOIG7787I The IMS Mobile feature was deactivated successfully.

Explanation:

This is an informational message.

Programmer response:

No action is required.

GMOIG7788I IMS Mobile service registry was successfully migrated. Total numbers migrated: {0} interaction properties, {1} connections , {2} transaction messages, {3} services.

Explanation:

This is an informational message to report the completion of service migration as a result of server restart after IMS service provider was updated.

Programmer response:

No action is required.

GMOIG7797I The ping request to the z/OS Connect EE server through the IMS service provider was successful.

Explanation:

This is an informational message that indicates the IMS service provider is configured properly on the z/OS Connect EE server.

Programmer response:

No action is required.

GMOIG7798I The ping request to the IMS system "*IMS_connection_detail*" through z/OS Connect EE and the

IMS service provider was successful.

Explanation:

This is an informational message that indicates the IMS service provider is configured properly on the z/OS Connect EE server, and the IMS host system is configured properly for communication with the IMS service provider.

Programmer response:

No action is required.

GMOIG7799E	The ping request to the IMS system "<i>IMS_connection_detail</i>" through z/OS Connect EE and the IMS service provider did not succeed.
-------------------	--

Explanation:

Most likely the IMS system is not yet set up to process the request from the IMS service provider.

System action:

GMOPR messages

GMOPR messages are related to properties resources.

GMOPR0002E	A duplicate profile named <i>profile_name</i> was found in the properties registry.
-------------------	--

Explanation:

The request cannot be executed because a profile of the same name already exists.

System action:

The request cannot be executed. An HTTP 409 error is returned.

Programmer response:

Specify a different profile name.

GMOPR0004E	No property with the profile name of <i>name</i> was found in the registry.
-------------------	--

Explanation:

The specified profile name was not found in the registry.

System action:

An HTTP 404 error is returned.

Programmer response:

Ensure that the profile name is specified correctly. Profile names are case-sensitive.

GMOPR0020E	The profile name is invalid.
-------------------	-------------------------------------

Explanation:

Profile names are case-sensitive. Allowed characters are A-Z, a-z, 0-9, and "-", ".", "_", and "~".

System action:

An HTTP 400 bad request error is returned.

An HTTP 500 status code is returned.

Programmer response:

Ensure that you have the permission to write to the database.

GMOIG8888E	The service <i>service_name</i> could not be updated in the registry.
-------------------	--

Explanation:

The specified service could not be updated, most likely due to a permission issue.

System action:

An HTTP 500 status code is returned.

Programmer response:

Ensure that IMS and IMS Connect is configured properly and up and running to process requests. If RACF is turned on in IMS Connect, the user ID for each request must be configured in RACF on the IMS system. For more information about required configuration, see [“Security configuration for the IMS service provider” on page 250](#).

Programmer response:

Ensure that the profile name is specified correctly.

GMOPR0201I	The <i>property_name</i> property was successfully created
-------------------	---

Explanation:

This is an informational message.

System action:

An HTTP 201 status code is returned.

Programmer response:

No action is required.

GMOPR0202I	The <i>property_name</i> property was successfully updated.
-------------------	--

Explanation:

This is an informational message.

System action:

An HTTP 200 status code is returned.

Programmer response:

No action is required.

GMOPR0203I	The <i>property_name</i> property was successfully deleted.
-------------------	--

Explanation:

This is an informational message.

System action:

An HTTP 200 status code is returned.

Programmer response:

No action is required.

GMOPR0213I	No property profiles exist in the registry.	System action: An HTTP 404 status code is returned.
Explanation: This is an informational response to queries for available property profiles in the registry.	Programmer response: No action is required.	
<h2>GMOSR messages</h2>		
	GMOSR messages are related to service resources.	
GMOSR0002E	Duplicate services were found in the registry with service name <i>name</i>.	No system action was taken. An HTTP 412 status code is returned.
Explanation: The specified service name already exists.	Programmer response: Update the service definition and try again.	
System action: No system action was taken. An HTTP 409 status code is returned.	GMOSR0012E	The service <i>service_name</i> could not be invoked because it is not started.
Programmer response: Specify a different service name.	Explanation: A service must be started before it can be invoked.	
GMOSR0004E	The <i>service_name</i> service was not found in the registry.	System action: An HTTP 412 status code (precondition failed) is returned.
Explanation: The specified service name does not exist in the registry.	Programmer response: Start the service by issuing the start action first before invoking the service.	
System action: No system action was taken. An HTTP 404 status code is returned.	GMOSR0013E	An unexpected error occurred processing service <i>service_name</i> using transcode <i>transaction_code</i>. Originating exception message = <i>exception_message</i>
Programmer response: Specify a valid service name.	Explanation: An exception occurred in a service for using a specific transcode.	
GMOSR0005E	The service could not be created because the referenced <i>resource_type</i> named <i>resource_name</i> was not found in the registry.	System action: An originating exception message is returned.
Explanation: The specified name for the identified resource type is not valid.	Programmer response: Refer to the error messages accompanying the exception for more information.	
System action: No system action was taken. An HTTP 404 status code is returned.	GMOSR0201I	Service <i>service_name</i> was successfully created.
Programmer response: Specify a valid name for the identified resource type.	Explanation: This is an informational message.	
GMOSR0011E	The service type is not supported. Only service with TRAN service type is supported. Update the service definition and try again.	System action: The named service was created, and an HTTP 201 status code is returned.
Explanation: A value other than TRAN for the service type was specified.	Programmer response: No action is required.	
System action:	GMOSR0202I	Service <i>service_name</i> was successfully updated.
	Explanation:	

This is an informational message.

System action:

An HTTP 200 status code is returned.

Programmer response:

No action is required.

GMOSR0203I	Service <i>service_name</i> was successfully deleted.
-------------------	--

Explanation:

This is an informational message.

System action:

An HTTP 200 status code is returned.

Programmer response:

No action is required.

GMOSR0204I	Service <i>service_name</i> was successfully started.
-------------------	--

Explanation:

This is an informational message.

System action:

An HTTP 200 status code is returned.

GMOTM messages

GMOTM messages are related to transaction management.

GMOTM0001E	An error occurred while accessing the transaction message table.
-------------------	---

Explanation:

The transaction message information in the registry is not accessible.

System action:

An HTTP 500 status code is returned.

Programmer response:

Check the trace for possible causes. Restart the server, check that the connection is properly configured, and retry the operation. If the problem persists, contact IBM Software Support and provide the trace output.

GMOTM0002E	A transaction with transaction code <i>tran_code</i> and message name <i>message_id</i> already exists in the registry.
-------------------	--

Explanation:

An entity for the identified transaction could not be created in the registry because it already exists.

System action:

An HTTP 409 status code is returned.

Programmer response:

Check the trace for possible causes. Restart the server, check that the connection is properly configured, and retry the operation. If the problem persists, contact IBM Software Support and provide the trace output.

Programmer response:

No action is required.

GMOSR0205I	Service <i>service_name</i> was successfully stopped.
-------------------	--

Explanation:

This is an informational message.

System action:

An HTTP 200 status code is returned.

Programmer response:

No action is required.

GMOSR0213I	No services exist in the registry.
-------------------	---

Explanation:

This is an informational response to queries for available services in the registry.

System action:

An HTTP 404 not found status code is returned.

Programmer response:

No action is required.

GMOTM0004E	The transaction code <i>tran_code</i> with message name <i>message_id</i> was not found.
-------------------	---

Explanation:

The transaction with the identified transaction code and message name does not exist in the registry.

System action:

An HTTP 404 status code is returned.

Programmer response:

Check the trace for possible causes. Restart the server, check that the connection is properly configured, and retry the operation. If the problem persists, contact IBM Software Support and provide the trace output.

GMOTM0011E	More than one transaction message with transaction code <i>tran_code</i> and message name <i>message_id</i> exists.
-------------------	--

Explanation:

The transaction with the identified transaction code and message name already exists in the registry.

System action:

An HTTP 409 status code is returned.

Programmer response:

Check the trace for possible causes. Restart the server, check that the connection is properly configured, and

retry the operation. If the problem persists, contact IBM Software Support and provide the trace output.

GMOTM0015E **No messages exist with transaction code *tran_code* and direction *message_direction*.**

Explanation:

The transaction with the identified transaction code in the input (*message_direction*=0) or output (*message_direction*=1) message does not exist.

System action:

An HTTP 404 not found status code is returned.

Programmer response:

Check that the transaction code is specified correctly in the interaction properties.

GMOTM0021E **The transaction code *tran_code* was not valid.**

Explanation:

The transaction code cannot be null, empty, or longer than eight characters.

System action:

An HTTP 400 status code is returned.

Programmer response:

Specify a valid transaction code. A valid transaction code contains no more than eight characters.

GMOTM0022E **The message name *message_id* was not valid.**

Explanation:

The message name cannot be null or empty.

System action:

An HTTP 400 status code is returned.

Programmer response:

Specify a valid message name.

GMOTM0023E **The direction *direction* was not valid.**

Explanation:

The identified direction is not valid for the transaction.

System action:

An HTTP 400 status code is returned.

Programmer response:

Specify a valid direction. Valid directions are INPUT or OUTPUT.

GMOTM0201I **The transaction message with transaction code *tran_code* and**

message name *message_name* was successfully created.

System action:

An HTTP 201 status code is returned.

Programmer response:

No action is required.

GMOTM0203I **The transaction message with transaction code *tran_code* and message name *message_name* was successfully deleted.**

System action:

An HTTP 200 status code is returned.

Programmer response:

No action is required.

GMOTM0213I **No transaction messages exist in the registry.**

Explanation:

This message is a response to queries for available transaction messages in the service registry.

System action:

An HTTP 404 not found status code is returned.

Programmer response:

No action is required.

GMOTM0214I **No transaction codes exist in the registry.**

Explanation:

This message is a response to queries for available transactions in the service registry.

System action:

An HTTP 404 not found status code is returned.

Programmer response:

No action is required.

GMOTM0221I **The transaction message with transaction code *tran_code* and message name *message_name* was successfully updated.**

System action:

An HTTP 200 status code is returned.

Programmer response:

No action is required.

Chapter 28. Reference

This section contains descriptions of the `server.xml` elements, Javadoc for the z/OS Connect EE SPI, and other samples and reference information. Use this information in your development or to extend z/OS Connect EE.

Capabilities compatibility

When you deploy an API or service, a compatibility check is made to ensure that the capabilities of the z/OS Connect EE API toolkit and z/OS Connect EE server are compatible. When capabilities are not compatible, z/OS Connect EE runtime messages **BAQR7086E** or **BAQR7085E** are issued.

The following list contains the minimum versions of the z/OS Connect EE API toolkit and z/OS Connect EE server runtime required for a given capability. Use this list to determine the level of support for capabilities you are using.

Capability	API Toolkit version	Server runtime version	Description
ibm.apis.multipleResponseCodes	V3.0.6	V3.0.16	Supports the definition of multiple response codes for a service based on conditional processing.
ibm.services.wbservice.arrayCounter	V3.0.6.7	V3.0.23 + APAR PH15332 See note below.	Supports the definition of array counters for service fields.
ibm.services.wbservice.overrideBoolean	V3.0.6.10	V3.0.26	Supports the definition of service fields as a Boolean data type.
ibm.services.wbservice.overrideDate	V3.0.8.3	V3.0.34	Supports date type service field overrides in the API toolkit.
ibm.apis.formatDate	V3.0.8.3	V3.0.34	Allows services to be imported into APIs using the properties <code>type:string</code> , <code>format:date</code> .
ibm.apis.contactInfo	V3.0.8.3	V3.0.34	Supports the specification of API contact information by using the API editor.
ibm.services.wbservice.redefineAsChar	V3.0.8.3	V3.0.34	Supports the Redefine as CHAR function, allowing STRUCT field groups to be represented as characters.

Capability	API Toolkit version	Server runtime version	Description
ibm.services.wvservice.importedImsLdsStartsWithLLZZTrancodeFields	V3.0.8.7	V3.0.38	<p>Supports the Imported IMS large data structure starts with LLZZTRANCODE fields advanced service data conversion option.</p> <p>This option allows for the tolerance of imported request IMS large data structures with leading LLL, ZZ, and TRANCODE fields. These fields are used when creating message segments to send to IMS.</p>
ibm.services.wvservice.outputImsLdsStartsWithLLZZFields	V3.0.8.7	V3.0.38	<p>Supports the Imported IMS large data structure starts with LLZZ fields advanced service data conversion option.</p> <p>This option allows for the tolerance of imported response IMS large data structures with leading LL or ZZ fields, with response data starting at the fifth byte of the DATA portion of the first message segment received from IMS.</p>

Note: Do not deploy services using the **ibm.services.wvservice.arrayCounter** capability in production until you have the fix for PH15332 applied. If you have either version V3.0.6.7 or V3.2.6.7 or later of the API toolkit, you can avoid using this capability when editing an Array field in the service interface editor, by not using the Array Properties section at the bottom. This section includes labels 'Counted by:' and 'Minimum items'.

z/OS Connect EE features

Use this section to find the latest levels of the z/OS Connect EE features

Features are the units of functionality by which you control the pieces of the runtime environment that are loaded into a z/OS Connect EE server. The latest feature levels are listed below:

```

<feature>imsmobile:imsmobile-2.0</feature>
<feature>zosconnect:dbService-1.0</feature>
<feature>zosconnect:cicsService-1.0</feature>
<feature>zosconnect:zosConnect-2.0</feature>
<feature>zosconnect:zosConnectCommands-1.0</feature>
<feature>zosconnect:apiRequester-1.0</feature>
<feature>zosconnect:mqService-1.0</feature>
```

Configuration elements

You can use the following elements in your `server.xml` file to configure z/OS Connect EE.

This list contains only those elements that are unique to z/OS Connect EE. For more information about Liberty configuration elements not listed here, see [Server configuration](#) in the *IBM WebSphere Application Server for z/OS Liberty* documentation.

Each server must have a server configuration file called `server.xml` in its server configuration directory `${server.config.dir}` . You can choose to keep all your configuration in the single `server.xml` file, or you can use include elements to consolidate configurations from separate files to create the structure that is most useful to you. For more information, see [Using include elements in configuration files](#) in the *IBM WebSphere Application Server for z/OS* documentation.

Care is needed to avoid defining multiple instances of the singleton elements, or elements with the same ID value, by understanding the rules that are used to merge these elements. For more details on the rules used to merge the multiple instances of the elements see [Configuration element merging rules](#) in the *IBM WebSphere Application Server for z/OS* documentation.

- “[zosconnect_apiRequesters](#)” on page 754
 - [apiRequester](#)
- [zosconnect_auditInterceptor](#)
- “[zosconnect_authorizationServer](#)” on page 758
- [zosconnect_authData](#)
- [zosconnect_authorizationInterceptor](#)
- “[zosconnect_authToken](#)” on page 760
 - [zosconnect_authToken>tokenRequest](#)
 - [zosconnect_authToken>tokenResponse](#)
- “[zosconnect_authTokenLocal](#)” on page 762
 - [claims](#)
- “[zosconnect_dbServices](#)” on page 765
- “[zosconnect_endpointConnection](#)” on page 765
- [zosconnect_cicsIpicConnection](#)
- “[zosconnect_localAdaptersConnectService](#)” on page 767
- “[zosconnect_mqService](#)” on page 769
- “[zosconnect_oAuthConfig](#)” on page 772
- “[zosconnect_policy](#)” on page 773
- “[zosconnect_proxyConfig](#)” on page 774
- “[zosconnect_services](#)” on page 774
 - [service](#)
 - [property](#)
- [zosconnect_zosConnectAPIs](#)
 - [zosConnectAPI](#)
- [zosconnect_zosConnectDataXform](#)
- [zosconnect_fileSystemloggerInterceptor](#)
- [zosconnect_zosConnectInterceptors](#)
- [zosconnect_zosConnectManager](#)
- [zosconnect_zosConnectService](#)
 - [interceptorsRef](#)

- [zosconnect_zosConnectServiceRestClient](#)
 - [basicAuth](#)
- [zosconnect_zosConnectServiceRestClientConnection](#)
 - [basicAuth](#)
- [zosconnect_zosConnectServiceRestClientBasicAuth](#)
- “[zosLocalAdapters \(WOLA\)](#)” on page 792

zosconnect_apiRequesters

Defines the directory where API requester archives are stored and how the server is notified about changes in this directory. It also defines additional configuration that applies to all API requesters.

Attribute name	Data type	Default value	Description
idAssertion	string	OFF	<p>Controls whether identity assertion is enabled in z/OS Connect EE for API requesters, and whether a surrogate check is supported if identity assertion is enabled. Acceptable values are OFF, ASSERT_SURROGATE, ASSERT_ONLY.</p> <p>OFF</p> <p>Identity assertion is disabled in z/OS Connect EE for API requesters. z/OS applications cannot invoke an API requester with an asserted identity that is provided in the application context.</p> <p>ASSERT_SURROGATE</p> <p>Identity assertion is enabled in z/OS Connect EE for API requesters. z/OS applications can invoke an API requester with an asserted identity that is provided in the application context. With the ASSERT_SURROGATE value specified, z/OS Connect EE checks whether the identity used for authenticating the z/OS subsystem access to the z/OS Connect EE server is a surrogate of the asserted identity and whether the asserted identity has the authority to invoke the API requester.</p> <p>ASSERT_ONLY</p> <p>Identity assertion is enabled in z/OS Connect EE for API requesters. z/OS applications can invoke an API requester with an asserted identity that is provided in the application context. With the ASSERT_ONLY value specified, z/OS Connect EE directly checks whether the asserted identity has the authority to invoke the API requester.</p> <p>Depending on the values set to the idAssertion and requireAuth attributes, z/OS Connect EE performs different actions on the asserted identity and the identity used for authenticating the z/OS subsystem access to the z/OS Connect EE server. For more information, see “Identity assertion for API requesters” on page 441.</p>

Attribute name	Data type	Default value	Description
location	string	\$ {server.config.dir }/ resources / zosconnect/ apiRequesters	Path to a directory location where API requester archive is stored. The value of location cannot be changed while the server is running; the value is set when the server is started.
pollingRate	long (A period of time with millisecond precision)	5s	Controls how often the server polls the <code>apiRequesters</code> directory. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500 ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.
requireAuth	boolean		This is available from V3.0.29.0. Require that users specify security credentials to be authenticated and that the authenticated user is authorized under the <code>zosConnectAccess</code> role, in order to access all API requesters. If the <code>requireAuth</code> attribute is not set, the global setting from the <code>requireAuth</code> attribute on the <code>zosconnect_zosConnectManager</code> element is used instead, unless the <code>requireAuth</code> attribute is overridden on the specific API requester.
updateTrigger	string	disabled	<p>Controls when the runtime is notified about changes in the <code>apiRequesters</code> directory. Acceptable values are <code>disabled</code>, <code>polled</code>, or <code>mbean</code>.</p> <p>disabled Polling for updates is disabled. Updates can be triggered by using the <code>MODIFY refresh</code> command, and the RESTful administration interface can be used to deploy API requester archive files.</p> <p>polled The server checks periodically for changes to the directory contents.</p> <p>mbean The server will check for changes when the <code>notifyFileChanges</code> method is invoked on the <code>FileNotificationMBean</code>. If you specify this value, you must also configure your server to use the Java Management Extensions (JMX) connector. For more information, see “Using an MBean to trigger updates” on page 301.</p> <p>The value of this attribute is ignored when the MODIFY command is used to refresh the z/OS Connect EE server artifacts.</p>

Sub elements

`zosconnect_apiRequesters > apiRequester`

Description: Defines additional configuration for the API requester archive.

Required: false

Attribute name	Data type	Default value	Description
name	string		A name of the API requester. The API requester is named automatically when the API requester archive file is generated. The name of the API requester indicates the name and version of the API to be called, and is associated with the API requester archive file to be used. For more information, see Configuring z/OS Connect EE to support API requesters.
requireSecure	boolean		Require that requests are sent over HTTPS. If the requireSecure attribute is not set, the global setting from the requireSecure attribute on the <code>zosconnect_zosConnectManager</code> element is used instead.
requireAuth	boolean		Requires that users specify security credentials to be authenticated and that the authenticated user is authorized under the <code>zosConnectAccess</code> role, to access the specific API requester. If the requireAuth attribute is not set, the setting for all API requesters from the requireAuth attribute on the <code>zosconnect_apiRequesters</code> element is used instead. If the requireAuth attribute on the <code>zosconnect_apiRequesters</code> element is not set either, the global setting from the requireAuth attribute on the <code>zosconnect_zosConnectManager</code> element is used instead.
runGlobalInterceptors	boolean	true	Indicates whether global interceptors should run for requests that are associated with this API requester. Global interceptors are listed in <code>globalInterceptorsRef</code> in the <code>zosconnect_zosConnectManager</code> element. By default, z/OS Connect EE processes all global and API requester-specific interceptors. If the runGlobalInterceptors attribute is set to false, z/OS Connect EE processes only the set of interceptors that are listed in the <code>interceptorsRef</code> attribute.
interceptorsRef	string		Reference name that identifies the set of configured interceptors that are associated with this API requester.
connectionRef	string		A reference to top level <code>zosconnect_endpointConnection</code> element. If set, the connection is made using the attributes of the <code>zosconnect_endpointConnection</code> element; if not set, or the associated <code>zosconnect_endpointConnection</code> element does not exist, the value of the connectionRef attribute in the build toolkit properties file is used, and if the value of the connectionRef attribute in the build toolkit properties file does not exist either, an error occurs.
adminGroup	string		Identifies the users that are able to use administrative functions on this API requester. The value of this attribute can be set to a group name or a comma-separated list of group names that are defined in the user registry. If <code>globalAdminGroup</code> is also under element <code>zosconnect_zosConnectManager</code> , the value that is defined under <code>adminGroup</code> is used. See Note 1.
invokeGroup	string		Identifies the users that are able to invoke this API requester. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. If <code>globalInvokeGroup</code> is also defined under element <code>zosconnect_zosConnectManager</code> , the value that is defined under <code>invokeGroup</code> is used. See Note 1.

Attribute name	Data type	Default value	Description
operationsGroup	string		Identifies the users that are able to perform operations such as starting or stopping this API requester. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. If <code>globalOperationsGroup</code> is also defined under element <code>zosconnect_zosConnectManager</code> , the value that is defined under <code>operationsGroup</code> is used. See Note 1 .
readerGroup	string		Identifies the users that are able to get information about this API requester, including the Swagger documentation. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. If <code>globalReaderGroup</code> is also defined under element <code>zosconnect_zosConnectManager</code> , the value that is defined under <code>readerGroup</code> is used. See Note 1 .

Note 1: If you use an LDAP registry, you must specify each LDAP group's distinguished name (DN) with the commas escaped with a backslash. For example "cn=employees\,ou=groups\,o=intern\,c=fr, cn=managers\,ou=groups\,o=intern\,c=fr". If specifying multiple groups, the commas separating the groups are not escaped. Specifying LDAP short names is not supported.

`zosconnect_auditInterceptor`

Defines the audit interceptor for z/OS Connect EE to allow request data to be logged in System Management Facility (SMF) 123 subtype 1 records on z/OS.

Attribute name	Data type	Default value	Description
id	string		A unique configuration ID.
sequence	integer Minimum: 0 Maximum: 2147483647	0	The sequence in which this interceptor is processed compared to other configured interceptors that implement the <code>com.ibm.wsspi.zos.connect.Interceptor</code> Service Provider Interface (SPI) for z/OS Connect EE.
apiProviderSmfVersion	integer Minimum: 1 Maximum: 2	1	The version of the SMF type 123 subtype 1 records that you want this audit interceptor to capture.
apiProviderRequestHeaders	string		(SMF type 123 subtype 1 version 2 records only) The value of this attribute can be set to a header name or a comma-separated list of header names that might be present on requests.
apiProviderResponseHeaders	string		(SMF type 123 subtype 1 version 2 records only) The value of this attribute can be set to a header name or a comma-separated list of header names that might be present on responses as a result of response data mapping, see " Defining and mapping headers, query parameters, or path parameters " on page 581.

`zosconnect_authData`

Reference name that identifies the basic authentication data to be used for a connection. The credentials are overridden by any values that are supplied on a request.

Attribute name	Data type	Default value	Description
id	string		The element identifier.
password	string		The password that is passed from the z/OS Connect EE server to establish the connection on every request. The value can be in clear text or encoded by xor, aes or hash. Use the WebSphere Liberty profile server securityUtility command (securityUtility encode <password>) to generate an encoded password. Copy the encoded password into the <code>server.xml</code> file. The password can be a password phrase.
user	string		The user ID that is passed from the z/OS Connect EE server to establish the connection on every request, if no user ID is supplied on the request.

zosconnect_authorizationInterceptor

Defines a z/OS Connect EE authorization interceptor.

Attribute name	Data type	Default value	Description
id	string		A unique configuration ID.
safCacheTimeout	integer Minimum: -1 Maximum: 2147483647	600	Specifies the amount of time in seconds that SAF credentials are kept in the SAF cache that is used by the ID assertion feature. If the timeout value is set to 0, cache is disabled. If set to -1, SAF credentials are kept indefinitely. Specify -1 or a positive integer in the range 0 - 2147483647. The cache is cleared if the timeout value is changed and the configuration refreshed.
sequence	integer Minimum: 0 Maximum: 2147483647	0	The sequence in which this interceptor is processed compared to other configured interceptors implementing z/OS Connect's com.ibm.wsspi.zos.connect.Interceptor Service Provider Interface (SPI).

zosconnect_authorizationServer

Allows requests for access tokens or JWTs to be routed from z/OS Connect EE to an authorization server or an authentication server. For more information about supported security configuration options when using JWT or OAuth 2.0, see [“How to configure a third-party JWT” on page 429](#) or [“How to configure OAuth 2.0” on page 425](#).

Attribute name	Data type	Default value	Description
id	string		A unique configuration ID.

Attribute name	Data type	Default value	Description
tokenEndpoint	string		<p>Token endpoint URL that is used for routing a request to get an access token or a JWT from an authorization server or an authentication server. This URL must follow the following format:</p> <pre data-bbox="703 340 1013 371"><code>"https://host:port/path"</code></pre> <p>For example,</p> <pre data-bbox="703 456 1380 498"><code>tokenEndpoint="https://authorization.server.com:8001/JWTTokenGenerator/getJwtToken"</code></pre> <p>Contact the authorization/authentication server administrator for details of the path value required for that server.</p>
basicAuthRef	Reference to top level zosconnect_authData element (string)		<p>Reference name that identifies the basic authentication data to be used for authenticating with an authorization server. The values of the user and password attributes that are set in the associated zosconnect_authData element take precedence over user credentials that are specified in the z/OS application.</p> <p>When your z/OS application calls an API secured with OAuth 2.0</p> <p>The values of the user and password attributes set in the associated zosconnect_authData element are used as client ID and client secret to verify the client identity of the z/OS Connect EE server with an authorization server to obtain an access token.</p> <p>When your z/OS application calls an API secured with a JWT</p> <p>The values of the user and password attributes set in the associated zosconnect_authData element are used as username and password to verify the user identity with an authentication server to obtain a JWT.</p>
connectionTimeout	A period of time with millisecond precision	30s	<p>The connection timeout specifies the amount of time that the z/OS Connect EE server attempts to establish a connection to the authorization/authentication server before it times out. If the timeout value is set to 0, the z/OS Connect EE server attempts to open a connection indefinitely. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.</p>
receiveTimeout	A period of time with millisecond precision	60s	<p>The receive timeout specifies the amount of time that the z/OS Connect EE server waits for a response from the authorization/authentication server before it times out. If the timeout value is set to 0, the z/OS Connect EE server waits for a response indefinitely. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.</p>

Attribute name	Data type	Default value	Description
proxyConfigRef	A reference to top level zosconnect_proxyConfig element. (string)		Reference name that identifies the proxy through which the request for access token is routed from the z/OS Connect EE server to the authorization/authentication server.
sslCertsRef	string		An SSL repertoire with an ID, a defined keystore, and truststore.

zosconnect_authToken

Defines the configuration of the JWT that is obtained from the authentication server.

Attribute name	Data type	Default value	Description
id	string		A unique configuration ID.
authServerRef	A reference to the top level zosconnect_authorizationServer element. (string)		A reference name that identifies the information about an authentication server that is used for JWT authentication.
header	string	Authorization	Specify the name of the header that contains the JWT on the API request.

Sub elements

zosconnect_authToken > tokenRequest

Description: Defines how the user credential is passed from the z/OS Connect EE server to the authentication server.

Required: true

Attribute name	Data type	Default value	Description
credentialLocation	string		<p>Specify where the user credential is included in the request to obtain a JWT from the authentication server. The following values are supported:</p> <p>header Include the user credentials in the HTTP header. If this value is set, the <code>header</code> attribute of the <code>tokenRequest</code> element must be specified.</p> <p>body Include the user credentials in the request body. If this value is set, the <code>requestBody</code> attribute must be specified.</p> <p>For both values, the <code>RequestMethod</code> attribute must be specified.</p>
header	string	Authorization	Specify the name of the header that needs to contain the user credentials.

Attribute name	Data type	Default value	Description
requestBody	string		<p>Specify a JSON string that contains the information about user credentials when the user credentials are included in the request body. The JSON string must contain the key-value pair that explicitly indicates the username and password values like Example A or contain the key-value pair that uses the \$ character like Example B.</p> <p>Example A</p> <pre data-bbox="665 481 1274 593">"\"credentials\":{\"username\":\"jwtuser\",\"password\":\"jwtpassword\"}\""</pre> <p>In this example, the user credentials "jwtuser" and "jwtpassword" are directly included in the specified JSON string.</p> <p>Example B</p> <pre data-bbox="665 798 1286 842">"\"apiuser\":\"\${userid}\",\"apipassword\":\"\${password}\""</pre> <p>In this example, the variables \${userid} and \${password} are replaced with the user credentials that you include in the z/OS application or set on the <code>zosconnect_authData</code> element that is referenced by the <code>zosconnect_authorizationServer</code> element basicAuthRef attribute.</p> <p>Important:</p> <ul style="list-style-type: none"> Typically, you use the Example B syntax. When the Example B syntax is used with the user credentials set on the <code>zosconnect_authData</code> element, the password in the <code>server.xml</code> file can be encoded to ensure confidentiality. The Example A syntax is provided to allow more flexibility in the request payload that is required by the authentication server. As shown in the examples above, " must be used to escape the double quotation mark ("") inside the attribute value, because the attribute value is already surrounded by double quotation marks to indicate it is a string value. And the following characters must also be escaped if they are contained in the attribute value because these special characters cannot be directly used in XML: <ul style="list-style-type: none"> < escaped with &lt; > escaped with &gt; & escaped with &amp;
requestMethod	string		Specify the method of the HTTP request to the authentication server. Acceptable values are GET, PUT, or POST.

zosconnect_authToken > tokenResponse

Description: Defines how a JWT is passed from the authentication server to the z/OS Connect EE server.

Required: true

Attribute name	Data type	Default value	Description
header	string	Authorization	Specify the name of the header that needs to contain the JWT.
responseFormat	string		Specify the format of the HTTP response from the authentication server when the JWT is returned in the response body. Acceptable values are Text or JSON.
tokenLocation	string		<p>Specify where the generated JWT is returned in the response from the authentication server to the z/OS Connect EE server. The following values are supported:</p> <p>header The JWT is returned in a header to z/OS Connect EE. If this value is set, the <code>header</code> attribute of the <code>tokenResponse</code> element must be specified.</p> <p>body The JWT is returned in the response body to z/OS Connect EE. If this value is set, the <code>responseFormat</code> and <code>tokenPath</code> attributes must be specified.</p>
tokenPath	string		<p>Specify the path to where the token is located in the JSON string when the <code>responseFormat</code> attribute is set to JSON. The value of this attribute must be a valid JSONPath expression.</p> <p>For example, if the generated token is included in the following JSON string:</p> <pre>{ "JWTInfo": { "tokenname": "eyJ0eXAiOiJKV1" } }</pre> <p>you must set the <code>tokenPath</code> attribute to <code>".JWTInfo.tokenname"</code>.</p>

zosconnect_authTokenLocal

Defines the locally generated JWT configuration in z/OS Connect EE.

Attribute name	Data type	Default value	Description
header	string	Authorization	Specify the name of the HTTP header that contains the JWT on the API request. The HTTP header includes the "Bearer" scheme keyword followed by the JWT.
tokenGeneratorRef	A reference to a <code>jwtBuilder</code> element. (string)		The <code>id</code> attribute value of a JWT builder element. For more information about the <code>jwtBuilder</code> element, see JWT Builder (jwtBuilder) in the WebSphere Application Server for z/OS Liberty documentation .

Sub elements

zosconnect_authTokenLocal > claims

Description: Specify the public and private claims to be included in the JWT. If specified, write the claims as a JSON string. For example,

```
<zosconnect_authTokenLocal id="myLocalJWTConfig"
 ...
<claims>{"employees" :
[{"ID": "1234567890", "Name": "Tom"}, {"ID": "1234567891", "Name": "Jerry"}, {"ID": "1234567892", "Name": "Hash"}, {"BRANCH": "HR_PROG1", "ACTION": "QUERY_STATUS"}]</claims>
</zosconnect_authTokenLocal>
```

Note:

1. The claims subelement is intended to specify only public and private claims. If registered claims, such as the aud (Audience) claim, are specified on the claims subelement, then these values overwrite the corresponding values configured on the jwtBuilder element referenced by the tokenGeneratorRef attribute of the zosconnect_authTokenLocal element. If the "sub" claim is specified on the claims subelement, its value is overwritten by the z/OS Connect EE server to be the z/OS application asserted user ID. Registered claims are defined in the [IANA JSON Web Token Claims Registry](#).
2. If the JSON string value of the claims subelement contains XML markup characters, such as <, > and &, then include the JSON string inside a CDATA section so that those characters are treated as literals. For example, if one of the private claims above was "BRANCH": "<HR_PROG1>" then the claims subelement value must be specified as:

```
<claims><![CDATA[{"employees" :
[{"ID": "1234567890", "Name": "Tom"}, {"ID": "1234567891", "Name": "Jerry"}, {"ID": "1234567892", "Name": "Hash"}, {"BRANCH": "<HR_PROG1>", "ACTION": "QUERY_STATUS"}]]></claims>
```

For more information about the CDATA section, see [CDATA](#).

Required: false

Data type: a string or CDATA section

zosconnect_cicsIpicConnection

Represents a connection to a CICS system.

Note: When an IPIC connection is established with CICS, updates to the authDataRef, transid and transidUsage attributes take immediate effect but updates to other attributes that are associated with this element do not take effect until the connection is released and acquired again. To release the connection in CICS, change the status of the corresponding IPCONN in CICS to Released.

Attribute name	Data type	Default value	Description
authDataRef	string		Optional. A reference to a zosconnect_authData element that contains the basic authentication data to be used for the connection if no credentials are supplied on a request. For more information, see "zosconnect_authData" on page 757 .
cicsApplid	string		Optional. The APPLID of the target CICS region. If specified, the value of cicsApplid is used, together with the value of cicsNetworkId , to verify that the connected CICS region is the expected region.

Attribute name	Data type	Default value	Description
cicsNetworkid	string		Optional. The network ID of the target CICS region. The default value is 9UNKNOWN. If specified, the value of cicsNetworkid is used, together with the value of cicsApplid , to verify that the connected CICS region is the expected region. The network ID of the target CICS region is either its z/OS Communications Server NETID or, for VTAM®=NO systems, the value of its UOWNETQL system initialization parameter, or defaults to 9UNKNOWN.
connectionTimeout	A period of time with millisecond precision.	30s	Optional. The maximum amount of time that is allowed for the socket to establish a connection to CICS. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds. A value of 0 disables this timeout.
heartbeatInterval	A period of time with millisecond precision. Maximum: 3600s.	30s	Optional. This attribute sets the time that the connection must be inactive before heartbeats are sent to CICS. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), or seconds (s). For example, specify 30 seconds as 30s. You can include multiple values in a single entry. For example, 1m30s is equivalent to 90 seconds. A value of 0 disables IPIC heartbeats.
host	string		Required. The IP address, domain name server (DNS) hostname with domain name suffix, or just the DNS hostname, of the host on which the CICS region is running.
id	string		Required. A unique configuration ID. This must match the value that is specified for the connectionRef build toolkit property that is used to generate the .sar files that use this connection with the CICS service provider.
port	integer Minimum: 1 Maximum: 65,535		Required. The port number on which the target CICS region is listening. This must match the port number of a TCPIPSERVICE definition in the CICS region that is configured with the PROTOCOL parameter set to IPIC.
sendSessions	integer Minimum: 1 Maximum: 999	100	Optional. This attribute sets the maximum number of simultaneous requests over the connection. The actual number of send sessions established depends on the value of sendSessions and the value in the RECEIVECOUNT parameter of the IPCONN definition in the CICS region.
sharedPort	boolean	false	Optional. Indicates whether the port attribute specifies a shared port or a specific port.
sslCertsRef	string		Optional. Reference to an SSL repertoire with an ID, a defined keystore and truststore, or an SSL Default repertoire.
transid	string	CSMI	Optional. A CICS transaction name; the transidUsage parameter specifies how the value is used.

Attribute name	Data type	Default value	Description
transidUsage	<ul style="list-style-type: none"> • EIB_ON LY • EIB_AN D_MIRR OR 	EIB_AND_MIRROR	<p>Optional. Specifies how the value of the transid parameter is used.</p> <p>EIB_ONLY The transid parameter specifies the name of the CICS transaction that appears in the CICS exec interface block (EIB); the EIBTRNID field contains the value of the transid parameter. The called CICS program runs under the default mirror transaction CSMI.</p> <p>EIB_AND_MIRROR The transid parameter specifies the name of the CICS transaction under which the called CICS program runs. The transaction must be defined in the CICS region, and the transaction definition must specify the mirror program, DFHMIRS. The value specified by the transid parameter is available to the called CICS program for querying the transaction ID. The value of the transid parameter also appears in the EIBTRNID field of the CICS EIB.</p>
zosConnectApplid	string		<p>Optional. The APPLID of the z/OS Connect EE server passed to CICS. If specified, this value of zosConnectApplid is used, together with the value of zosConnectNetworkid, to match a predefined IPCONN definition in CICS or reject the request if no match is found and the CICS system has not been configured to autoinstall IPCONN connections.</p> <p>If you configure CICS to not allow autoinstall of IPCONN connections, only requests that have APPLIDs set on a predefined IPCONN definition are able to connect.</p>
zosConnectNetworkid	string		<p>Optional. The network ID of the z/OS Connect EE server passed to CICS. The default value is 9UNKNOWN.</p> <p>If specified, this value of zosConnectNetworkid is used, together with the value of zosConnectApplid, to match a predefined IPCONN definition in CICS or reject the request if no match is found and the CICS system has not been configured to autoinstall IPCONN connections.</p> <p>If a zosConnectNetworkid value is not specified and the NETWORKID in the CICS IPCONN definition is left blank, a match might not occur even if the z/OS Connect EE and CICS APPLIDs match, because CICS defaults the blank NETWORKID to the local network ID. This local network ID is specified by the z/OS Communications Server NETID or for VTAM=NO systems, the value of its UOWNETQL system initialization parameter and is only defaulted to 9UNKNOWN if no local network ID is set.</p>

zosconnect_dbServices

Defines the location for your user type converter libraries for IMS Database services. To learn more, see [“Configuring user type converter support for IMS database services” on page 262](#).

Attribute name	Data type	Default value	Description
imsDbUtcPaths	String		A colon separated list of fully qualified file paths to your .jar files that contain user type converters.

zosconnect_endpointConnection

Allows requests to be routed from z/OS Connect EE to a request endpoint.

Attribute name	Data type	Default value	Description
basicAuthRef	A reference to top level <code>zosconnect_authData</code> element. (String)		Reference name that identifies the basic authentication data to be used for connecting to a request endpoint. This attribute is now deprecated. It is recommended to use the <code>authenticationConfigRef</code> attribute instead.
authenticationConfigRef	A reference to top level <code>zosconnect_authData</code> , <code>zosconnect_oAuthConfig</code> , <code>zosconnect_authToken</code> or <code>zosconnect_authTokenLocal</code> element. (string)		<p>Reference name that identifies the authentication data used for basic authentication, OAuth 2.0 or JWT when the z/OS Connect EE establishes a connection to a remote REST endpoint:</p> <ul style="list-style-type: none"> For basic authentication, it must be associated with the <code>zosconnect_authData</code> element. For OAuth 2.0, it must be associated with the <code>zosconnect_oAuthConfig</code> element. For using a JWT that is obtained from an authentication server, it must be associated with the <code>zosconnect_authToken</code> element. For using a JWT that is locally generated by the z/OS Connect EE server, it must be associated with the <code>zosconnect_authTokenLocal</code> element. <p>Note: The <code>authenticationConfigRef</code> attribute can reference more than one element to support the combined use of basic authentication, OAuth 2.0 or JWT. For more information, see “Calling an API secured with multiple authentication and authorization methods” on page 438.</p>
connectionTimeout	A period of time with millisecond precision	30s	The connection timeout specifies the amount of time that the z/OS Connect EE server attempts to establish a connection to the request endpoint before it times out. If the timeout value is set to 0, the z/OS Connect EE server attempts to open a connection indefinitely. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.
host	string		The address that is used to route the request to the request endpoint. The value can be the protocol <code>http://</code> or <code>https://</code> followed by the IP address, the domain name server (DNS) hostname with domain name suffix, or just the DNS hostname. If the protocol is not specified, the default protocol <code>http://</code> is used.
id	string		A unique configuration ID.
port	string		Port used for routing HTTP or HTTPS requests.
receiveTimeout	A period of time with millisecond precision	60s	Specifies the amount of time that the z/OS Connect EE server waits for a response from the request endpoint before it times out. If the timeout value is set to 0, the z/OS Connect EE server waits for a response indefinitely. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

Attribute name	Data type	Default value	Description
sslCertsRef	string		An SSL repertoire with an ID, a defined keystore, and truststore.
proxyConfigRef	A reference to top level zosconnect_proxyConfig element. (string)		Reference name that identifies the proxy via which the request is routed from the z/OS Connect EE server to the request endpoint.
domainBasePath	string		An additional path that is added between the {host}:{port} and {basePath} field in an API URL to identify domain-related information.

zosconnect_localAdaptersConnectService

Represents a WOLA service:

The following table lists the attributes that are applicable to both COMMAREA and channel payloads.

Attribute name	Data type	Default value	Description
connectionFactoryRef	string		Reference to the configured connection factory element that contains the JNDI name of the WOLA resource adapter connection factory to be used.
connectionWaitTimeout	integer Minimum: 0 Maximum: 2147483647		Number of seconds to wait for an external address space application that matches the registration name to issue a WOLA Receive Request or Host Service API and become active.
id	string		A unique configuration ID.
serviceName	string		Name of the WOLA target service. This service name must match the name an external address space application is using for the service name on a WOLA Receive Request or Host Service API call, or the program name used for SVC with a WOLA CICS Link Server.
registerName	string		Name of the WOLA target register. This name must match the name an external address space application is using for the register name on a WOLA Register API call, or the name used for Register Name with a WOLA CICS Link Server.
linkTaskTranID	string		When using the WOLA CICS Link Server, specifies the name of the WOLA CICS Link Server link invocation task transaction ID.
useCICSContainer	boolean	false	When using the WOLA CICS Link Server, defines the mechanism to use for data propagation. When set to true, the payload is passed to the target CICS application program using CICS containers. When set to false (default), the payload is passed to the target CICS application program using a COMMAREA. See the tables below for a description of the additional attributes required for channel payloads.
useGenericError	boolean	false	When enabled, all error cases from the service return a HTTP status code of 500 Internal Server Error. This option is retained for compatibility with previous versions of z/OS Connect EE.

There are two different methods of specifying the channel and container attributes. These methods are mutually exclusive:

- **Method 1:** Use a channel name of IBM-WAS-ADAPTER to flow a single payload container. Specify the following attributes:

Attribute name	Data type	Default value	Description
linkTaskReqContID	string		When using the WOLA CICS Link Server and the linkTaskRspContID and useCICSContainer (true) attributes are also configured, specifies the name of the request, or input, container. The default CICS channel name is IBM-WAS-ADAPTER. The container name must not include blank characters.
linkTaskReqContType	integer Minimum : 0	0	When using the WOLA CICS Link Server and the linkTaskReqContID and useCICSContainer (true) attributes are also configured, specifies the type of the request container (0=CHAR, 1=BIT). The default CICS channel name is IBM-WAS-ADAPTER.
linkTaskRspContID	string		When using the WOLA CICS Link Server and the linkTaskReqContID and useCICSContainer (true) attributes are also configured, specifies the name of the response, or output container. The container name must not include blank characters.
linkTaskRspContType	integer Minimum : 0	0	When using the WOLA CICS Link Server and the linkTaskRspContID and useCICSContainer (true) attributes are also configured, specifies the type of response container (0=CHAR, 1=BIT).

- **Method 2:** Use a channel name of your choice to flow a single payload container with the HTTP context containers. Specify the following attributes:

Attribute name	Data type	Default value	Description
linkTaskChanID	string		When using the WOLA CICS Link Server and the useCICSContainer (true) attribute is also configured, specifies the CICS channel name to use for delivering messages and receiving payloads using CICS containers. The channel name must not include blank characters.
linkTaskChanType	integer Minimum: 0	0	When using the WOLA CICS Link Server and the linkTaskChanID and useCICSContainer (true) attributes are also configured, specifies the type of the CICS containers (0=CHAR, 1=BIT) that are to be associated with the configured channel ID. When set to 0(default), the encoding of the character data in the input/output containers are expected/returned in ASCII (CCSID 819) and the data is converted to or from EBCDIC (cp037) before/after it is sent to/ from the destination program. Use the BIT type to avoid data type and encoding expectations.
linkTaskChanReqContID	string	ZCONReqData	When using the CICS Link Server and when the linkTaskChanID and useCICSContainer (true) attributes are also configured, specifies the name of the request container. The container name must not include blank characters.
linkTaskChanRespContID	string	ZCONRspData	When using the CICS Link Server and when the linkTaskChanID and useCICSContainer (true) attributes are also configured, specifies the name of the response container. The container name must not include blank characters.
linkTaskChanCtxContEncoding	string	cp819	When using the CICS Link Server and when the linkTaskChanID and useCICSContainer (true) attributes are also configured, specifies the encoding of the data in all context containers that are sent to the destination program.

Attribute name	Data type	Default value	Description
linkTaskChanCtxHeaders	string		When using the CICS Link Server and the linkTaskChanID and useCICSContainer (true) attributes are also configured, specifies the HTTP header name or list of comma-separated and case-sensitive HTTP header names that are passed to the destination program using the context container with the name of ZCONHTTPHeaders. The information contained in this context container is in JSON format: {httpHeaders: {"header1": "header1Value", ..., "header-n": "headerValue-n"}}. If the request contains multiple headers with the same name, the value that is used is the one for the first header in the request.

zosconnect_mqService

Defines a one or two-way service for the IBM MQ Service Provider.

Note: This element is for users who have already created services from the IBM MQ service provider that is shipped with IBM MQ. Users who want to create new services should use service archive files. For more information, see “[Migrating a service to the IBM MQ service provider in z/OS Connect EE](#)” on page 280.

Attribute name	Data type	Default value	Description
connectionFactory	A JNDI name (string).		Required. Specifies the JNDI name of an IBM MQ messaging provider connection factory. The IBM MQ service provider uses the connection factory to connect to IBM MQ. For more information, see JMS Connection Factory (jmsConnectionFactory) in the <i>WebSphere Application Server for z/OS Liberty</i> documentation.
destination	A JNDI name (string).		Required. Specifies the JNDI name of an IBM MQ messaging provider destination. <ul style="list-style-type: none"> • For a one-way service, the target for HTTP POST, HTTP GET, and HTTP DELETE requests. Queue destinations are supported for all three request types whereas topic destinations are supported only with HTTP POST requests. • For a two-way service, destination must be a queue destination that represents the request queue that is used by the back-end service. Two-way services support only HTTP POST requests. For more information, see JMS Queue (jmsQueue) and JMS Topic (jmsTopic) in the <i>WebSphere Application Server for z/OS Liberty</i> documentation.
expiry	integer	-1	Optional. Specifies the length of time, in thousandths of a second, that messages sent by the MQ service provider are valid. Messages become eligible to be discarded if they have not been removed from the destination queue before this period of time elapses. <p>A negative value means that messages never expire.</p> <p>REST clients can override expiry by specifying an ibm-mq-md-expiry HTTP header with a valid 64-bit integer.</p>

Attribute name	Data type	Default value	Description
id	string		Required. Must be unique across all elements in <code>server.xml</code> . <code>id</code> is used by the <code>zosConnectService</code> element to refer to a target service provider instance.
mqmdFormat	string		Optional. Used to set the value of the MQMD format field in messages that are sent by the IBM MQ service provider. Only used when the IBM MQ service provider is configured to use z/OS Connect EE data transformations, otherwise it is ignored. If you do not specify this attribute, and data transformations are used, messages are sent with the MQMD format field set to "MQSTR". The length of this attribute must be less than, or equal to, 8 characters.
password	string		Optional. The password that the IBM MQ service provider presents to IBM MQ for authentication and authorization purposes. You can specify the password in plain text, but best practice is to encode the password using the <code>securityUtility</code> tool that is provided with z/OS Connect EE, with the <code>encode</code> option. For more information, see Liberty: securityUtility command . If you do not specify this attribute, the value in the <code>password</code> attribute that is specified in the Connection Factory referred to by the <code>connectionFactory</code> attribute is used. If a <code>password</code> attribute is specified both on the referenced connection factory and on this <code>zosconnect_mqService</code> element, the value in the <code>zosconnect_mqService</code> element is used. If you specify this attribute, you must also specify the <code>userName</code> attribute.
persistence	boolean	false	Optional. Specifies the persistence of messages sent by a service and is equivalent to setting the MQMD Persistence field. The value must be one of the following: false Means messages are non-persistent. true Means messages are persistent. You can override <code>persistence</code> by setting one of these values in the <code>ibm-mq-md-persistence</code> HTTP header.
receiveTextCCSID	integer	37	Optional. The CCSID that is used to transform the data in a <code>javax.jms.TextMessage</code> message. For example, an HTTP GET or HTTP DELETE with a one-way service, or when retrieving a response message for a two-way service. The text in the message is converted into the CCSID specified by <code>receiveTextCCSID</code> .

Attribute name	Data type	Default value	Description
replyDestination	A JNDI name (string).		<p>Optional. Specifies the JNDI name of an IBM MQ messaging provider queue where the back-end service sends reply messages.</p> <p>If <code>replyDestination</code> is not specified, the service is a one-way service. If <code>replyDestination</code> is specified, the service is a two-way service.</p>
replySelection	string	msgIDToCorrelID	<p>Optional. Describes the mechanism that is used to match reply messages with request messages.</p> <p><code>replySelection</code> is used only with two-way services. If <code>replySelection</code> is used with a one-way service, it is ignored.</p> <p>Specify one of the following values:</p> <p>msgIDToCorrelID Reply messages are assumed to be generated with the correlation ID set to the value of the message ID from the request message. The service generates a suitable message selector based on this information.</p> <p>correlIDToCorrelID Reply messages are assumed to be generated with the correlation ID set to the value of the correlation ID from the request message. The service generates a suitable message selector based on this information. If the request message does not have a correlation ID specified, the service generates a random correlation ID for the request message.</p> <p>none No mechanism is used to correlate reply messages with request messages. The service gets the first available message on the reply queue.</p>
selector	string		<p>Optional. Used on HTTP GET and HTTP DELETE requests to select which message is returned. Must be set to a valid JMS message selector as described by the JMS specification.</p> <p>If <code>ibm-mq-md-msgID</code> or <code>ibm-mq-md-correlID</code> headers are specified, <code>selector</code> is ignored.</p> <p><code>selector</code> is only used with one-way services and is optional. If <code>selector</code> is specified on a two-way service it is ignored.</p> <p>Some characters in the attribute value must be escaped in order to be embedded in <code>server.xml</code> because these special characters cannot be directly used in XML. For example,</p> <ul style="list-style-type: none"> " escaped with &quot; ' escaped with &apos; < escaped with &lt; > escaped with &gt;

Attribute name	Data type	Default value	Description
useCallerPrincipal	boolean	false	<p>Optional. When set to true, the name of the authenticated principal of a request to z/OS Connect EE, is passed on to IBM MQ for authentication and authorization purposes.</p> <p>The name of the principal, but not the password, is used when connecting to IBM MQ. Any values specified in the password and userName attributes are ignored.</p>
userName	string		<p>The user name that the IBM MQ service provider presents to IBM MQ for authentication and authorization purposes.</p> <p>If you do not specify this attribute, the value of the userName attribute that is specified in the connection factory referred to by the connectionFactory attribute is used.</p> <p>If a userName attribute is specified on both the referenced connection factory and this zosconnect_mqService element, the value in the zosconnect_mqService element is used.</p> <p>If you specify this attribute, you must also specify the password attribute.</p>
waitInterval	integer		<p>This attribute is optional for one-way services, required for two-way services.</p> <p>For HTTP DELETE requests to one-way services, waitInterval specifies the number of milliseconds that the service waits for a matching message on the queue specified by the destination attribute.</p> <p>For HTTP POST requests to two-way services, waitInterval specifies the number of milliseconds that the service waits for a matching message on the queue specified by the replydestination attribute.</p> <p>waitInterval is not supported with HTTP GET requests. If waitInterval is zero, the service does not wait.</p> <p>A waitInterval of zero is not supported with two-way services.</p> <p>If waitInterval is negative, the service waits for ever until a message is available.</p> <p>REST clients can override this value by specifying an ibm-mq-gmo-waitInterval HTTP header with a valid 64 bit integer.</p> <p>Note: Specifying a large, or negative waitInterval, is likely to result in transaction timeouts and asynchronous service request timeouts. If these events occur, you can increase the timeout, reduce the wait interval, or do both.</p>

zosconnect_oAuthConfig

Defines the OAuth 2.0 configuration in z/OS Connect EE. For more information about supported security configuration options when using OAuth 2.0, see [“How to configure OAuth 2.0” on page 425](#).

Attribute name	Data type	Default value	Description
id	string		A unique configuration ID.

Attribute name	Data type	Default value	Description
grantType	Either password or client_credentials (string)		Specifies the OAuth grant type. In z/OS Connect EE, only two grant types are supported. If set to password, the Resource Owner Password Credential grant type is used. If set to client_credentials, the Client Credentials grant type is used.
authServerRef	Reference to top level zosconnect_authorizationServer Element. (string)		Reference name that identifies the information of an authorization server that is used for authentication and authorization.

zosconnect_policy

Defines the z/OS Connect EE policy rules to be applied to API requests.

Attribute name	Data type	Default value	Description
id	String		Required. Unique name to identify this policy.
location	String	\$ {server.config.dir }/resources/zosconnect/rules	The directory where the rule set file is located.
pollingRate	long (A period of time with millisecond precision)	1m	For dynamic configuration, controls how often the server polls the directory that contains the ruleset files. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.
updateTrigger	String	disabled	<p>Controls when the runtime is notified about changes in the ruleset directory. Acceptable values are disabled or polled.</p> <p>disabled Polling for updates is disabled. Updates can be triggered using the MODIFY refresh command.</p> <p>polled The server will periodically check for changes to the ruleset directory contents.</p> <p>The value of this attribute is ignored when the MODIFY command is used to refresh the z/OS Connect EE server artifacts.</p>

Sub elements

zosconnect_policy > ruleset

Description: Defines the name of the rule set file.

Required: true

Attribute name	Data type	Default value	Description
file			The file name of the rule set. Note: Do not include the path.

zosconnect_proxyConfig

Allows requests to be routed from z/OS Connect EE to an endpoint via a proxy.

Attribute name	Data type	Default value	Description
id	String		Required. A unique configuration ID.
host	String		Required. The IP address, domain name server (DNS) host name with domain name suffix, or just the DNS host name of the proxy server, used to route the request.
port	Integer		Required. Port used by the proxy server for routing HTTP or HTTPS requests.
type	String		Required. Proxy type, the value should be HTTP or SOCKS.

zosconnect_services

Defines the directory where service archive files are stored and how the server is notified about changes in this directory. This element must be defined for service archive files to be processed at server start up, even if all the attributes use their default values.

Note: This element is only for specific services defined by service archive files (.sar files)

Attribute name	Data type	Default value	Description
location	string	\$ {server.config.dir}/resources/zosconnect/services	Path to a directory location where service archive (.sar) files are stored. This location is referred to as the "services directory". The value of location cannot be changed while the server is running; the value is set when the server is started.
pollingRate	A period of time with millisecond precision	5s	Controls how often the server polls the services directory. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

Attribute name	Data type	Default value	Description
updateTrigger	string	disabled	<p>Controls when the runtime is notified about changes in the services directory. Acceptable values are disabled, polled or mbean. The default value is disabled.</p> <p>disabled Polling for updates is disabled. Updates can be triggered using the MODIFY refresh command, and service archive files can be deployed using the RESTful administration interface.</p> <p>polled The server will periodically check for changes to the directory contents.</p> <p>mbean The server will check for changes when the notifyFileChanges method is invoked on the FileNotificationMBean. If you specify this value, you must also configure your server to use the Java Management Extensions (JMX) connector. For more information, see “Using an MBean to trigger updates” on page 301.</p> <p>The value of this attribute is ignored when the MODIFY command is used to refresh the z/OS Connect EE server artifacts.</p>

Sub elements

zosconnect_services > service

Description: Defines additional configuration for the service.

Required: false

Attribute name	Data type	Default value	Description
name	string		The name of the service.
requireSecure	boolean		Require that requests are sent over HTTPS. If the requireSecure attribute is not set, the global setting from the requireSecure attribute on the zosconnect_zosConnectManager element is used instead.
requireAuth	boolean		Require that users specify security credentials to be authenticated and that the authenticated user is authorized under the zosConnectAccess role, in order to access the service. If the requireAuth attribute is not set, the global setting from the requireAuth attribute on the zosconnect_zosConnectManager element is used instead.
runGlobalInterceptors	boolean	true	Indicates whether global interceptors should run for requests that are associated with this service. Global interceptors are listed in globalInterceptorsRef in the zosconnect_zosConnectManager element. By default, z/OS Connect EE processes all global and service endpoint-specific interceptors. If the runGlobalInterceptors is set to false, z/OS Connect EE processes only the set of interceptors that are listed in the interceptorsRef attribute.

Attribute name	Data type	Default value	Description
interceptorsRef	string		<p>Reference name that identifies the set of configured interceptors that are associated with this service.</p> <p>Note: If the service is called by an API, only the interceptors configured for that API are processed. Interceptors defined for the service are ignored.</p>
adminGroup	string		<p>Identifies the users that are able to use administrative functions on this service. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. If <code>globalAdminGroup</code> is also defined under element <code>zosconnect_zosConnectManager</code>, the value defined under <code>adminGroup</code> is used. See Note 1.</p>
invokeGroup	string		<p>Identifies the users that are able to invoke this service. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. If <code>globalInvokeGroup</code> is also defined under element <code>zosconnect_zosConnectManager</code>, the value defined under <code>invokeGroup</code> is used. See Note 1.</p>
operationsGroup	string		<p>Identifies the users that are able to perform operations such as starting or stopping this service. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. If <code>globalOperationsGroup</code> is also defined under element <code>zosconnect_zosConnectManager</code>, the value defined under <code>operationsGroup</code> is used. See Note 1.</p>
readerGroup	string		<p>Identifies the users that are able to get information about this service, including the Swagger documentation. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. If <code>globalReaderGroup</code> is also defined under element <code>zosconnect_zosConnectManager</code>, the value defined under <code>readerGroup</code> is used. See Note 1.</p>
property	A list of property elements.		<p>The <code>property</code> elements define the override properties for the service archive file.</p>

Note 1: If using an LDAP registry, you must specify each LDAP group's distinguished name (DN) with the commas escaped with a backslash. for example "cn=employees\,ou=groups\,o=intern\,c=fr, cn=managers\,ou=groups\,o=intern\,c=fr". If specifying multiple groups, the commas separating the groups are not escaped. Specifying LDAP short names is not supported.

zosconnect_services > service > property

Description: Optional properties for the service provider. These values take precedence over equivalent values specified in the service archive file.

Required: false

Attribute name	Data type	Default value	Description
name	string		The name of the property
value	string		The value of the property

The values that can be specified for the name attribute are specific to the service provider being used by the service. The following name attribute values are supported:

- **CICS service provider:** transid and transidUsage.
- **Db2 service project**

Attribute name	Description
collectionId	Overrides the value of Collection ID that was entered in the project editor.
connectionRef	Overrides the value of Connection Reference that was entered in the project editor.

- **REST client service provider:** does not support optional properties.
- **WOLA service provider:** tranId.
- **IMS service provider:**

Attribute name	Default value	Description
imsTranCodeOverride		Overrides the transaction code that the service invokes at run time.
imsConnectionRef		Name of the connection profile to use at service invocation. This value overrides the connection profile that is specified in the API toolkit during service archive file generation.
imsInteractionRef		Name of the interaction profile to use at service invocation. This value overrides the interaction profile that is specified in the API toolkit during service creation.
imsDatastoreOverride		Overrides the IMS datastore name that specifies the IMS subsystem against which to invoke the service.
trimOutputLeadingWhiteSpace	false	Trims the leading whitespace from JSON property values in the output messages. By default, leading whitespace is not trimmed.
trimOutputTrailingWhiteSpace	true	Trims trailing whitespace from JSON property values in output messages. By default, trailing whitespace is trimmed.
escapeOutputControlCharacters	false	Escapes non-printable control characters, such as tokens or control blocks, in JSON property values as \uNNNN for necessary internal processing, instead of removing them. By default, control characters are omitted, not escaped.
initializeInputFields	false	Initializes fields in the input data structure according to their type if a default is not specified for the field and either the field is omitted from the input interface or the field is included but the respective JSON tag is not received at run time. By default, fields are not initialized.
omitOutputFieldsByValue	false	Omits the JSON name-value pair for a non-numeric field from the JSON output message when the data for the field is composed of the same byte value repeated throughout, such as all 0x00 or all 0xFF. By default, this option is set to false.
omitOutputFieldsByValueByte	00	Specifies the hexadecimal value that all bytes in a non-numeric field must contain to be omitted. The default value is 00.
omitOutputEmptyTags	false	Omits JSON tags that contain an empty string ("tag": "") from JSON output messages after whitespace and control characters are processed. By default, empty tags are not omitted.

Attribute name	Default value	Description
enforceMinArrayOccurrence	true	Enforces the minimum number of array occurrence in the input data structure as defined in the copybook. By default, minimum number of array occurrence is enforced.

- **IBM MQ service provider:**

Property name	Default value	Description
connectionFactory		Defines a JNDI name that is used to locate a connection factory that connects to a z/OS queue manager on the same LPAR as the z/OS Connect EE server or on a different LPAR. For more information, see JMS connection factory in the <i>WebSphere Application Server for z/OS Liberty</i> documentation.
destination		<p>Defines a JNDI name that is used to locate an IBM MQ queue or topic.</p> <p>For one-way services for sending messages, the queue or topic that messages are put to. For one-way services for receiving messages, this value is the name of the queue where messages are destructively got.</p> <p>For two-way services, this value is the name of the queue where request messages are put.</p> <p>For more information, see the following topics in the <i>WebSphere Application Server for z/OS Liberty</i> documentation: JMS Queue (jmsQueue) if the destination is a queue or JMS Topic (jmsTopic) if the destination is a topic.</p>
expiry	-1	<p>Specifies the expiry time in milliseconds of messages that are sent by the IBM MQ service provider. If set, the value is an integer that describes how long the message is available before it expires. By default, messages do not expire.</p> <p><code>expiry</code> is equivalent to setting the MQMD Expiry field.</p> <p>Negative values mean that messages never expire.</p> <p>REST clients can override <code>expiry</code> by specifying an <code>ibm-mq-md-expiry</code> HTTP header with a valid 64-bit integer.</p>
mqmdFormat		Completes the format field of the MQMD header in messages that are sent by the IBM MQ service provider. Only supported when the <code>language</code> property is specified. If not specified, then messages are sent with a blank format.
password		<p>The password that the IBM MQ service provider presents to IBM MQ for authentication and authorization purposes.</p> <p>The password can be provided in plain text, but this is not good practice. Instead, encode the password by using the <code>securityUtility</code> tool that is provided with z/OS Connect EE, using the <code>encode</code> option. For more information, see securityUtility command in the <i>WebSphere Application Server for z/OS Liberty</i> documentation.</p> <p>If this property is specified, the <code>userName</code> property must also be specified.</p> <p>If this property is not specified, the <code>password</code> property in the connection factory that is referred to by the <code>connectionFactory</code> property is used.</p> <p>If a <code>password</code> property is specified in both the referenced connection factory and this <code>service</code> property subelement, the <code>service</code> property value is used.</p>
persistence	false	Describes the persistence of messages that are sent to the queue referenced by the <code>destination</code> property. If set to <code>true</code> , messages are sent as persistent.

Property name	Default value	Description
replyDestination		<p>Defines a JNDI name that is used to locate a queue that contains response messages for two-way services. If specified, the service is a two-way service. This property is configured in the same way as the destination property.</p> <p>Can be set only if <code>replyDestination</code> is already set in the service archive file.</p>
replySelection	msgIDToCorrelID	<p>Defines how a two-way service locates reply messages on the queue that is referenced by the <code>replyDestination</code> property. If <code>replySelection</code> is used with a one-way service, it is ignored. The following values are valid:</p> <p>msgIDToCorrelID Reply messages are assumed to be generated with the correlation ID set to the value of the message ID from the request message. The service generates a suitable message selector based on this information.</p> <p>none No mechanism is used to correlate reply messages with request messages. The service gets the first available message on the reply queue.</p> <p>correlIDToCorrelID Reply messages are assumed to be generated with the correlation ID set to the value of the correlation ID from the request message. The service generates a suitable message selector based on this information. If the request message has no correlation ID specified, the service generates a random correlation ID for the request message. For more information, see “ibm-mq-md-correlID” on page 272.</p>
selector		<p>Defines a valid JMS message selector that is used to locate messages from the queue that is referenced by the destination attribute. Only valid with one-way services for receiving messages. For more information, see Message selectors in JMS in the <i>IBM MQ</i> documentation.</p> <p>Some characters in the attribute value must be escaped in order to be embedded in <code>server.xml</code> because these special characters cannot be directly used in XML. For example,</p> <ul style="list-style-type: none"> “ escaped with &quot; ' escaped with &apos; < escaped with &lt; > escaped with &gt;
useCallerPrincipal		<p>When a request is made to z/OS Connect EE, the caller authenticates with the z/OS Connect EE server. The name of the authenticated principal can be passed onto IBM MQ for authentication and authorization purposes by setting the value of <code>useCallerPrincipal</code> to <code>true</code>.</p> <p>The name of the principal, but no password, is used to connect to IBM MQ. Any values that are specified in the <code>password</code> and <code>userName</code> attributes are ignored.</p>

Property name	Default value	Description
userName		<p>The user name that the IBM MQ service provider presents to IBM MQ for authentication and authorization purposes.</p> <p>If this property is not specified, the <code>userName</code> property in the connection factory that is referred to by the <code>connectionFactory</code> property is used.</p> <p>If a <code>userName</code> property is specified both in the referenced connection factory and this <code>service</code> property subelement, the <code>service</code> property value is used.</p> <p>If this property is specified, the <code>password</code> property must also be specified.</p>
waitInterval		<p>Specifies a time, in milliseconds that the IBM MQ service provider waits for messages to arrive on a queue.</p> <p>This property is only valid for two-way services, or one-way services for receiving messages.</p> <p>If <code>replyDestination</code> is set, then <code>waitInterval</code> must be a positive number.</p> <p>If <code>messagingAction=mqget</code>, then <code>waitInterval</code> can be negative, which means that the IBM MQ service provider waits forever until a message is available. If <code>waitInterval</code> is 0, the IBM MQ service provider does not wait.</p>

zosconnect_zosConnectAPIs

Defines the directory where API archive files are stored and how the server is notified about changes in this directory. It also defines additional configuration which applies to all APIs

Attribute name	Data type	Default value	Description
location	string	<code>\$ {server.config.dir}/resources/zosconnect/apis</code>	Path to a directory location where API archive files are stored. The value of location cannot be changed while the server is running; the value is set when the server is started.
policyRef	string		Reference name that identifies the zosconnect_policy element that is active for these APIs.
pollingRate	long (A period of time with millisecond precision)	5s	Controls how often the server polls the <code>apis</code> directory. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.

Attribute name	Data type	Default value	Description
updateTrigger	string	disabled	<p>Controls when the runtime is notified about changes in the apis directory. Acceptable values are disabled, polled or mbean.</p> <p>disabled Polling for updates is disabled. Updates can be triggered using the MODIFY refresh command, and API archive files can be deployed using the RESTful administration interface.</p> <p>polled The server will periodically check for changes to the directory contents.</p> <p>mbean The server will check for changes when the notifyFileChanges method is invoked on the FileNotificationMBean.</p> <p>If you specify this value, you must also configure your server to use the Java Management Extensions (JMX) connector. For more information, see “Using an MBean to trigger updates” on page 301.</p> <p>The value of this attribute is ignored when the MODIFY command is used to refresh the z/OS Connect EE server artifacts.</p>

Sub elements

zosconnect_zosConnectAPIs > zosConnectAPI

Description: Defines additional configuration for the API.

Required: true

Attribute name	Data type	Default value	Description
name	string		A name of the API.
requireSecure	boolean		Require that requests are sent over HTTPS. If the requireSecure attribute is not set, the global setting from the requireSecure attribute on the zosconnect_zosConnectManager element is used instead.
requireAuth	boolean		Require that users specify security credentials to be authenticated and that the authenticated user is authorized under the zosConnectAccess role, in order to access the API. If the requireAuth attribute is not set, the global setting from the requireAuth attribute on the zosconnect_zosConnectManager element is used instead.
runGlobalInterceptors	boolean	true	Indicates whether global interceptors should run for requests that are associated with this API. Global interceptors are listed in globalInterceptorsRef in the zosconnect_zosConnectManager element. By default, z/OS Connect EE processes all global and endpoint-specific interceptors. If the runGlobalInterceptors is set to false, z/OS Connect EE processes only the set of interceptors that are listed in the interceptorsRef attribute.
interceptorsRef	string		<p>Reference name that identifies the set of configured interceptors that are associated with this API.</p> <p>Note: If an API operation is invoked, only the interceptors configured for the API are processed. The service-specific interceptor that is configured on the related service is never called.</p>

Attribute name	Data type	Default value	Description
adminGroup	string		Identifies the users that are able to use administrative functions on this API. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. If globalAdminGroup is also defined under element <code>zosconnect_zosConnectManager</code> , the value defined under adminGroup is used. See Note 1 .
invokeGroup	string		Identifies the users that are able to invoke this API. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. If globalInvokeGroup is also defined under element <code>zosconnect_zosConnectManager</code> , the value defined under invokeGroup is used. See Note 1 .
operationsGroup	string		Identifies the users that are able to perform operations such as starting or stopping this API. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. If globalOperationsGroup is also defined under element <code>zosconnect_zosConnectManager</code> , the value defined under operationsGroup is used. See Note 1 .
policyRef	string		Reference name that identifies the “ <code>zosconnect_policy</code> ” on page 773 element that is active for this API.
readerGroup	string		Identifies the users that are able to get information about this API, including the Swagger documentation. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. If globalReaderGroup is also defined under element <code>zosconnect_zosConnectManager</code> , the value defined under readerGroup is used. See Note 1 .

Note 1: If using an LDAP registry, you must specify each LDAP group's distinguished name (DN) with the commas escaped with a backslash. for example "cn=employees\,ou=groups\,o=intern\,c=fr, cn=managers\,ou=groups\,o=intern\,c=fr". If specifying multiple groups, the commas separating the groups are not escaped. Specifying LDAP short names is not supported.

`zosconnect_zosConnectDataXform`

Defines a z/OS Connect EE data transformer.

Attribute name	Data type	Default value	Description
bindFileLoc	string		File system path where the bind files are located.
bindFileSuffix	string		Suffix name that is associated with the bind files.
id	string		A unique configuration ID.
pollingRate	A period of time with millisecond precision	2s	Rate at which the server checks for updates to data transformation-related files such as bind or schema files. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.
requestSchemaLoc	string		File system path where the request schema files are located.

Attribute name	Data type	Default value	Description
requestSchemaSuffix	string		Suffix name that is associated with the request schema files.
responseSchemaLoc	string		File system path where the response schema files are located.
responseSchemaSuffix	string		Suffix name that is associated with the response schema files.
updateTrigger	<ul style="list-style-type: none"> • mbean • polled • disabled 	polled	<p>Update trigger for data transformation files such as bind and schema files.</p> <p>mbean Server reloads data transformation files when prompted by an MBean called by an external program such as an integrated development environment or a management application.</p> <p>If you specify this value, you must also configure your server to use the Java Management Extensions (JMX) connector. For more information, see “Using an MBean to trigger updates” on page 301.</p> <p>polled Server scans for changes at the polling interval and reload any files that have detectable changes.</p> <p>disabled Polling for updates is disabled. Updates can be triggered using the MODIFY refresh command.</p> <p>The value of this attribute is ignored when the MODIFY command is used to refresh the z/OS Connect EE server artifacts.</p>

zosconnect_fileSystemloggerInterceptor

Defines a z/OS Connect EE File System logger interceptor.

Attribute name	Data type	Default value	Description
bufferSize	integer	8192	Buffer size in bytes to be used when the bufferLogging attribute is set to true.
bufferedLogging	boolean	false	Indicates whether entries to the log are buffered before they are written to the log file.
encoding	string	UTF-8	Encoding that is used when writing to the log file.
id	string		A unique configuration ID.
logName	string		Log file name pattern that is used for payload logging.
logOption	<ul style="list-style-type: none"> • RESPONSE • REQUEST • ALL 	ALL	<p>Log option that controls what is logged.</p> <p>RESPONSE Indicates that only response data is logged.</p> <p>REQUEST Indicates that only request data is logged.</p> <p>ALL Indicates that both request and response data are logged.</p>

Attribute name	Data type	Default value	Description
logPath	string		File system location where the log file is created. By default, the log files are created in the \${server.config.dir}/logs/zosConnect directory. For example, if you use the default value of /var/zosconnect for the WLP_USER_DIR environment variable, then the file system logger interceptor will write the logs to /var/zosconnect/servers/serverName/logs/zosConnect.
maxPayloadSize	integer	524288	Maximum payload size in characters that are allowed to be written to the log file.
rollOffLogPolicy	<ul style="list-style-type: none"> • SIZE • DURATION 	SIZE	<p>Indicates that log file is rolled off based on size or duration.</p> <p>SIZE Indicates that the log file is roll-off based on the size of the log.</p> <p>DURATION Indicates that the roll-off log policy is based on the elapsed time since the active log file was created.</p>
rollOffLogPolicyDuration	integer	1440	Roll off policy duration in minutes.
rollOffLogPolicySize	integer	52428800	Roll off policy size in bytes.
sequence	integer Minimum: 0 Maximum: 2147483647	0	Sequence in which the interceptor is processed with respect to others.

zosconnect_zosConnectInterceptors

Bundles 1 to N interceptors.

Attribute name	Data type	Default value	Description
id	string		A unique configuration ID.
interceptorRef	List of references to top level interceptor elements (comma-separated string).		The identifier of one or more interceptors.

zosconnect_zosConnectManager

Defines global configuration settings for z/OS Connect EE.

Attribute name	Data type	Default value	Description
id	string		A unique configuration ID.

Attribute name	Data type	Default value	Description
asyncRequestTimeout	A period of time with millisecond precision	30s	Timeout value that is associated with every HTTP request (services and APIs) when processing asynchronous work. This does not apply to API requester calls. It specifies the time in milliseconds in which requests have to complete. This timeout value overrides the web container's <code>asyncTimeoutDefault</code> attribute value. If neither <code>asyncRequestTimeout</code> nor <code>asyncTimeoutDefault</code> are configured, the timeout used is the <code>asyncTimeoutDefault</code> attribute default value (i.e. 30 seconds). If <code>asyncRequestTimeout</code> is not configured, but the <code>asyncTimeoutDefault</code> attribute is, the value defined in <code>asyncTimeoutDefault</code> is used. A timeout might occur at any time during processing of the request z/OS Connect EE. The request might still be active after the timeout is detected and a response is sent to the client. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds. An <code>asyncRequestTimeout</code> value of zero means "do not time out". From V3.0.8, requests that timeout receive a 503 HTTP response code instead of a 500 HTTP response code.
globalAdminGroup	string		Identifies the users that are able to use administrative functions on all APIs, services, service endpoints and API requesters. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. See Note 1 .
globalDataXformRef	A reference to top level <code>zosconnect_zosConnectDataXform</code> element (string).		Reference name that identifies the data transformation handler that is associated with all service endpoints.
globalInterceptorsRef	tokenType		Reference name that identifies the set of configured interceptors that is associated with all APIs and services. If services do not require global interceptors association, the <code>runGlobalInterceptors</code> attribute of the <code>zosconnect_zosConnectService</code> element can be set to false. If APIs do not require global interceptors association, the <code>runGlobalInterceptors</code> attribute of the <code>zosConnectAPI</code> element can be set to false. See Note 1 .
globalInvokeGroup	string		Identifies the users that are able to invoke all APIs, services, service endpoints and API requesters. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. See Note 1 .
globalOperationsGroup	string		Identifies the users that are able to perform operations such as starting, stopping or obtaining the status of all APIs, services, service endpoints and API requesters. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. See Note 1 .
globalReaderGroup	string		Identifies the users that are able to get lists of, or information about, all APIs, services, service endpoints and API requesters, including Swagger documentation. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. See Note 1 .

Attribute name	Data type	Default value	Description
operationMode	<ul style="list-style-type: none"> • SYNC • ASYNC 	ASYNC	<p>Specifies the mode in which z/OS Connect EE requests will be processed.</p> <p>SYNC Synchronous mode of operation. Requests use the same thread.</p> <p>ASYNC Asynchronous mode of operation. Multiple threads are used to manage requests.</p> <p>Use SYNC when running z/OS Connect EE embedded in CICS and ASYNC when running standalone.</p>
preserveJsonObjectPayloadOrder	boolean	false	When enabled, the order of entries in a JSON object payload is preserved.
preserveJsonPayloadCharFormat	boolean	false	Indicates if the characters in the JSON payload should flow unchanged through z/OS Connect EE during request handling such as API invocation, service invocation and schema retrievals. When set to false, UTF-8 encoded characters might be converted to their respective escaped Unicode representation. For this attribute to take effect, the attribute definition setUTF8ResponseEncoding must be set to true.
requireAuth	boolean	true	Require that users specify security credentials to be authenticated and that the authenticated user is authorized under the zosConnectAccess role, in order to access APIs, services and API requesters, unless overridden on the specific resource definitions.
requireSecure	boolean	true	Require that requests are sent over HTTPS. The requireSecure attribute can be overridden on the specific resource definitions.
useJsonErrorResponses	boolean	true	When enabled, all error responses from the server are in JSON format. This option is retained for compatibility with previous versions of z/OS Connect EE.
setUTF8ResponseEncoding	boolean	false	<p>Indicates whether the character encoding in the HTTP response is set to UTF-8. The default encoding is ISO-8859-1. Set this attribute to true if your service or API contains double-byte character set (DBCS) characters.</p> <p>For compatibility with previous versions of z/OS Connect EE, this option defaults to false. Set this value to true to conform to the standard JSON encoding of UTF-8.</p>

Note 1: If using an LDAP registry, you must specify each LDAP group's distinguished name (DN) with the commas escaped with a backslash. for example "cn=employees\,ou=groups\,o=intern\,c=fr, cn=managers\,ou=groups\,o=intern\,c=fr". If specifying multiple groups, the commas separating the groups are not escaped. Specifying LDAP short names is not supported.

Sub elements

zosconnect_zosConnectManager > globalInterceptorsRef

Description: Reference name that identifies the set of configured interceptors that is associated with all APIs and services. If services do not require global interceptors association, the runGlobalInterceptors attribute of the zosconnect_zosConnectService element can be set to false. If APIs do not require global interceptors association, the runGlobalInterceptors attribute of the zosConnectAPI element can be set to false.

Note:

1. If a service is called by an API, only the interceptors configured for the API are processed. Interceptors defined for the service are ignored.
2. The set of configured interceptors referenced by `globalInterceptorsRef` is run for every HTTP request (services and APIs).

Required: false

Data type: tokenType

zosconnect_zosConnectService

Defines the configuration settings for a service endpoint.

Note: This element is for services that are not defined by service archive files.

Attribute name	Data type	Default value	Description
adminGroup	string		Identifies the users that are able to use administrative functions on this service endpoint. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. If it is configured along with its global counterpart, <code>globalAdminGroup</code> defined under element <code>zosconnect_zosConnectManager</code> , the value defined under <code>adminGroup</code> is used. See Note 1 .
dataXformRef	string		Reference name that identifies the data transformation handler that is associated with a service endpoint. If configured along with its global data transformation handler counterpart (<code>globalDataXformRef</code> defined under element <code>zosconnect_zosConnectManager</code>), the data transformer defined for the service endpoint is used.
id	string		A unique configuration ID.
invokeGroup	string		Identifies the users that are able to invoke this service endpoint. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. If it is configured along with its global counterpart, <code>globalInvokeGroup</code> defined under element <code>zosconnect_zosConnectManager</code> , the value defined under <code>invokeGroup</code> is used. See Note 1 .
invokeURI	string		URI or list of comma-separated URIs to associate with a service endpoint. InvokeURIs can end with a wildcard character in the form /a/b/* or x/y* to generically match a service endpoint invocation. Specifying multiple wildcard characters (i.e. /a/b/**) or wildcard characters in the middle of the requestURI (i.e. /a/*/*) is not supported. If service endpoints with configured invokeURIs using the wildcard character are associated with overlapping invokeURIs, the service endpoint associated with the most specific invokeURI is matched. For instance, if a service endpoint request is issued with the following: https://host:port/a/b/c going to a server with the following configuration: service1 -> invokeURI="/a/b/c/*" and service2 -> invokeURI="/a/b/*", z/OS Connect EE will match the request to service1. Configured invokeURI entries must start with the / character. The use of an invokeURI is equivalent to a service request where the action=invoke query parameter is specified.
operationsGroup	string		Identifies the users that are able to perform operations such as starting, stopping or obtaining the status of this service endpoint. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. If it is configured along with its global counterpart, <code>globalOperationsGroup</code> defined under element <code>zosconnect_zosConnectManager</code> , the value that is defined under <code>operationsGroup</code> is used. See Note 1 .

Attribute name	Data type	Default value	Description
readerGroup	string		Identifies the users that are able to get information about this service endpoint, including the Swagger documentation. The value of this attribute can be set to a group name or a comma-separated list of group names, that are defined in the user registry. If <code>globalReaderGroup</code> is also defined under element <code>zosconnect_zosConnectManager</code> , the value defined under <code>readerGroup</code> is used. See Note 1 .
requireAuth	boolean		Require that users specify security credentials to be authenticated and that the authenticated user is authorized under the <code>zosConnectAccess</code> role, in order to access the service. If the <code>requireAuth</code> attribute is not set, the global setting from the <code>requireAuth</code> attribute on the <code>zosconnect_zosConnectManager</code> element is used instead.
requireSecure	boolean		Require that requests are sent over HTTPS. If the <code>requireSecure</code> attribute is not set, the global setting from the <code>requireSecure</code> attribute on the <code>zosconnect_zosConnectManager</code> element is used instead.
runGlobalInterceptors	boolean	true	Indicates whether global interceptors run for requests that are associated with a service endpoint. By default z/OS Connect EE processes all global and service endpoint-specific interceptors.
serviceAsyncRequestTimeout	A period of time with millisecond precision	30s	<p>Timeout value that is associated with a service endpoint when processing asynchronous work. It specifies the time in milliseconds in which requests must complete. This timeout value overrides the web container's <code>asyncTimeoutDefault</code> attribute value. If neither <code>asyncRequestTimeout</code> nor <code>asyncTimeoutDefault</code> are configured, the timeout used is the <code>asyncTimeoutDefault</code> attribute default value (i.e. 30 seconds). If <code>asyncRequestTimeout</code> is not configured, but the <code>asyncTimeoutDefault</code> attribute is, the <code>asyncTimeoutDefault</code>'s configured value is used. If configured along with its global counterpart: <code>asyncRequestTimeout</code> defined under element <code>zosconnect_zosConnectManager</code>, the value defined under <code>serviceAsyncRequestTimeout</code> is used. A timeout might occur at any time during z/OS Connect's processing of the request. The request might still be active after the timeout is detected and a response is sent to the client. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds. From V3.0.8, requests that timeout receive a 503 HTTP response code instead of a 500 HTTP response code.</p> <p>This timeout applies only to direct requests to this service endpoint and not if this service endpoint is invoked via an API.</p>
serviceDescription	string		Description that is associated with a service endpoint.
serviceGroupingName	string		Name that can be used to group or associate a set of service endpoints together.
serviceName	string		Name that is associated with a service endpoint. This name identifies a service endpoint to a client.
serviceRef	string		Reference name that identifies the service endpoint that is registered with z/OS Connect.

Note 1: If using an LDAP registry, you must specify each LDAP group's distinguished name (DN) with the commas escaped with a backslash, for example "cn=employees\,ou=groups\,o=intern\,c=fr, cn=managers\,ou=groups\,o=intern\,c=fr". If specifying multiple groups, the commas separating the groups are not escaped. Specifying LDAP short names is not supported.

Sub elements

zosconnect_zosConnectService > interceptorsRef

Description: Reference name that identifies the set of configured interceptors that is associated with a service endpoint. If it is configured along with its global interceptors counterpart (`globalInterceptorsRef` defined under the `zosconnect_zosConnectManager` element), z/OS Connect EE processes both sets of interceptors. If the `runGlobalInterceptors` attribute for the service endpoint is set to false, z/OS connect will only process the set of interceptors configured for the service endpoint.

Required: false

Data type: tokenType

zosconnect_zosConnectServiceRestClient

Allows requests to be routed from z/OS Connect EE to a remote REST endpoint.

Attribute name	Data type	Description
basicAuthRef	A reference to top level <code>zosconnect_zosConnectServiceRestClientBasicAuth</code> element (string).	Reference name that identifies the basic authentication data to be used for connecting to a remote REST endpoint.
connectionRef	A reference to top level <code>zosconnect_zosConnectServiceRestClientConnection</code> element (string)	If set, the connection will be made using the attributes of the <code>zosconnect_zosConnectServiceRestClientConnection</code> element. If not set, or the <code>zosconnect_zosConnectServiceRestClientConnection</code> element does not exist, the values from the <code>zosconnect_zosConnectServiceRestClient</code> element will be used.
connectionTimeout	A period of time with millisecond precision	The connection timeout specifies the amount of time that the client will attempt to establish a connection to the remote endpoint before it times out. If the timeout value is set to 0, the client will attempt to open a connection indefinitely. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.
host	string	IP address, domain name server (DNS) host name with domain name suffix, or just the DNS host name, used to route the request.

Attribute name	Data type	Description
httpMethod	string	Name of the HTTP method to be used when routing HTTP requests. If no method is specified, the method used is the one in the original request. The valid methods are GET, PUT, POST, OPTIONS, and DELETE.
id	string	A unique configuration ID.
port	string	Port used for routing HTTP or HTTPS requests.
receiveTimeout	A period of time with millisecond precision	The receive timeout specifies the amount of time that the client will wait for a response from the remote endpoint before it times out. If the timeout value is set to 0, the client will wait for a response indefinitely. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.
sslCertsRef	string	An SSL repertoire with an ID, a defined keystore, and truststore.
uri	string	URI that identifies the resource to contact when routing HTTP requests. If no URI is specified everything after the port number from the original request is used.

Sub elements

zosconnect_zosConnectServiceRestClient > basicAuth

Description: Reference name that identifies the basic authentication data to be used for connecting to a remote REST endpoint.

Required: false

Attribute name	Data type	Default value	Description
password	Reversably encoded password (string)		Password of the user under which the request will be routed. The value can be stored in clear text or encoded. It is recommended that the password be encoded. To do so, use the securityUtility shipped with WebSphere Liberty profile.
userName	string		Name of the user under which the request will be routed.

zosconnect_zosConnectServiceRestClientConnection

Allows requests to be routed from z/OS Connect EE to a remote REST endpoint.

Attribute name	Data type	Default value	Description
allowChunking	boolean	true	Allow chunking on messages greater than 4KB.
basicAuthRef	A reference to top level zosconnect_zosConnectService RestClientBasicAuth element (string).		Reference name that identifies the basic authentication data to be used for connecting to a remote REST endpoint.
connectionTimeout	A period of time with millisecond precision	30s	The connection timeout specifies the amount of time that the client will attempt to establish a connection to the remote endpoint before it times out. If the timeout value is set to 0, the client will attempt to open a connection indefinitely. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.
host	string		IP address, domain name server (DNS) host name with domain name suffix, or just the DNS host name, used to route the request.
id	string		A unique configuration ID.
port	string		Port used for routing HTTP or HTTPS requests.
receiveTimeout	A period of time with millisecond precision	60s	The receive timeout specifies the amount of time that the client will wait for a response from the remote endpoint before it times out. If the timeout value is set to 0, the client will wait for a response indefinitely. Specify a positive integer followed by a unit of time, which can be hours (h), minutes (m), seconds (s), or milliseconds (ms). For example, specify 500 milliseconds as 500ms. You can include multiple values in a single entry. For example, 1s500ms is equivalent to 1.5 seconds.
sslCertsRef	string		An SSL repertoire with an ID, a defined keystore, and truststore.

Sub elements

zosconnect_zosConnectServiceRestClientConnection > basicAuth

Description: Reference name that identifies the basic authentication data to be used for connecting to a remote REST endpoint.

Required: false

Attribute name	Data type	Default value	Description
password	Reversably encoded password (string)		Password of the user under which the request will be routed. The value can be stored in clear text or encoded. It is recommended that the password be encoded. To do so, use the securityUtility shipped with WebSphere Liberty profile.
username	string		Name of the user under which the request will be routed.

Attribute name	Data type	Default value	Description
applName	string		The name of the application that requests and uses the PassTickets.

zosconnect_zosConnectServiceRestClientBasicAuth

Basic authentication data for connecting to a remote REST endpoint.

Attribute name	Data type	Default value	Description
id	string		A unique configuration ID.
password	Reversably encoded password (string)		Password of the user under which the request will be routed. The value can be stored in clear text or encoded. It is recommended that the password be encoded. To do so, use the securityUtility shipped with WebSphere Liberty profile.
userName	string		Name of the user under which the request will be routed.
applName	string		The name of the application that requests and uses the PassTickets.

zosLocalAdapters (WOLA)

WebSphere Optimized Local Adapters.

Attribute name	Data type	Default value	Description
useCicsTaskUserId	boolean	false	If security is enabled in the WOLA client, this setting controls which user ID is propagated to this z/OS Connect EE server when a WOLA request is sent from the client to the server. If you enable this setting, the user ID of the task that makes the request is propagated. If you do not enable this setting, the user ID of the client address space is propagated. This setting is applicable only to CICS clients.
wolaGroup	string		The WOLA group name is the first part of the 3-part WOLA name. The name cannot contain lowercase letters.
wolaName2	string		The 2nd part of the 3-part WOLA name. The name cannot contain lowercase letters.
wolaName3	string		The 3rd part of the 3-part WOLA name. The name cannot contain lowercase letters.

Command reference

Syntax of the z/OS Connect EE commands.

apideploy command syntax

Deploy or undeploy APIs

Syntax

```
apideploy -deploy -a <path to api archiveFile> -p <path to apiDeploy location> [-w]
```

```
apideploy -undeploy -a <path to api archiveFile> -p <path to apiDeploy location>
```

Parameters

-deploy

Specifies the API deployment task.

-undeploy

Specifies the API undeployment task.

-a

Specifies the relative or absolute path to the API archive file. The archive file has a file type of .aar.
The path cannot include parentheses and soft links are not supported.

-p

Specifies the relative or absolute path to the API deployment location for the server. For example,
<WLP_USER_DIR>/servers/<serverName>/resources/zosconnect/apis. The path cannot
include parentheses and soft links are not supported.

-w

If specified, the API replaces the API, if it is already deployed, with the new version. The API identifier
is the API name that is specified in the z/OS Connect EE API toolkit when you create the API.

Examples

The following example deploys the API that is defined in the goodhealth.aar file by specifying an
absolute path to the .aar file and an absolute path to the API deployment directory.

```
apideploy -deploy -a /usr/lpp/myAPIs/goodhealth.aar  
-p <WLP_USER_DIR>/servers/<serverName>/resources/zosconnect/apis
```

The following example redeploys an API by specifying the -w parameter:

```
apideploy -deploy -a /usr/lpp/myAPIs/goodhealth.aar  
-p <WLP_USER_DIR>/servers/<serverName>/resources/zosconnect/apis -w
```

The following example undeploys the API that is defined in the goodhealth.aar file by specifying an
absolute path to the .aar file and an absolute path to the API deployment directory.

```
apideploy -undeploy -a /usr/lpp/myAPIs/goodhealth.aar  
-p <WLP_USER_DIR>/servers/<serverName>/resources/zosconnect/apis
```

MVS system MODIFY commands

MVS system **MODIFY** commands are available for use with z/OS Connect EE servers.

Liberty provides **MODIFY** commands for tasks such as configuring trace, requesting dumps, pausing HTTP
endpoints, or refreshing keystores, including SAF key rings. For more information, see [Liberty: Modify
commands on z/OS](#) and [Pausing and resuming a Liberty server from the z/OS console](#).

In the z/OS Connect EE server, you can issue the following command to get help with all the MODIFY
commands available.

```
modify <jobname>.<identifier>,help
```

z/OS Connect Enterprise Edition provides a MODIFY command to refresh the z/OS Connect EE server
artifacts. For more information, see [“The MODIFY command” on page 473](#)

MODIFY command syntax

Use the **MODIFY** command to refresh the z/OS Connect EE server artifacts such as Bind files, policies, .aar, .sar, and .ara files, and the `server.xml` configuration file.

Syntax

```
modify jobname.identifier,zcon [,refresh] | [cleartokencache] | [clearsafcache]
```

Parameter usage

jobname

Specifies the job name of the target server.

identifier

Specifies an identifier to distinguish between multiple jobs that use the same job name.

zcon

Must be included and written as shown.

refresh

Specifies that the z/OS Connect EE server artifacts be refreshed. These artifacts include Bind files, .aar, .sar, and .ara files, and the `server.xml` configuration file.

Note:

1. To update the configuration file, the configuration must contain the statement `<config updateTrigger="mbean" />`.
2. If an API or service is invoked while a refresh is in progress, interceptor calls might not occur as expected. For example, you change the `server.xml` file to add an interceptor, and issue the `refresh` command. While the refresh occurs, a service is invoked. The `preInvoke` interceptor method is not called because the refresh is not finished. After the refresh is complete, the `postInvoke` interceptor method for the service is called because it now exists.
3. When you use the **MODIFY** command to refresh the z/OS Connect EE server artifacts, the value of the `updateTrigger` attribute on the `zosconnect_apiRequesters`, `zosconnect_policy`, `zosconnect_services`, `zosconnect_zosConnectAPIs`, and `zosconnect_zosConnectDataXform` elements are ignored. Leave these values to default (`updateTrigger="disabled"`) to prevent changes from other update mechanisms.

cleartokencache

Causes all OAuth 2.0 access tokens and JWTs (either generated by an external authentication server or locally generated) to be cleared from the cache.

clearsafcache

Causes the SAF cache to be cleared. The SAF cache contains SAF user IDs and any associated RACF groups in which the user ID resides. The SAF cache is only applicable to API requester, and only when ID assertion is enabled.

Examples

The following example command refreshes the artifacts that have new updates.

```
modify zosconn01.1023,zcon,refresh
```

The following command clears the OAuth 2.0 access token and JWT cache:

```
modify zosconn01.1023,zcon,cleartokencache
```

The following command clears the SAF cache:

```
modify zosconn01.1023,zcon,clearsafcache
```

zconbt command syntax

The **zconbt** command starts the build toolkit tool. You can use the build toolkit to generate archive files for services, APIs or API requesters.

Syntax

Use the **zconbt** command to generate a service archive (.sar) or API archive (.aar) file. For API requesters, use the **zconbt** command to generate the required artifacts, including the API requester archive (.ara) file.

To generate a service archive (.sar) or an API requester archive (.ara) file from a properties file, use the following syntax:

```
zconbt -p=<filename>.properties -f=<filename>.sar
```

```
zconbt -p=<filename>.properties -f=<filename>.ara
```

To generate a service archive (.sar) or API archive (.aar) file from an API toolkit project directory, use the following syntax:

```
zconbt -pd=<projectDirectory> -od=<projectDirectory>
```

```
zconbt -pd=<projectDirectory> -f=<filename>.sar
```

```
zconbt -pd=<projectDirectory> -f=<filename>.aar
```

Note: Depending on the service provider, the steps and the tools to create a service might differ. For example, for the IMS service provider, you must use the API toolkit to create a service. If an automated process is required for service archive file generation such as in a DevOps environment, you can use the build toolkit to generate the IMS service archive file from the API toolkit service project directory. For some service providers, a service must be generated by using the build toolkit with a properties file as the input. For more information, see [“Creating services” on page 487](#).

Parameter usage

-p | --properties

Specifies the name of the properties file that is used to create the service archive file or artifacts for API requester.

Note: The properties file must use UTF-8 encoding.

-pd | --projectDirectory

Added in Build toolkit V1.1. Specifies the path and project directory name that is used to create the service archive file or API archive file. Use this parameter for service projects or API projects created in API toolkit.

-f | --file

Specifies the path and file name of the service archive file, the API archive file, or the API requester archive file to be generated. The file type is .sar, .aar, or .ara.

-od | --outputDirectory

Added in Build toolkit V1.1. Specifies the path and directory name for the output service archive file or API archive file to be generated. The service or API archive file inherits the same name as the service or API with a .sar or .aar file extension.

-r | --reportError

If specified, returns more specific error response codes. 0 - Success, 4 - Warning, 8 - Error.

-v | --verbose

If specified, returns more information when the trace utility is running.

-h | --help

If specified, returns list of available options that the user can enter.

Examples: Service archive generation

- The following example command uses the values in the `health.properties` file to generate the service archive.

```
zconbt --properties=health.properties --file=./health.sar
```

- The following example commands use files in service project directories to generate service archive files in the SARs output directory.

```
zconbt --projectDirectory=./u/serviceProjects/HospitalService --outputDirectory=./u/SARs
```

```
zconbt --projectDirectory=./u/serviceProjects/CatalogService --outputDirectory=./u/SARs
```

- The following example commands use files in service project directories to generate service archive files with specified names.

```
zconbt --projectDirectory=./u/serviceProjects/HospitalService --file=./u/SARs/Hospital.sar
```

```
zconbt --projectDirectory=./u/serviceProjects/CatalogService --file=./u/SARs/Catalog.sar
```

For more information about service archive generation depending on the service providers, see [“Generating service archives for DevOps” on page 560](#).

Examples: API archive generation

- The following example commands use files in API project directories to generate API archive files in the AARs output directory.

```
zconbt --projectDirectory=./u/apiProjects/lookupAPI --outputDirectory=./u/AARs
```

```
zconbt --projectDirectory=./u/apiProjects/purchaseAPI --outputDirectory=./u/AARs
```

- The following example commands use files in API project directories to generate API archive files with specified names.

```
zconbt --projectDirectory=./u/apiProjects/lookupAPI --file=./u/AARs/lookup.aar
```

```
zconbt --projectDirectory=./u/apiProjects/purchaseAPI --file=./u/AARs/purchase.aar
```

Example: API requester artifacts generation

The following example command uses the values in the `watson.properties` file to generate artifacts for API requester.

```
zconbt --properties=watson.properties --file=./watson/archives/Watson.ara
```

For more information about how to prepare the properties file and the generated output, see [“Generating artifacts for an API requester from the command line” on page 618](#).

zosconnect command syntax

Use the `zosconnect` command to create a z/OS Connect EE server, or perform other tasks on that server.

Tip: Enter the `zosconnect` command from the `<installation_path>/bin` directory. For example, `/usr/lpp/IBM/zosconnect/v3r0/bin`

Syntax

```
zosconnect <action> <serverName> <options>
```

Parameters

<action>

The task that you want to perform:

create

Creates a z/OS Connect EE server. Specify the server name and optionally, the name of a template.

For example, `zosconnect create <serverName> --template=<template_name>`

start

Launches the named server as a USS background process. Progress can be seen in the `messages.log` file. You can optionally specify the `--clean` option to clear the cache. For more information, see “[Starting and stopping z/OS Connect EE](#)” on page 471.

Note: Do not use **start** when running z/OS Connect EE as a started procedure.

For example, `zosconnect start <serverName> [--clean]`

serverName

The name of the server on which the action is performed. When `serverName` is not provided the name `defaultServer` is used.

options

--clean

Deletes all persistent cached information that is related to the specified server instance, which includes OSGi resolver metadata and persistent OSGi bundle data. If you use this option, the server will be required to recompute any cached data at the next startup, which might take more time than a restart that can reuse cached data.

Use only with the **start** parameter.

The **--clean** option is not necessary for normal operation. IBM service might request that you use this option when providing an interim fix, or if there is a suspected problem with the cached data. This option might also be necessary if you are developing a product extension, and you are either updating OSGi manifests or planning to clear persistent OSGi bundle data.

--template=templateName

Use only with the **create** parameter.

Specifies the name of the template to be used to create the server. See “[Supplied server templates](#)” on page 825 for a description of the templates that are supplied with z/OS Connect EE.

Examples

```
zosconnect create myserver --template=zosconnect:default
zosconnect start myserver --clean
```

zconsetup command syntax

Syntax

```
zconsetup install
```

Parameters

install

Specifies that the product extension directories be created.

Example

```
zconsetup install
```

Creating a CICS service for C or top-down

If you cannot use the API toolkit to create your CICS service because, for example, you have a C program, your source of input is JSON schema, or you do not have access to the API toolkit, you can manually prepare the required properties file that the build toolkit needs to generate the service archive.

Note: The use of BAQLS2JS and BAQJS2LS to generate service archives is not supported when using the CICS service provider.

The service archive created by the build toolkit can be deployed by copying it to the services directory.

Service archives can be created from either existing language structures or from JSON schemas. The following tables describe the build toolkit properties, including those which are specific to starting with either language structures or JSON schemas. Define the following properties to build your service archive:

Table 107. Common properties for CICS service provider		
Property	Importance	Description
provider	Required	Must be set to <code>cics</code> to build a service archive for the CICS service provider.
name	Required	The name of the service.
version	Required	The version of the service.
description	Optional	A description of the service.
connectionRef	Required	The id of the <code>zosconnect_cicsIpicConnection</code> element that defines the connection to CICS. See Notes® 1 and 2 .
language	Required	The language of the CICS program. Valid values are COBOL C CPP PLI-ENTERPRISE PLI-OTHER.
program	Required	The CICS program to invoke.
programInterface	Required	The program interface for the CICS program. Valid values are COMMAREA CHANNEL. This value must be set to CHANNEL to use context containers. See Table 4.
ccsid	Optional	Specifies the CCSID that is used at run time to encode character data in COMMAREA and BIT container application data structures. The default is 037 (EBCDIC).

Table 107. Common properties for CICS service provider (continued)

Property	Importance	Description
characterVarying	Optional	<p>Specifies how character fields in the language structure are mapped. A character field in COBOL is a Picture clause of type X, for example PIC(X) 10; a character field in C/C++ is a character array. You can select these options:</p> <p>NO Character fields are mapped to a JSON string and are processed as fixed-length fields. The maximum length of the data is equal to the length of the field. This value does not apply to Enterprise and Other PL/I language structures.</p> <p>NULL Character fields are mapped to a JSON string and are processed as null-terminated strings. z/OS Connect EE adds a terminating null character when transforming from a JSON payload. The maximum length of the character string is calculated as one character less than the length indicated in the language structure. NULL is the default value for the characterVarying property for C and C++. This value does not apply to Enterprise and Other PL/I language structures.</p> <p>COLLAPSE Character fields are mapped to a JSON string. Trailing white space in the field is not included in the JSON payload. The request JSON payload is parsed to remove all leading, trailing, and embedded white space. COLLAPSE is the default value for the characterVarying property for COBOL and PL/I.</p> <p>BINARY Character fields are mapped to a JSON string containing base64 encoded data and are processed as fixed-length fields.</p> <p>Messages with the DFH prefix might refer to this property as CHAR-VARYING.</p>
dataScreening	Optional	<p>Specifies whether application supplied data is screened for errors. The default is ENABLED.</p> <p>ENABLED Any application-supplied runtime data that is inconsistent with the language structure, is treated as an error and message DFHPI1010 is issued. An error response is returned to the application.</p> <p>DISABLED Data values in application-supplied runtime data that are inconsistent with the language structure are replaced by default values. For example, a zero replaces a bad value in a numeric field. Message DFHPI1010 is not issued and a normal response is returned to the application. This feature can be used to avoid INVALID_PACKED_DEC and INVALID_ZONED_DEC error responses that are generated from uninitialized output fields.</p> <p>Messages with the DFH prefix might refer to this property as DATA-SCREENING.</p>

Table 107. Common properties for CICS service provider (continued)

Property	Importance	Description
dataTruncation	Optional	<p>Specifies if variable length data is tolerated in a fixed-length field structure. The default is DISABLED.</p> <p>ENABLED If the data is less than the fixed length that CICS is expecting, z/OS Connect EE rejects the truncated data and issues an error message.</p> <p>DISABLED If the data is less than the fixed length that CICS is expecting, z/OS Connect EE tolerates the truncated data and processes the missing data as null values.</p> <p>Messages with the DFH prefix might refer to this property as DATA-TRUNCATION.</p>
languageStructureCodeP	Optional	Specifies the Java character set name for the source language structures specified by requestStructure , responseStructure or referenced from a channel description document. For example, for IBM Latin-1 EBCDIC, the value is IBM037. When this property is used both the requestStructure and responseStructure must name members in the same partitioned dataset.
structure	Optional	Specifies the names of the high-level language structures that are contained in the file or partitioned data set members that are specified in the requestStructure and responseStructure , or generatedRequestStructure and generatedResponseStructure properties. This property is valid only for C and C++ languages. The value of this parameter has the format <i>(request,response)</i> . For example the value <i>(ReqStruct,RespStruct)</i> would specify a request structure named <i>ReqStruct</i> in the file or PDS specified by requestStructure and a response structure named <i>RespStruct</i> in the file or PDS specified by responseStructure . The default request structure name is DFHREQUEST and the default response structure name is DFHRESPONSE.
transid	Optional	A CICS transaction name. This property will override the transid set on the connection for this service. Additionally this property can be set on the service element. The usage of this property varies depending on the type of connection to CICS and the transidUsage property. For more information, see “ zosconnect_cicsIpicConnection ” on page 763 and “ zosconnect_services ” on page 774. See Note 3.

Table 107. Common properties for CICS service provider (continued)

Property	Importance	Description
transidUsage	Optional	<p>Specifies how the transid property value is used. This property only applies to CICS IPIC connections and will override the <code>transidUsage</code> set on the <code>zosconnect_cicsIpicConnection</code> for this service. Additionally this property can be set on the <code>service</code> element. For more information, see “zosconnect_cicsIpicConnection” on page 763 and “zosconnect_services” on page 774.</p> <p>EIB_ONLY The transid property specifies the name of the CICS transaction that appears in the CICS exec interface block (EIB); the EIBTRNID field contains the value of the transid property. The called CICS program runs under the default mirror transaction CSMI.</p> <p>EIB_AND_MIRROR The transid property specifies the name of the CICS transaction under which the called CICS program runs. The transaction must be defined in the CICS region, and the transaction definition must specify the mirror program, DFHMIRS. The value specified by the transid property is available to the called CICS program for querying the transaction ID. The value of the transid property also appears in the EIBTRNID field of the CICS EIB.</p> <p>See Note 3.</p>
useContextContainers	Optional	<p>Valid values are <code>true</code> or <code>false</code>:</p> <p>true Context containers are sent to CICS.</p> <p>false Context containers are not sent to CICS.</p> <p>useContextContainers can only be used when programInterface is set to CHANNEL</p> <p>For more information, see “Context containers” on page 230.</p>
httpHeaders	Optional	<p>A comma-separated list of header names to include in the BAQHTTPHEADERS context container if they are present on the HTTP request. For example, <code>httpHeaders=MyHeader,SecondHeader</code>.</p> <p>httpHeaders can only be used when useContextContainers is set to <code>true</code>.</p>

If you generate the service archive from a language structure, as indicated by the **requestStructure** property, the following properties are also available.

Table 108. Properties specific to generating a service archive from a language structure

Property	Importance	Description
requestStructure	Required	Specifies the relative or absolute path to the file containing the language structure for the request, or the channel description document file describing the request channel. See notes 4, 5, and 7.

Table 108. Properties specific to generating a service archive from a language structure (continued)

Property	Importance	Description
responseStructure	Required	Specifies the relative or absolute path to the file containing the language structure for the response or the channel description document file describing the response channel. See notes 4, 5, and 7.
characterOccurs	Optional	<p>Specifies how character arrays in the language structure are mapped. For example, PIC X OCCURS 20. This property is only for use by COBOL language. The default is STRING.</p> <p>ARRAY Character arrays are mapped to a JSON array. This means that every character is mapped as an individual JSON value.</p> <p>STRING Character arrays are mapped to an JSON string. This means that the entire COBOL array is mapped as a single JSON string.</p> <p>Messages with the DFH prefix might refer to this property as CHAR-OCCURS.</p>
characterUsage	Optional	<p>In COBOL, the national data type, PIC N, can be used for UTF-16 or DBCS data. This setting is controlled by the NSYMBOL compiler option. You must set the characterUsage property on the build toolkit to the same value as the NSYMBOL compiler option to ensure that the data is handled appropriately. This is typically set to characterUsage=NATIONAL when you use UTF-16.</p> <p>DBCS Data from PIC (n) fields is treated as DBCS encoded data.</p> <p>NATIONAL Data from PIC (n) fields is treated as UTF-16 encoded data.</p> <p>The default is NATIONAL. Messages with the DFH prefix might refer to this property as CHAR-USAGE.</p>
dateTime	Optional	<p>Specifies if potential ABSTIME fields in the high-level language structure are mapped as timestamps.</p> <p>PACKED15 Packed decimal fields of length 15 (8 bytes) are treated as CICS ABSTIME fields, and mapped as timestamps.</p> <p>UNUSED Packed decimal fields of length 15 (8 bytes) are not treated as timestamps.</p> <p>The default is UNUSED.</p>

Table 108. Properties specific to generating a service archive from a language structure (continued)

Property	Importance	Description
truncateNullArrays	Optional	<p>Specifies how structured arrays are processed.</p> <p>ENABLED z/OS Connect EE will attempt to recognize empty records within an array (See truncateNullArrayValues for more information about identifying empty records). If five consecutive empty array records are detected, the array is truncated at the first such record when generating JSON. This truncation capability is only enabled for arrays with structured content, arrays of simple primitive fields are not subject to truncation. Truncation of arrays can result in a more concise representation of the data in JSON, but is not without risk. If five consecutive data records are misidentified as uninitialised storage (perhaps because they legitimately contain low values), data loss can be experienced. If truncateNullArrays is enabled and truncateNullArrayValues is not set, then the default value for truncateNullArrayValues is used.</p> <p>DISABLED z/OS Connect EE will not attempt to truncate arrays.</p> <p>The default is DISABLED. Messages with the DFH prefix might refer to this property as TRUNCATE-NUL-ARRAYS</p>
truncateNullArrayValue	Optional	<p>Specifies which values are treated as empty for truncateNullArrays processing. If all of the bytes of storage within a record of a structured array contain nulls then the entire record is considered to be empty. One or more of the NULL, SPACE and ZERO values can be specified in a comma separated list. Any matching combination of the selected bytes within a structured array record will cause the entire record to be identified as empty. If truncateNullArrayValues has a value defined, then truncateNullArrays must be ENABLED.</p> <p>NULL The null character (0x00) is treated as empty.</p> <p>SPACE An SBCS EBCDIC Space (0x40) is treated as empty.</p> <p>ZERO An unsigned zoned decimal zero (0xF0) is treated as empty.</p> <p>The default is NULL. Messages with the DFH prefix might refer to this property as TRUNCATE-NUL-ARRAY-VALUES.</p>

If you generate the service archive from a JSON schema, as indicated by the **requestSchema** property, the following properties are also available.

Table 109. Properties specific to generating a service archive from a JSON schema

Property	Importance	Description
requestSchema	Required	Specifies the relative or absolute path to the JSON schema for the request. See notes 5 and 6 .

Table 109. Properties specific to generating a service archive from a JSON schema (continued)

Property	Importance	Description
responseSchema	Required	Specifies the relative or absolute path to the JSON schema for the response. See notes 5 and 6 .
generatedRequestStructure	Required	Specifies the relative or absolute path to the file or PDS that contains the high-level language structures for the request that is generated from the JSON schema. See notes 4 and 5 .
generatedResponseStructure	Required	Specifies the relative or absolute path to the file or PDS that contains the high-level language structures for the response that is generated from the JSON schema. See notes 4 and 5 .
characterMultiplier	Optional	<p>Specifies the number of bytes to allow for each character. The value of this property can be a positive integer in the range of 1 - 2147483647. All nonnumeric character-based mappings, are subject to this multiplier. Binary, numeric, zoned, and packed decimal fields are not subject to this multiplier.</p> <p>This property can be useful if, for example, you are planning to use DBCS characters where you might opt for a multiplier of 3 to allow space for potential shift-out and shift-in characters around every double-byte character at run time.</p> <p>When you set ccsid=1200 (indicating UTF-16), the only valid values for characterMultiplier are 2 or 4. When you use UTF-16, the default value is 2. Use characterMultiplier=2 when you expect application data to contain characters that require 1 UTF-16 encoding unit. Use characterMultiplier=4 when you expect application data to contain characters that require 2 UTF-16 encoding units.</p> <p>Note: Setting characterMultiplier to 1 does not preclude the use of DBCS characters, and setting it to 2 does not preclude the use of UTF-16 surrogate pairs. However, if wide characters are routinely used then some valid values will not fit into the allocated field. If a larger characterMultiplier value is used, it can be possible to store more characters in the allocated field than are valid in the JSON. Care must be taken to conform to the appropriate range restrictions.</p> <p>The default is 1. Messages with the DFH prefix might refer to this property as CHAR-MULTIPLIER.</p>
characterVaryingLimit	Optional	Specifies the maximum size of binary data and variable-length character data that is mapped to the language structure. If the character or binary data is larger than the value specified in this property, it is mapped to a container and the container name is used in the generated language structure. The value can range from 0 to the default 32767 bytes.

Table 109. Properties specific to generating a service archive from a JSON schema (continued)

Property	Importance	Description
characterWhitespace	Optional	<p>Specifies how white space in values of type string is handled by z/OS Connect EE.</p> <p>COLLAPSE Leading, trailing, and embedded white space is removed and all tabs, new lines, and consecutive spaces are replaced with single space characters.</p> <p>REPLACE Any tabs or new lines are replaced with the appropriate number of spaces.</p> <p>PRESERVE Retains any white space in the data value.</p> <p>The default value is COLLAPSE.</p>
dateTime	Optional	<p>Specifies how JSON date-time format properties are mapped to the language structure.</p> <p>PACKED15 JSON date-time format properties are processed as a time stamp and is mapped to CICS ABSTIME format.</p> <p>STRING JSON date-time format properties are processed as text.</p> <p>The default is PACKED15.</p>
defaultCharacterMaxLen gth	Optional	Specifies the default array length of character data in characters for mappings where no length is implied in the JSON schema. The value of this property can be a positive integer in the range of 1 - 2147483647. The default is 255.

Table 109. Properties specific to generating a service archive from a JSON schema (continued)

Property	Importance	Description
inlineMaxOccursLimit	Optional	<p>Specifies whether inline variable repeating content is used based on the maxItems JSON schema keyword. Variably repeating content that is mapped inline is placed in the current container with the rest of the generated language structure. The variably repeating content is stored in two parts, as a counter that stores the number of occurrences of the data and as an array that stores each occurrence of the data. The alternative mapping for variably repeating content is container-based mapping, which stores the number of occurrences of the data and the name of the container where the data is placed. Storing the data in a separate container has performance implications that might make inline mapping preferable.</p> <p>The value of inlineMaxOccursLimit can be a positive integer in the range of 0 - 32767. A value of 0 indicates that inline mapping is not used. A value of 1 ensures that optional elements are mapped inline. If the value of the maxOccurs property is greater than the value of inlineMaxOccursLimit, container-based mapping is used; otherwise inline mapping is used.</p> <p>When deciding whether you want variably repeating lists to be mapped inline, consider the length of a single item of recurring data. If few instances of long length occur, container-based mapping is preferable; if many instances of short length occur, inline mapping is preferable.</p> <p>The default is 1.</p>

Table 109. Properties specific to generating a service archive from a JSON schema (continued)

Property	Importance	Description
mappingOverrides	Optional	<p>Specifies whether the default behavior is overridden when generating language structures. One or more of the options can be specified in a comma-separated list.</p> <p>SAME-AS-MAPPING-LEVEL This option generates language structures in the default style.</p> <p>UNDERSCORES-AS-HYPHENS For COBOL only. This option converts any underscores in the JSON schema to hyphens, rather than the character X, to improve the readability of the generated COBOL language structures. If any field name clashes occur, the fields are numbered to ensure that they are unique.</p> <p>INTEGER-AS-PIC9 For COBOL only. This option generates language structures that contain integer values from the JSON schema as numerals rather than alphanumeric characters.</p> <p>LESS-DUP-NAMES This option generates non-structural structure field names with _value at the end of the name to enable direct referencing to the field. For example, in the following PL/I language structure, when mappingOverrides=LESS-DUP-NAMES is specified, level 12 field streetName is suffixed with _value:</p> <pre>09 streetName, 12 streetName CHAR(255) VARYING UNALIGNED, 12 filler BIT (7), 12 attr_nil_streetName_value BIT (1),</pre> <p>The resulting structure is as follows:</p> <pre>09 streetName, 12 streetName_value CHAR(255) VARYING UNALIGNED, 12 filler BIT (7), 12 attr_nil_streetName_value BIT (1),</pre> <p>The default is UNDERSCORES-AS-HYPHENS for COBOL and SAME-AS-MAPPING-LEVEL for other languages.</p>
nameTruncation	Optional	<p>Specifies whether JSON names are truncated from the left or the right. The build toolkit truncates JSON names to the appropriate length for the high-level language specified; by default names are truncated from the right.</p> <p>RIGHT Names are truncated from the right.</p> <p>LEFT Names are truncated from the left.</p>

Table 109. Properties specific to generating a service archive from a JSON schema (continued)

Property	Importance	Description
wideComp3	Optional	<p>Controls the maximum size of the packed decimal variable length in the generated COBOL or PL/I language structure.</p> <p>YES build toolkit supports the maximum size of 31 when generating the COBOL language structure type COMP-3.</p> <p>NO build toolkit limits the packed decimal variable length to 18 when generating the COBOL language structure type COMP-3. If the packed decimal size is greater than 18, message DFHPI9022W is displayed to indicate that the specified type is being restricted to a total of 18 digits.</p> <p>The default is YES.</p>

Sample properties file for the CICS service provider using a C structure

```

name=service2
provider=cics
version=1.0
description=An example CICS service for a C structure
program=CICSPGM3
connectionRef=cicsConn
language=C
programInterface=COMMAREA
characterVarying=NO
structure=(CommAreaDetail,CommAreaDetail)
requestStructure=/u/myhome/REQUEST
responseStructure=/u/myhome/RESPONSE

```

Sample properties file for the CICS service provider using JSON

```

provider=cics
name=service3
version=1.0
description=An example CICS service
program=CICSPROG
language=COBOL
programInterface=CHANNEL
requestSchema=request.json
responseSchema=response.json
generatedRequestStructure=request.cpy
generatedResponseStructure=response.cpy
connectionRef=cicsConn

```

Note:

1. CICS services can only be deployed by adding them to the services directory. CICS .sar files reference a zosconnect_cicsIpicConnection element in the server.xml configuration file.
2. The service is initialized separately from the zosconnect_cicsIpicConnection element, so the service might show as available but without a connection. If the connection is not available to the CICS service either because it is not configured or there is a configuration error, the following message is logged when the service is invoked:
 BAQR0655E: Service {0} in service archive file {1} references the CICS connection {2} which is either not defined or defined incorrectly.
3. Specifying the **transid** or **transidUsage** properties when creating a service archive will override the equivalent attributes on the connection the service uses. The properties **transid** and **transidUsage** can additionally be defined on a service element under the zosconnect_services element of the

`server.xml` configuration file to set different transactions in different environments with a single service archive. For more information, see “[zosconnect_services](#)” on page 774.

4. If you run the build toolkit on z/OS to generate a service archive for a CICS COMMAREA program, the values **requestStructure** and **responseStructure** can be a UNIX System Services file or a member of a partitioned dataset. To specify a member of a partitioned dataset use the syntax `//'partitionDatasetName(member)'`. If you run the build toolkit on z/OS to generate a service archive for a CICS channel program, these values must specify a channel description document. The location attribute of the structure element in the channel description document can also support a UNIX System Services file or a member of a partitioned dataset. See the **languageStructureCodePage** property to indicate the code page of the z/OS partitioned dataset member
5. If you do not specify an absolute path for the referenced file, the build toolkit uses a relative path from the directory in which the **zconbt** command is run.
6. All JSON schema files must use UTF-8 encoding.
7. Language structure files must use the default code page for the system where the build toolkit is run.

You can now use the build toolkit to create the service archive. For more information, see “[Generating service archives for DevOps](#)” on page 560.

Related concepts

[“Context containers” on page 230](#)

In some situations, CICS programs need information about the context in which they were called. The CICS service provider sends this information to CICS in context containers.

Related tasks

[“Sample for creating a CICS service with a properties file” on page 810](#)

This sample walks you through the steps to create a CICS service by preparing a properties file for the service and generating the service archive using the build toolkit.

Related reference

[“zconbt command syntax” on page 795](#)

The **zconbt** command starts the build toolkit tool. You can use the build toolkit to generate archive files for services, APIs or API requesters.

Creating a channel description document

Channel description documents are used by the z/OS Connect EE build toolkit when building a service archive file for the CICS service provider to define the channel interface to your CICS program.

Use an XML editor to create the channel description document. The document must conform to the following schema:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.ibm.com/xmlns/prod/CICS/channel"
xmlns:tns="http://www.ibm.com/xmlns/prod/CICS/channel" elementFormDefault="qualified">
  <element name="channel"> NOTE 1
    <complexType>
      <sequence>
        <element name="container" maxOccurs="unbounded" "unbounded" minOccurs="0"> NOTE 2
          <complexType>
            <sequence>
              <element name="structure" minOccurs="0"> NOTE 3
                <complexType>
                  <attribute name="location" type="string" use="required"/>
                  <attribute name="structure" type="string" use="optional"/>
                </complexType>
              </element>
            </sequence>
            <attribute name="name" type="tns:name16Type" use="required"/>
            <attribute name="type" type="tns:typeType" use="required"/>
            <attribute name="use" type="tns:useType" use="required"/>
          </complexType>
        </element>
      </sequence>
      <attribute name="name" type="tns:name16Type" use="optional" />
    </element>
  </schema>
```

```

</complexType>
</element>
<simpleType name="name16Type">
  <restriction base="string">
    <maxLength value="16"/>
  </restriction>
</simpleType>
<simpleType name="typeType">
  <restriction base="string">
    <enumeration value="char"/>
    <enumeration value="bit"/>
  </restriction>
</simpleType>
<simpleType name="useType">
  <restriction base="string">
    <enumeration value="required"/>
    <enumeration value="optional"/>
  </restriction>
</simpleType>
</schema>

```

Note:

1. This name="channel" element represents a CICS channel. This element has an optional name attribute to specify the name of the CICS channel.
2. This name="container" element represents a CICS container within the channel. You must specify one name="container" element for each container in the channel. Each element must specify the attributes name with the name of the container, type with the type of the container (either char or bit) and use to indicate if the container is required or optional.
3. A structure element can only be used with bit mode containers. The location attribute indicates the location of a file that maps the contents of the container. The file can be a member of a partitioned dataset or a UNIX System Services file. The structure attribute can be used in C and C++ to indicate the name of the structure in the header file.
4. Do not use container names starting with BAQ or DFH, unless directed to do so by IBM.

The following example shows a channel called "myChannel" with three containers with different attribute values. Containers "cont1" and "cont2" are character containers, container "cont3" is a bit container whose content is defined in a language structure located in a partitioned data set member.

```

<channel name="myChannel" xmlns="http://www.ibm.com/xmlns/prod/CICS/channel">
  <container name="cont1" type="char" use="required"/>
  <container name="cont2" type="char" use="optional"/>
  <container name="cont3" type="bit" use="required">
    <structure location="//hlq.pdsname(member)"/>
  </container>
</channel>

```

Note: Context containers do not need to be included in the channel description document.

Sample for creating a CICS service with a properties file

This sample walks you through the steps to create a CICS service by preparing a properties file for the service and generating the service archive using the build toolkit.

Before you begin

Note: Although the primary use cases are when the source of input is neither COBOL or PL/I, this sample shows you how to recreate the CICS inquireSingle service for the Catalog Manager example application (COBOL) for ease of testing.

To create and test the inquireSingle service, ensure that the following tasks are complete:

1. Install and configure the CICS catalog manager example application. The procedure is described in the scenario ["Prepare the sample CICS application" on page 59](#)
2. Install z/OS Connect EE v3.0 or later. For more information, see ["Installing z/OS Connect EE" on page 196](#).

3. Configure and test your IPIC connection to CICS. The procedure is described in the scenario “Create a server and connect to CICS” on page 69 and installs relevant artifacts using the supplied configuration template `sampleCicsIpicCatalogManager`. This includes the `inquireService` service.
4. Use the z/OS Connect EE API toolkit to Stop the `inquireSingle` service, and then Remove the `inquireService` service.

About this task

In this sample we use the build toolkit to recreate the CICS `inquireSingle` service. The service archive file uses an existing CICS COBOL copybook from the catalog manager example application.

Procedure

In this procedure, substitute the Windows example for the platform of your choice. The build toolkit properties files must be encoded in UTF-8.

1. Create a local directory called: `C:/CICSSPscenario` on the Windows system.
2. Download the Catalog Manager sample copybook to the local directory `C:/CICSSPscenario` created in step 1.
The copybook is supplied by CICS TS in `hlq.SDFHSAMP` and is called `DFH0XCP4`, where `hlq` is the high-level qualifier (HLQ) of your CICS libraries. When you download the file, give it a file type of `cpx`, for example: `DFH0XCP4.cpx`.
3. Download the build toolkit compressed file, `zconbt.zip` as a binary file, to the same local directory, `C:/CICSSPscenario`.
The build toolkit is in the z/OS Connect EE `<installation_path>` and is called `zconbt.zip`.
4. Extract the `zconbt.zip` file into a `zconbt` sub directory of the `C:/CICSSPscenario` local directory.
The directory structure should be as follows.

```
C:/CICSSPscenario
  DFH0XCP4.cpx
  zconbt
    bin
    doc
    ...
  ...
```

5. Add the `zconbt` executable to your path. For example, enter the following command: `set PATH=C:/CICSSPscenario/zconbt/bin;%PATH%`
6. Create a service archive file for the Inquire Single service.
 - a) Create a build toolkit properties file called `InquireSingle.properties` in the `C:/CICSSPscenario` directory.
For example,

```
name=inquireSingle
provider=cics
description=CICS Catalog Manager Inquire Single service
version=1.0
program=DFH0XCMN
connectionRef=cicsConn
language=COBOL
programInterface=COMMAREA
requestStructure=C:/CICSSPscenario/DFH0XCP4.cpx
responseStructure=C:/CICSSPscenario/DFH0XCP4.cpx
```

- b) Run the build toolkit CLI to create the service archive file.

In a command prompt window, enter the following command:

```
zconbt --properties=C:/CICSSPscenario/InquireSingle.properties --file=C:/CICSSPscenario/InquireSingle.sar
```

Results

You can now deploy the service archive file by copying it to the services directory of your server.

Deploy the CICS catalog manager service manually

About this task

The inquireSingle service archive (.sar) file created in the previous task must be uploaded to the z/OS Connect EE server's services directory.

Procedure

FTP the service archive file: InquireSingle.sar in binary mode to the services directory.

This directory is defined on the zosconnect_services element of the server configuration file. In this example we are using the default location of \${server.config.dir}/resources/zosconnect/services. For example, /var/zosconnect/servers/catalogManager/resources/zosconnect/services.

Test the CICS catalog manager service

You can test that your newly created inquireSingle service is working by invoking the supplied catalog API directly from z/OS Connect EE API toolkit by using the **Try it out!** function in the Swagger UI that is embedded in the editor.

Procedure

1. Start CICS and ensure that the TCPIPSERVICE is open.
2. Start your z/OS Connect EE server. For more information, see [“Starting and stopping z/OS Connect EE” on page 471](#).
 - a) Check the messages.log file for the following messages that confirm that the services are installed.

BAQR7043I: z/OS Connect EE service archive placeOrder installed successfully.
BAQR7043I: z/OS Connect EE service archive inquireSingle installed successfully.
BAQR7043I: z/OS Connect EE service archive inquireCatalog installed successfully.

- b) Check the messages.log file for the following messages that confirm that the API is installed.

BAQR7000I: z/OS Connect API archive file catalog installed successfully.
3. In the **z/OS Connect EE Servers** view under the **APIs** folder, double-click on the catalog API.
The API is opened in the Swagger UI.
4. Click **List Operations** to see available operations in the API.

catalog

The screenshot shows the Swagger UI interface for the 'catalog' API. At the top, there is a header with the API name 'catalog' and tabs for 'default', 'Show/Hide', 'List Operations' (which is highlighted with a red box), and 'Expand Operations'. Below the header, there are three operation entries: a GET method for '/items', a GET method for '/items/{itemID}', and a POST method for '/orders'. At the bottom of the screen, there is a note: '[BASE URL: /catalogManager , API VERSION: 1.0.0]'.

Figure 121. Operations in the API

5. Test inquiring on an item by clicking **GET /items/{itemID}**.
 - a) Specify a value of 10 for the itemID.

The screenshot shows a REST API testing interface. At the top, it says "GET /items/{itemID} Response Class (Status 200) normal response". Below this, there are tabs for "Model" and "Example Value". The "Example Value" tab displays a JSON response body:

```
{
  "DFH0XCP4": {
    "CA_RETURN_CODE": 0,
    "CA_RESPONSE_MESSAGE": "string",
    "CA_INQUIRE_SINGLE": {
      "FILL_0": 0,
      "FILL_1": 0,
      "CA_SINGLE_ITEM": {
        "CA_SNGL_ITEM_REF": 0,
        "CA_SNGL_DESCRIPTION": "string",
        ...
      }
    }
  }
}
```

Below the JSON, it says "Response Content Type application/json". Under "Parameters", there are two entries:

Parameter	Value	Description	Parameter Type	Data Type
itemID	10		path	string
Authorization			header	string

At the bottom left is a "Try it out!" button.

Figure 122. Testing the GET request

b) Click **Try it out!**.

Information about the request URL, request headers, response body, response code, and response headers are provided. The response body contains the output message:

- "CA_SNGL_DESCRIPTION": "Ball Pens Black 24pk"

Results

You have verified that your newly created inquireSingle service works to CICS.

What to do next

Follow the steps in [“Create an API to invoke the CICS catalog manager services ” on page 115](#) to see how to create an API for yourself.

Creating an IBM MQ service by using a properties file

You can create an IBM MQ service by using either the API toolkit or passing a properties file to the build toolkit.

The preferred method is to use the API Toolkit as described in [“Creating a CICS, IMS or IBM MQ service” on page 488](#).

Alternatively, a service can be created by specifying a set of properties in a file and passing that file to the build toolkit. The supported set of properties are listed in [Table 110 on page 814](#).

Note: MQ services which have been created by using properties files support data transformation, but the maximum transformed payload for both request and reply messages must be less than, or equal to, 32 KB. If larger message sizes are needed, use the API toolkit.

When an IBM MQ service provider service archive file is generated and deployed to the z/OS Connect EE runtime, more properties can be specified in the `server.xml` configuration file to take effect at run time. Some of these properties can override the properties that were provided to the build toolkit. See [IBM MQ override properties](#).

For more information about running the build toolkit, see [“Generating service archives for DevOps” on page 560](#).

The following properties are supported by the build toolkit:

Table 110. Supported properties for creating a service with the build toolkit.

Property name	Importance	Property description
provider	Mandatory	Used to select the plug-in. Must be the value mq.
name	Mandatory	The name of the service. Can be any string value. For example, service1.
version	Mandatory	The version of the service. Can be any string value. For example, 1.0.
description	Optional	A description of the service.
language	Optional	If data transformation is required, this property defines the type of language structure that is used. If this property is specified, the following conditions apply: <ul style="list-style-type: none"> • The value must be one of either COBOL or PLI. • One or both of the <code>requestStructure</code> and <code>responseStructure</code> properties must be specified. • The <code>operationName</code> property must be specified. • When data transformations are used, the maximum supported length of the COBOL or PL/I data structure is 32 KB.
languageStructureCodePage	Optional	Specifies the Java character set name for the source language structures that are specified by <code>requestStructure</code> or <code>responseStructure</code> . For example, for IBM Latin-1 EBCDIC, the value is IBM037. This property is only applicable if the <code>requestStructure</code> and <code>responseStructure</code> are members of a partitioned data set. When this property is used, both the <code>requestStructure</code> and <code>responseStructure</code> must name members in the same partitioned data set.

Table 110. Supported properties for creating a service with the build toolkit. (continued)

Property name	Importance	Property description
requestStructure	Optional	<p>A path to a language structure to be used as the request payload. This attribute must be specified when the language property is specified with messagingAction set to mqput, or the replyDestination property is specified.</p> <p>The language structure must use the syntax of the language property. For example, if language=COBOL it must be a valid COBOL copybook.</p> <p>See Note.</p>
responseStructure	Optional	<p>A path to a language structure to be used as the response payload. This attribute must be specified when the language property is specified with messagingAction set to mqput, or the replyDestination property is specified.</p> <p>The language structure must use the syntax of the language property. For example, if language=COBOL it must be a valid COBOL copybook.</p> <p>See Note.</p>
operationName	Optional	<p>This property must be specified when the language property is specified. It defines the operation name that is used in the generated JSON schema. This property can be up to 8 characters long.</p>
ccsid	Optional	<p>This property is only supported if the language property is specified. It defines the integer code page of the data structures that are generated by the z/OS Connect EE data transformation code. The default value is 37.</p>

Table 110. Supported properties for creating a service with the build toolkit. (continued)

Property name	Importance	Property description
connectionFactory	Mandatory	Defines a JNDI name that is used to locate a connection factory that connects to a z/OS queue manager on the same LPAR as the z/OS Connect EE server or on a different LPAR. For more information, see JMS connection factory in the <i>WebSphere Application Server for z/OS Liberty</i> documentation.
destination	Mandatory	Defines a JNDI name that is used to locate an IBM MQ queue or topic. For one-way services, if <code>messagingAction=mqput</code> , then destination is the queue or topic that messages are sent to. If <code>messagingAction=mqget</code> , then destination is the queue that messages are destructively got from. For two-way services, request messages are sent to this queue. For more information, see the following topics in the <i>WebSphere Application Server for z/OS Liberty</i> documentation: JMS Queue (jmsQueue) if the destination is a queue or JMS Topic (jmsTopic) if the destination is a topic.
replyDestination	Optional	Defines a JNDI name that is used to locate a queue that contains response messages for two-way services. If specified, the service is a two-way service. This property is configured in the same way as the destination property.

Table 110. Supported properties for creating a service with the build toolkit. (continued)

Property name	Importance	Property description
expiry	Optional	<p>Specifies the expiry time of messages that are sent by the IBM MQ service provider. If set, the value is an integer that describes how long the message is available in milliseconds before it expires. By default messages do not expire.</p> <p>Setting <code>expiry</code> is equivalent to setting the MQMD Expiry field.</p> <p>Negative values mean that messages never expire. The default value is -1.</p> <p>REST clients can override <code>expiry</code> by specifying an <code>ibm-mq-md-expiry</code> HTTP header with a valid 64-bit integer.</p>
waitInterval	Optional	<p>Specifies how long, in milliseconds the IBM MQ service provider waits for messages to arrive on a queue.</p> <p>This property is only valid if <code>messagingAction=mqget</code> or <code>replyDestination</code> is set.</p> <p>If <code>replyDestination</code> is set, it must be a positive number.</p> <p>If <code>messagingAction=mqget</code>, <code>waitInterval</code> can be negative, which means that the IBM MQ service provider waits for ever until a message is available. If <code>waitInterval</code> is 0, the IBM MQ service provider does not wait.</p>

Table 110. Supported properties for creating a service with the build toolkit. (continued)

Property name	Importance	Property description
replySelection	Optional	<p>Defines how a two-way service locates reply messages on the queue that is referenced by the <code>replyDestination</code> property. If <code>replySelection</code> is used with a one-way service, it is ignored. The following values are valid:</p> <ul style="list-style-type: none"> msgIDToCorrelID Reply messages are assumed to be generated with the correlation ID set to the value of the message ID from the request message. The service generates a suitable message selector based on this information. This option is the default value. none No mechanism is used to correlate reply messages with request messages. The service gets the first available message on the reply queue. correlIDToCorrelID Reply messages are assumed to be generated with the correlation ID set to the value of the correlation ID from the request message. The service generates a suitable message selector based on this information. If the request message does not specify a correlation ID, the service generates a random correlation ID for the request message. For more information, see “ibm-mq-md-correlID” on page 272.
selector	Optional	<p>Defines a valid JMS message selector that is used to locate messages from the queue referenced by the destination attribute. Only valid with one-way services that have <code>messagingAction=mqget</code> defined. For more information, see Message selectors in JMS in the <i>IBM MQ</i> documentation.</p>

Table 110. Supported properties for creating a service with the build toolkit. (continued)

Property name	Importance	Property description
persistence	Optional	Describes the persistence of messages that are sent to the queue referenced by the destination property. The default value is <code>false</code> , indicating that messages are non-persistent. If set to <code>true</code> , messages are sent as persistent.
mqmdFormat	Optional	Completes the format field of the MQMD header in messages that are sent by the IBM MQ service provider. Only supported when the language property is specified. If not specified, then messages are sent with a blank format.
messagingAction	Optional	If the service is used to send messages, the value must be <code>mqput</code> . If the service is used to receive messages, the value must be <code>mqget</code> . If this property is set, the service is a one-way service.

Note: If you use the build toolkit on z/OS to generate a service archive, the values `requestStructure` and `responseStructure` can be a UNIX System Services file or a member of a partitioned data set. To specify a member of a partitioned data set, use the syntax `//'partitionDatasetName(member)'`. See the `languageStructureCodePage` property to indicate the code page of the z/OS partitioned data set member.

Create a two-way IBM MQ service with a properties file

Use a properties file to create a two-way IBM MQ stock query service that drives the stock query application.

Before you begin

Ensure that the following tasks are complete:

- Compile and test the sample IBM MQ stock query application, and create the necessary IBM MQ queue definitions. For more information, see “[Prepare the sample IBM MQ stock query application](#)” on page [63](#).
- Install z/OS Connect EE v3.0 or later. For more information, see “[Installing z/OS Connect EE](#)” on page [196](#).
- Create a z/OS Connect EE server and configure it to connect to IBM MQ. For more information, see “[Create a server to connect to IBM MQ](#)” on page [77](#).

Procedure

In this procedure, substitute the Windows example for the operating system of your choice. The build toolkit properties file must be encoded in UTF-8.

1. Create a local directory such as `C:/DevOpsScenarios` on the Windows system.
2. Download the build toolkit compressed file, `zconbt.zip` as a binary file, to the local directory created in step “1” on page [819](#).

The build toolkit is in the z/OS Connect EE <installation_path> and is called `zconbt.zip`.

3. Extract the `zconbt.zip` file into a `zconbt` sub directory of the `C:/DevOpsScenarios` local directory.
4. Add the `zconbt` executable file to your path.

For example, enter the following command:

```
set PATH=C:/DevOpsScenarios/zconbt/bin;%PATH%
```

5. Create a properties file, called `stockQuery.properties`, that defines the stock query service in `C:/DevOpsScenarios`.

The file contains the following properties.

```
provider=mq
name=stockQuery
version=1.0
description=A stock query service based on IBM MQ.
connectionFactory=jms/cf1
destination=jms/stockRequestQueue
replyDestination=jms/stockResponseQueue
waitInterval=10000
language=COBOL
requestStructure=C:/DevOpsScenarios/STOCKREQ.COPY
responseStructure=C:/DevOpsScenarios/STOCKRES.COPY
operationName=STOCKQRY
mqmdFormat=MQSTR
```

6. Create a file called `STOCKREQ.COPY` in `C:/DevOpsScenarios`.

This file contains the COBOL copybook that is used for the request payload of the stock query service and contains the following:

```
01 SQREQ.
 05 ITEM-ID  PIC 9(6).
```

7. Create a file called `STOCKRES.COPY` in `C:/DevOpsScenarios`.

This file contains the COBOL copybook that is used for the response payload of the stock query service and contains the following lines:

```
01 SQRESP.
 05 ITEM-ID    PIC 9(6).
 05 ITEM-DESC  PIC X(20).
 05 ITEM-COUNT PIC 9(6).
 05 ITEM-COST  PIC 9(6).
```

8. Run the build toolkit CLI to create the service archive file.

Enter the following command:

```
zconbt -p=C:/DevOpsScenarios/stockQuery.properties -f=C:/DevOpsScenarios/stockQuery.sar
```

The response from this command contains the following messages:

```
BAQB0000I: z/OS Connect Enterprise Edition Build Toolkit Version 1.1.
BAQB0001I: Creating service archive from configuration file C:/DevOpsScenarios/
stockQuery.properties.
BAQB1022I: IBM MQ for z/OS plugin for IBM z/OS Connect EE V3.0 build toolkit, code level is
3.0.21.0(20190417-1944).
BAQB0002I: Successfully created service archive file C:/DevOpsScenarios/stockQuery.sar.
```

Results

You created a service archive file that defines the stock query service.

What to do next

Deploy the stock query service to the z/OS Connect EE server

Deploy the stock query service manually

Deploy the stock query service to the z/OS Connect EE server by using the z/OS Connect EE REST API. For more information about the z/OS Connect EE REST API, see [“Deploy a service” on page 663](#)

Before you begin

Ensure that the following tasks are complete:

- [“Prepare the sample IBM MQ stock query application” on page 63.](#)
- [“Installing z/OS Connect EE” on page 196.](#)
- [“Create a server to connect to IBM MQ” on page 77.](#)
- [“Create a two-way IBM MQ service with a properties file” on page 819.](#)

Procedure

1. Enter the following command from the C:\DevOpsScenarios directory, replacing host and port with values that are appropriate for your z/OS Connect EE server.

```
curl -i -k -X POST http://host:port/zosConnect/services --data-binary "@/DevOpsScenarios/stockQuery.sar" -H "Content-Type:application/zip"
```

This command uses the following parameters:

- -i gives more output
- -k ignores any problems with self-signed TLS certificates.
- -X defines the HTTP verb to use. POST in this case as you are creating a service.
- --data-binary defines the path to stockQuery.sar.
- -H defines a header to send to the service. In this case, you are indicating that you are sending a .zip file.

2. Check the return code.

If the command was successful, the return code is HTTP 201. Information about the deployed service is also displayed.

Results

The stock query service is deployed to the z/OS Connect EE server.

What to do next

Test the stock query service.

Test the stock query service

Test the stock query service by using curl to send a HTTP request to invoke the service.

Before you begin

Ensure that the following tasks are complete:

- [“Prepare the sample IBM MQ stock query application” on page 63](#) and check it is running waiting to receive messages.
- [“Installing z/OS Connect EE” on page 196.](#)
- [“Create a server to connect to IBM MQ” on page 77.](#)
- [“Create a two-way IBM MQ service with a properties file” on page 819.](#)
- [“Deploy the stock query service manually” on page 821.](#)

Procedure

1. Enter the following command to query the state of the stock item with ID 2033.

Replace host and port with values that are appropriate for your z/OS Connect EE server.

```
curl -i -k -X POST http://host:port/zosConnect/services/stockQuery?action=invoke --data
"{"STOCKQRYOperation": { "sqreq": { "item_id": 2033 }}}" -H "Content-Type:application/
json"
```

2. Check the response.

An HTTP response code of 200 is expected with the following response:

```
{"STOCKQRYOperationResponse": {"sqresp": {"item_count": 500, "item_cost": 4, "item_desc": "A
description.", "item_id": 2033}}}
```

Results

You successfully used REST to communicate to the stock query application by using the stock query service that is running in z/OS Connect EE.

What to do next

Follow the steps in [“Create an API to invoke the IBM MQ stock query service” on page 149](#).

Building service archive using the build toolkit SDK

This sample demonstrates how to use the build toolkit SDK to build your service archive file.

Before you begin

- Ensure that the z/OS Connect EE build toolkit is installed. Follow the instructions in [“Installing the z/OS Connect EE build toolkit” on page 199](#).
- Ensure that all the .jar files from the lib and plugin directories are on the class path.

About this task

Follow these steps to use the z/OS Connect EE build toolkit SDK to create a service archive file. To learn more about the build toolkit, see [“The z/OS Connect EE build toolkit” on page 13](#).

Procedure

1. In your IDE, create a Map<String, String> object to contain the properties required to build the service archive file.
2. Create a SARGenerator object passing in the Map created in Step 1.
If any properties are invalid, an InvalidPropertyException occurs. The error message contains the property in error and the reason.
3. Save the service archive file.
Call the save() method and pass the filename. For example
4. You can now deploy the service archive file by copying it to the services directory.

Example

Example for the REST client:

```
Map<String, String> properties = new HashMap<String, String>();
properties.put("name", "example");
properties.put("provider", "rest");
properties.put("version", "1.0");
properties.put("description", "An example REST client service");
properties.put("requestSchemaFile", "request.json");
```

```

properties.put("responseSchemaFile", "response.json");
properties.put("verb", "POST");
properties.put("uri", "/resource/item");
properties.put("connectionRef", "conn");

SARGenerator sarGenerator = new SARGenerator(properties);
sarGenerator.save("/example/output/dir/file.sar");

```

Related tasks

[“Re-importing changed services” on page 600](#)

Related reference

[“zconbt command syntax” on page 795](#)

The **zconbt** command starts the build toolkit tool. You can use the build toolkit to generate archive files for services, APIs or API requesters.

Sample JCL

The following sample JCL is included with z/OS Connect EE and can be found in the *<hlq>.SBAQSAMP* partitioned dataset.

BAQJS2LS

The BAQJS2LS utility reads a JSON schema and generates the corresponding binding file and language structure file. For more information, see [“Creating bind and schema files” on page 297](#).

BAQLS2JS

The BAQLS2JS utility reads a COBOL copybook, PLI structure, or C structure file and generates a binding file and JSON schema files. For more information, see [“Creating bind and schema files” on page 297](#).

BAQSTRT

Started task used to run a z/OS Connect EE server. For more information, see [“Starting and stopping z/OS Connect EE” on page 471](#).

BAQZCBT

Sample JCL to run the build toolkit CLI on z/OS.

BAQZANGL

Creates the Angel process started task. You must customize the sample JCL, create SAF definitions to associate the started task with a user ID, and permit your z/OS Connect EE server to use the z/OS authorized services. For more information, see [“Configuring the Liberty Angel process and z/OS authorized services” on page 465](#).

BAQS123

Sample JCL to format SMF 123 subtype 1 version 1 records. For more information, see [“Using SMF records to monitor requests” on page 475](#).

BAQS123A

Uses the ICETOOL utility to format SMF 123 subtype 1 version 2 records, but only formats the first request data section in the record. For more information, see [“Using SMF records to monitor requests” on page 475](#).

BAQSMFP

Uses a sample C program to format all the request data sections in an entire SMF subtype 1 version 2 record.

HTTP response code reference

REST APIs use HTTP status codes to relay the results from an API operation. Status codes are primarily useful for detecting REST API errors, and by specifying your own response codes you can diagnose those errors more efficiently.

HTTP status codes are divided into classes according to the leading digit of the status code. The following table summarizes the HTTP status codes supported by the z/OS Connect EE API toolkit, and common practices in implementation.

Class	Description
2xx Success	The user request was successfully received, accepted, and processed.
3xx Redirection	Indicates that the user must take additional action to complete the request.
4xx Client Error	The user request was not fulfilled due to malformed request syntax or other client issues.
5xx Server Error	The user request appeared valid, but was not fulfilled due to an issue with the server.

Note: The z/OS Connect EE API toolkit supports user-created response codes. To create your own response code, specify a numeric value in the range of 100 to 599 within the Add Response window of the API editor.

Restriction: The z/OS Connect EE API toolkit does not support the specification of **1xx** class response codes.

The following list demonstrates how HTTP status codes are commonly used in REST APIs. Use this list to determine which status codes are suitable to implement into your API operation responses.

200 (OK)

The REST API accepted and fulfilled the user request.

201 (Created)

The REST API accepted the user request, and created a resource inside of a collection.

You can reference the newly created resource in the response message URI path using HTTP-to-JSON mapping.

202 (Accepted)

The REST API has accepted the user request. A 202 response is typically used for requests which either take a long time to process, or for requests that are not guaranteed to be fulfilled.

The current status of the user's request can be returned in the response message URI path using HTTP-to-JSON mapping.

204 (No Content)

The REST API has successfully completed the user request and is not returning a status message.

The API operation was successful, but no information is returned in the response body. A 204 response does not have a message body. If the client sending the HTTP request is a user agent, the 204 response should not change the user's document view.

300 (Multiple Choices)

The requested resource has multiple options, or there are multiple versions of the requested resource.

301 (Moved Permanently)

The REST API resource model has been changed, the requested resource has been assigned a new URI.

302 (Found)

The requested resource has been either moved temporarily, or may be requested by using a different HTTP request.

304 (Not Modified)

A 304 status code informs the user that the requested resource has not been modified since the date specified in the If-Modified-Since field.

400 (Bad Request)

The REST API either cannot process the user request due to a client error or malformed request syntax.

401 (Unauthorized)

The user request requires authentication. If authentication credentials were provided, access has not been granted to the requested resource.

403 (Forbidden)

The user request is valid, but the REST API refuses to process the request.

A 403 status code differs from a 401 status code, and should not be used interchangeably. A 403 status code typically indicates that permission is required for the requested resource, a 401 status code informs the user that valid authentication credentials are required for the requested resource.

404 (Not Found)

The requested resource cannot be located. The REST API either cannot map the user request URI to a resource, or the resource is no longer available.

410 (Gone)

The requested resource is no longer available. This status code should only be issued if it is certain that the requested resource has been permanently removed.

500 (Internal Server Error)

The REST API encountered an unexpected condition. This generic error response may be issued when a more specific status code is not suitable.

501 (Not Implemented)

The REST API does not support the functionality specified in the user request.

503 (Service Unavailable)

A 503 status code is issued when the REST API service is unavailable. Generally this indicates that the service could be down for maintenance, has crashed, or is temporarily unable to handle the user request.

550 (Permission Denied)

The REST API accepts the user request as valid, but did not process. The user does not have the necessary permissions to make the request.

Supplied server templates

z/OS Connect EE includes server templates that can be used to create new Liberty servers.

For more information about running the **zosconnect create** command see “[Creating a z/OS Connect Enterprise Edition server](#)” on page 217 and “[zosconnect command syntax](#)” on page 796.

zosconnect:default

This template provides a starting point for z/OS Connect EE servers.

To create a server with the supplied z/OS Connect EE template, use the following command. Enter this command from the <installation_path>/bin directory, for example /usr/lpp/IBM/zosconnect/v3r0/bin:

```
zosconnect create myServer --template=zosconnect:default
```

imsDefault

This template provides a starting point for z/OS Connect EE servers that connect to IMS.

To create a server with the supplied z/OS Connect EE template, use the following command. Enter this command from the <installation_path>/bin directory, for example /usr/lpp/IBM/zosconnect/v3r0/bin:

```
zosconnect create myServer --template=imsmobile:imsDefault
```

zosconnect:apiRequester

This template provides a starting point for z/OS Connect EE servers that use the API requester feature.

To create a server with the supplied IMS service provider template, use the following command. Enter this command from the <installation_path>/bin directory, for example /usr/lpp/IBM/zosconnect/v3r0/bin:

```
zosconnect create myServer --template=zosconnect:apiRequester
```

zosconnect:sampleCicsIpicCatalogManager

This template provides a server that is configured with an API and CICS service provider services for the CICS Catalog Manager sample application. The server is configured to communicate with CICS over an IPIC connection.

To create a server with the supplied CICS service provider template, use the following command. Enter this command from the <installation_path>/bin directory, for example /usr/lpp/IBM/zosconnect/v3r0/bin:

```
zosconnect create catalogManager --template=zosconnect:sampleCicsIpicCatalogManager
```

zosconnect:sampleImsPhonebook

This template provides a server that is configured with an API and services for the IMS phonebook sample application.

To create a server with the supplied IMS service provider template, use the following command. Enter this command from the <installation_path>/bin directory, for example /usr/lpp/IBM/zosconnect/v3r0/bin:

```
zosconnect create phonebook --template=zosconnect:sampleImsPhonebook
```

zosconnect:sampleImsDatabase

This template provides a server configured with an API and services for the IMS IVP sample application database.

To create a server with the supplied IMS database service provider template, use the following command. Enter this command from the <installation_path>/bin directory, for example /usr/lpp/IBM/zosconnect/v3r0/bin:

```
zosconnect create imsDatabase --template=zosconnect:sampleImsDatabase
```

zosconnect:mqDefault

This template provides a starting point for z/OS Connect EE servers that connect to IBM MQ.

To create a server with the supplied z/OS Connect EE template, use the following command. Enter this command from the <installation_path>/bin directory. For example, /usr/lpp/IBM/zosconnect/v3r0/bin.

```
zosconnect create myServer --template=zosconnect:mqDefault
```

zosconnect:sampleMqStockManager

This template provides a server that is configured with an API and IBM MQ service provider services for the IBM MQ stock manager sample application. The server is configured to communicate with IBM MQ through a bindings connection.

To create a server with the supplied z/OS Connect EE template, use the following command. Enter this command from the <installation_path>/bin directory, for example /usr/lpp/IBM/zosconnect/v3r0/bin:

```
zosconnect create myServer --template=zosconnect:sampleMqStockManager
```

zosconnect:sampleDb2ProjectManager

This template provides a server configured with an API and services for the DB2® project manager sample. This scenario uses the REST client service provider to communicate with Db2.

To create a server with the supplied Db2 template, use the following command. Enter this command from the <installation_path>/bin directory, for example /usr/lpp/IBM/zosconnect/v3r0/bin:

```
zosconnect create db2ProjectManager --template=zosconnect:sampleDb2ProjectManager
```

Legal Notices

This section provides the product legal notices.

This information was developed for products and services offered in the U.S.A. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property rights may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America*

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication.

IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact

IBM Director of Licensing

IBM Corporation

North Castle Drive, MD-NC119 Armonk,

NY 10504-1785

United States of America

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at [Copyright and trademark information at www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM online privacy statement

IBM Software products, including software as a service solutions, (*Software Offerings*) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you, as customer, the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see and , the section entitled *Cookies, Web Beacons and Other Technologies* and the .

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property rights may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America*

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact

*IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US*

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM online privacy statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at www.ibm.com/privacy and IBM's Online Privacy Statement at www.ibm.com/privacy/details the section entitled *Cookies, Web Beacons and Other Technologies* and the *IBM Software Products and Software-as-a-Service Privacy Statement* at www.ibm.com/software/info/product-privacy.

Glossary

The z/OS Connect Enterprise Edition glossary of terms and definitions.

This section contains terms and definitions that are used in z/OS Connect EE and some general computing terms.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

#

<installation_path>

A variable that is used throughout this documentation in file paths to represent the directory where you installed z/OS Connect EE. The default value is /usr/lpp/IBM/zosconnect/v3r0.

<server.config.dir>

A Liberty property that identifies the server configuration directory, and refers to the \${wlp.user.dir}/servers/<serverName> directory.

<server.output.dir>

A Liberty property that identifies the location for server-generated output such as the logs directory, the work area directory, and generated files. It has the same value as <server.config.dir>.

A

Angel process

A started task that you configure to allow z/OS Connect EE to use z/OS authorized services such as System Authorization Facility (SAF). z/OS Connect EE provides sample JCL BAQZANGL, which invokes the IBM WebSphere Liberty program bbgzangl.

application programming interface (API)

In z/OS Connect EE, an API that follows REST principles, which you can create by using the API toolkit.

API archive (.aar file)

An archive file of type .aar that contains an API definition and information about how the API maps to one or more services. You deploy an API archive to z/OS Connect EE to make an API available.

API consumer

An application that calls a RESTful API to access valuable content, data, or resources.

API information file

A file that contains a data structure and parameters for calling an API from a z/OS application. The data structure is generated by using the z/OS Connect EE build toolkit.

API project

An Eclipse project type that you can use to define an API and how the API maps to one or more services. An API project is exported as an API archive.

API provider

An application that is exposed by a RESTful API.

API requester

An interface that manages the API calls made by a z/OS application and the responses it receives.

API requester archive (.ara file)

An archive file of type .ara that z/OS Connect EE uses to process API requests made by z/OS applications.

API requesters directory

A directory that you can use to deploy API requester archives automatically on the server. The default path is \${server.config.dir}/resources/zosconnect/apiRequesters. However, the server does not automatically create this directory. Either create this directory or specify an existing directory before you deploy an API requester archive.

APIs directory

A directory that you can use to deploy API archives automatically on the server. Polling of this directory is disabled by default.

Application Programming Interface key (API key)

An API key is a token provided by a client to identify the calling application or user when making an API call. API keys should be a secret that only the client and server know. API key-based authentication is only considered secure if used together with other security mechanisms such as TLS.

Application Transparent Transport Layer Security (AT-TLS)

Is based on z/OS System SSL. It creates a secure session on behalf of an application. Instead of implementing TLS in every application that requires a secure connection, AT-TLS provides encryption and decryption of data based on policy statements that are coded in a Policy Agent. The application sends and receives clear text (unencrypted data) while AT-TLS encrypts and decrypts data at the TCP transport layer.

C

communication stub

A load module that handles the HTTP connection and communication from a z/OS subsystem to a z/OS Connect EE server when you want to make API calls from a z/OS application.

D

data transformer

An OSGI service that converts between JSON documents and a format that is suitable for a target z/OS application.

I

IBM WebSphere Trust Association Interceptor (TAI).

The Trust Association Interceptor (TAI) is a WebSphere Liberty Profile interface. It can be used by a z/OS Connect EE server to perform custom authentication, by enabling an external component to authenticate the user's credentials and then assert the identity to the z/OS Connect EE server.

IMS mobile feature and IMS mobile solution

These terms were previously used to describe the IMS service provider, and are synonymous.

installation_path

See “[<installation_path>](#)” on page 835

interceptor

An OSGI service that performs an action before and after an HTTP request. Interceptors can read the data in a request and reject requests if necessary.

J

JavaScript Object Notation (JSON)

A lightweight data-interchange format that is based on the object-literal notation of JavaScript. JSON is programming-language neutral but uses conventions from languages that include C, C++, C#, Java, JavaScript, Perl, Python.

Java Secure Socket Extension (JSSE)

Provides a framework and an implementation for a Java version of the SSL, TLS, and DTLS protocols and includes functionality for data encryption, server authentication, message integrity, and optional client authentication.

JSON schema

A JSON document that describes the format of other JSON documents.

JSON web service

A web service that accepts and produces JSON payloads.

L

Lightweight Third-Party Authentication (LTPA)

Authentication token used in IBM WebSphere products. LTPA tokens can be used to encrypt, digitally sign, and securely transmit authentication-related data, and later decrypt and verify the signature. The protocol also provides a Single Sign-On (SSO) feature, whereby a user is required to authenticate only when connecting to a domain name system (DNS). Then, the user can access resources in other Liberty servers in the same domain without getting prompted.

O

OAuth 2.0

OAuth 2.0 is an open standard that allows an end user to securely share access to information with a third party. It simplifies the user experience by avoiding the need of managing many different credentials for multiple web sites. The goal of OAuth 2.0 is to enable a web site to request user information from another site, whilst allowing the end user to control access to the information.

One-way service (MQ)

A one-way service can be used to provide a RESTful API on top of a single IBM MQ queue or topic.

P

policy

A set of rules used to adjust how an API request is processed, based on the HTTP header values set by the client.

port sharing

A way of load balancing TCP/IP connections across a group of servers that run in the same z/OS image.

R

request endpoint

Defines the endpoint of an API provider for API requests made by z/OS applications.

RESTful

Relating to applications and services that conform to Representational State Transfer (REST) constraints. The z/OS Connect EE API toolkit is designed to help you create RESTful APIs.

response code

The HTTP status code that is returned in the response of a z/OS Connect EE RESTful API operation.

S

SAF cache

Contains SAF user IDs and any associated RACF groups in which the user ID resides. The SAF cache is only applicable to API requester, and only when ID assertion is enabled.

server.config.dir

See “[`<server.config.dir>`](#)” on page 835

server.output.dir

See “[`<server.output.dir>`](#)” on page 835

server.xml

The primary configuration file for a z/OS Connect EE server. z/OS Connect EE is a Liberty-based product, and therefore contains many of the same configuration elements.

service

A resource that processes HTTP requests, which contain JSON payloads, to call z/OS assets. JSON payloads might be converted by the service, or a separate data transformer, into a format suitable for the z/OS assets.

service archive (.sar file)

An archive file of type .sar that contains the information that is needed by a z/OS Connect EE service provider to make a z/OS asset available as a service. You deploy a service archive to z/OS Connect EE to make a service available.

service interface (.si file)

A file of type .si that defines the fields to expose for request and response messages to and from a target z/OS asset, and how these fields are exposed when you map an API to one or more services in an API project. Service interfaces are used in service projects that target CICS, IMS, and IBM MQ programs.

service project

An Eclipse project type that you can use to define a service. A service project is exported as a service archive.

service provider

A resource that manages the connection between z/OS Connect EE and a target z/OS subsystem.

services directory

A directory that you can use to deploy service archives automatically on the server. Polling of this directory is disabled by default.

Swagger

The swagger specification defines the format of RESTful APIs. The specification enables development and consumption of an API by mapping the resources and operations associated with the API. For more information, see [Swagger specification](#).

T**TAI**

See “[IBM WebSphere Trust Association Interceptor \(TAI\).](#)” on page 836

TCP/IP load balancing

The ability to distribute TCP/IP connections across target servers.

Two-way service (MQ)

A two-way service allows a RESTful client to perform request-reply messaging on a pair of IBM MQ queues.

W**WebSphere Optimized Local Adapters (WOLA)**

An efficient, cross-memory connectivity technology that is used by the WOLA service provider.

WLP_USER_DIR

An environment variable that you can use to specify an alternative location for the user directory \${wlp.user.dir}. The default value is /var/zosconnect.

Web service binding (WSBind file)

A resource that describes the specifics of a web service. WSBind files are required when you use the WOLA service provider or configure a custom data transformer. You can generate WSBind files by using the BAQLS2JS and BAQJS2LS utilities.

Z**z/OS application**

An application that runs on z/OS. These can include CICS programs, IMS transactions, Db2 stored procedures, WOLA-enabled applications, batch programs, and long-running z/OS applications.

z/OS asset

A resource that you can expose as an API by using z/OS Connect EE. For example, a supported z/OS application, an IBM Db2 table, or an IBM MQ queue.

z/OS Connect EE API toolkit

An Eclipse-based workstation tool that you install into IBM Explorer for z/OS to design and create services and RESTful APIs for accessing z/OS resources.

z/OS Connect EE build toolkit

A command line tool and SDK that you can use to generate service archives and API requester archives.

z/OS subsystem

A system that runs on z/OS. For example, IBM CICS, IBM IMS, IBM MQ, or IBM Db2.

Index

A

Abend codes [725](#)
ACL
 IMS database service provider [258](#)
 IMS service provider [238](#)
administering [673](#)
administration [328](#)
and
 JSON Web Token
 JWT [439](#)
 OAuth [439](#)
Angel process [250](#)
API [298](#), [299](#), [673](#), [676–678](#), [680](#), [682](#), [683](#)
API archive (aar) [570](#), [601](#)
API archive file
 creating [589](#)
 Customizing [591](#)
 definition [570](#)
 deploying [595](#), [674](#)
 editing [599](#)
 exporting [600](#)
 undeploying [674](#)
API call [176](#), [184](#)
API consumer
 workflow [9](#)
API editor
 preferences [606](#)
 usage tips [586](#)
API key [420](#), [637](#)
API management [329](#)
API package
 editor [579](#)
 examining [596](#)
 overview [579](#)
 removing [596](#)
 starting [596](#)
 stopping [596](#)
 testing [596](#)
API requester
 communication stub [397](#)
 configuring [304](#)
 IMS [309](#)
 IMS communication stub [309](#)
 IMS configuration [309](#)
 workflow [9](#)
z/OS applications
 COBOL [653](#)
 testing [653](#)
API requester (Db2)
 communication stub [312](#)
 configuration [312](#)
API requester (z/OS applications)
 communication stub [312](#)
 configuration [312](#)
API requester archive
 generating [189](#), [618](#), [620](#)

API requester archive (*continued*)

 generation attributes [621](#)
API requester scenario [178](#)
API requesters
 HA environment [314](#)
API toolkit
 capabilities [751](#)
 modeling [590](#)
 service import [600](#)
apideploy [792](#)
ara [617](#), [621](#)
array
 counters [497](#)
artifacts for an API requester [617](#)
asserted ID [431](#)
authentication
 IMS database service provider [258](#)
 IMS service provider [238](#)
authorization
 IMS database service provider [258](#)
 IMS service provider [238](#)

B

BAQR7085E [751](#)
BAQR7086E [751](#)
basic authentication [438](#)
bind [297](#)
bind files [297](#)
Boolean
 type
 service [497](#)
build toolkit [13](#), [199](#), [560](#), [602](#), [822](#)
build toolkit CLI [189](#), [618](#)
build toolkit SDK [620](#)

C

Call an API with a
 local
 JWT [431](#)
call REST APIs [9](#)
capabilities [751](#)
catalog manager [59](#)
CCSID [515](#)
change the of an
 API requester [696](#)
 status [696](#)
CICS [176](#)
client [606](#)
client certificates [604](#)
Coded character set identifiers [515](#)
Communication stub
 Db2 applications) [312](#)
 IMS [309](#)
 z/OS applications [397](#)
 z/OS applications) [312](#)

concepts 6
configuration 171, 326, 327
configure
 id assertion 444
 identity assertion 444
configure a locally generated JWT 434
configure CICS sample 59
Configure z/OS Connect EE for
 OAuth 2.0 425
configuring
 IMS Mobile feature 243
 IMS service provider 243
Configuring
 Db2(Communication stub) 312
 services 219
connection profiles 252
connection to the request point 628
contacting IBM Software Support 717
Conversion 606
CORS
 CORS 597
CORS preflight check 300
counters
 array
 service 497
create 217
create api 118, 121, 124, 127
create server 217
creating service 701
Creating service provider 701
custom
 CCSID 527
 code page 527

D

data conversion
 escaping special characters 537, 540
 input field initialization 537
 suppressing empty tags 537
 suppressing low values 537
data transformation 606
data transformer 701, 713
database
 Service Provider 263
database service provider 263
DB
 SP 263
DELETE 669, 683, 699
delete an
 API requester 699
delete server 219
deploy an
 API requester 693, 697
deploying 673
deploying API 298, 299
deploying API requester 304
developing 183, 192
development 606
devOps 14
distributed identity 253
dynamic
 routing 643
 URIMAP 643

E

edit
 field
 wizard 497
Edit
 Field
 wizard 497
endpointConnection 179, 187

F

facilitating RESTful API calls 617
for
 API key 420, 637
 API requesters 637
 OAuth 637, 639

G

generating API archives 602
generating service archives 560
GET 656, 657, 659–661, 668, 676–678, 694, 695
get a list of
 API requesters 694
get details of an
 API requester 695
glossary of terms and abbreviations 835
GMOBA messages 741
GMOCR messages 743
GMOIG messages 744
GMOMW messages 738
GMOPR messages 747
GMOSR messages 748
GMOTM messages 749

H

HA
 connections to IMS 321
high availability 326
high availability
 connections to IMS 321
HTTP mapping 586, 591
HTTP response 593
HTTP Status
 Customizing 593

I

importing
 COBOL 489
 PL/I 490
IMS
 database service
 connecting 265
 service provider
 pass Ticket support 241
IMS connection profiles 247
IMS database
 host connection 265
 server connection 265
IMS database service provider

IMS database service provider (*continued*)

 pass Ticket support 260
 user type converter
 support 262

IMS DB

 overview 257
 service provider
 pass Ticket support 260
 user type converters 262

IMS DB service provider

 overview 257

IMS DB SP 263

IMS interaction profiles 248

IMS Mobile feature

 configuration 243
 connection profiles 246
 IMS communication verification 245
 interaction profiles 246
 interaction properties 246
 overview 238
 service definition 246

IMS service cache 238

IMS service provider

 configuration 243
 connection profiles 246
 IMS communication verification 245
 IMS connection profiles 247
 IMS interaction properties profiles 248
 interaction profiles 246
 interaction properties 246
 overview 238
 pass Ticket support 241
 restrictions 238, 257
 service definition 246

IMS technical ID 250

IMS technical password 250

installation

 API toolkit 201

installing 195, 196

interceptor 290, 701

IP interconnectivity 221

J

JSON schema 627

JSON schema mapping

 COBOL 510, 531
 escaping special characters 537
 input field initialization 537
 PL/I 535
 service-level 537, 540
 special characters 540
 suppressing empty tags 537
 suppressing low values 537

JSON Web Token 426, 439

JWT

 parameter 642
 variable 642

jwtBuilder 434

L

log records

log records (*continued*)

 IMS database service provider 258
 IMS service provider 238
logger 290

M

manage

- API requesters 693

management 329

mapping editor 586

mapping rules 581, 583

messages

- IMS service provider 741

Messages 725

Migrating 202

multiple and methods

- authentication 438
- authorization 438

Multiple response codes 593

N

Null type 588

O

OAuth

- parameter 639
- variable 639

OAuth 2.0 438

operating 471

operation 328

P

PassTicket support for

- IMS database service provider 260
- IMS service provider 241

port sharing 170, 318

POST 563, 666, 697

preferences 606

Prerequisites 178

Problem analysis 715

problem determination

- trace 715

Product extensions 197

protocols 221

PUT 667, 680, 682, 696, 698

R

RACF

- IMS database service provider 258

RACF groupname 250

RACF setting

- IMS service provider 238

REDEFINES 510

rename server 219

request endpoint 179, 187

request mapping 581, 583

Resolving problems 715

Response codes 593

response mapping 581, 583
 Response mapping 593
 REST APIs
 building 487, 564
 design 487, 564
 design workflow 8
 requirements 8
 RESTful administration interface
 API requesters 694
 APIs 675
 services 655
 RESTful services
 design 570
 overview 570
 Restrictions for
 IMS database service provider 257

S

SAF 250, 254
 scenario 59, 118, 121, 124, 127, 172–175, 183, 192, 193, 233, 236, 465
 Scenarios 55, 176
 schema 297
 schema files 297
 second server 173
 security
 CICS connection security 225
 configuration 250, 254
 IMS database process flow 258
 IMS DB process flow 258
 IMS process flow 238
 IPIC connection security 225
 REST client connection security 282, 287
 technical password 254
 server
 creation 217
 deletion 219
 rename 219
 server connection 603, 604
 service
 array
 counters 497
 array counters 497
 connections 510
 deploying project 529, 530
 exporting project 529, 530
 field
 edit 497
 editor 497
 interface 493
 properties 510
 type
 Boolean 497
 Service
 configuration 219
 service archive
 CICS service provider 798
 Service archive
 REST client 555
 service archive (sar) 570, 601
 service deployment 563
 service details 657
 service interfaces

service interfaces (*continued*)
 COBOL importer errors 489
 defining 493
 PL/I importer errors 490
 service list 656
 service registry
 IMS Mobile feature 238
 service request schema 660
 service response schema 661
 service status 659, 666, 667
 service updates 668
 services
 creating 487
 Services
 removing 529, 554
 starting 529, 554
 stopping 529, 554
 Setup 197
 shared configuration 171
 shared resources 172
 SMP/E 196
 SMPE 196
 starting 471
 status 682
 stopping 471
 string 636
 supported hardware 195
 supported software 195
 Swagger 601
 sysplex distributor 317

T

TCP/IP 170, 318
 tcip 170
 TCPIP 318
 technical ID 254
 technical password 254
 test 193
 transaction code override 256
 transformation 606
 transformer 713
 troubleshooting
 IMS service provider 741
 Troubleshooting 715
 type
 Boolean 497

U

undeploy 674
 undeploy an
 API requester 693
 update an
 API requester 693, 698
 updating 199, 202
 updating API editor 202
 upgrading 199
 Upgrading 203
 user authentication 252
 User type converter support for
 IMS database service provider 262

V

version 1 [202](#)
version 2 [203](#)

W

WebSphere optimized local adapters [233](#), [236](#)
wizard
 edit field [497](#)
[wlm](#) [317](#), [320](#)
[WOLA](#) [233](#), [236](#)
[workload](#) [317](#)
wrap an ID in the JWT [431](#)

Z

[z/OS application calls to APIs](#)
 [secure](#) [637](#)
[z/OS Connect EE](#)
 [capabilities](#) [751](#)
[zconbt](#) [795](#)
[zconsetup](#) [797](#)
[zosconnect_authTokenLocal](#) [434](#)

IBM.[®]