Zowe Documentation

Version 1.8.0

Contents

Chapter 1: Getting Started.....7
Zowe overview..... 8
Zowe overview..... 8
Zowe architecture.....13
Release notes.....14
Version 1.8.0 (January 2020)..... 14
Version 1.7.1 (December 2019)..... 16
Version 1.7.0 (November 2019).....17
Version 1.6.0 (October 2019).....18
Version 1.5.0 (September 2019)..... 19
Zowe SMP/E Alpha (August 2019).....20
Version 1.4.0 (August 2019).....20
Version 1.3.0 (June 2019)..... 22
Version 1.2.0 (May 2019)..... 24
Version 1.1.0 (April 2019)..... 25
Version 1.0.1 (March 2019)..... 26
Version 1.0.0 (February 2019)..... 27
Zowe CLI quick start..... 29
Installing.....29
Issuing your first commands..... 30
Using profiles.....30
Writing scripts.....30
Next Steps..... 31
Frequently Asked Questions.....31
Zowe FAQ..... 31
Zowe CLI FAQ.....33

Chapter 2: User Guide.....35
Planning and preparing the installation.....36
Introduction..... 36
System requirements.....36
Installing Node.js on z/OS..... 37
Configuring z/OSMF..... 39
Configuring z/OSMF Lite (for non-production use).....42
Installing Zowe z/OS components..... 59
Installation roadmap.....59
Installing Zowe runtime from a convenience build..... 62
Installing Zowe SMP/E Alpha..... 65
Configuring the z/OS system for Zowe..... 83
Configuring Zowe certificates..... 91
Creating and configuring the Zowe instance directory.....94
Installing and configuring the Zowe cross memory server (ZWESISTC).....97
Installing the Zowe started task (ZWESVSTC).....100
Verifying Zowe installation on z/OS..... 102
Zowe Auxiliary Address space..... 103
Uninstalling Zowe from z/OS..... 103
Installing Zowe CLI..... 104
Installing Zowe CLI..... 104

Updating Zowe CLI....106
Uninstalling Zowe CLI....107
Advanced Zowe configuration....108
Zowe Application Framework configuration....108
Configuring Zowe CLI....122
Using Zowe....127
Getting started tutorial....127
Using the Zowe Desktop....146
Using the Editor....150
API Catalog....151
Zowe CLI extensions and plug-ins....154
Extending Zowe CLI....154
Software requirements for Zowe CLI plug-ins....154
Installing Zowe CLI plug-ins....154
IBM® CICS® Plug-in for Zowe CLI....158
IBM® Db2® Database Plug-in for Zowe CLI....158
Zowe Explorer Extension for VSCode....161

Chapter 3: Extending....163
Developing for API Mediation Layer....164
Onboarding Overview....164
Onboarding a REST API service with the Plain Java Enabler (PJE)....167
Onboarding a service with the Zowe API Meditation Layer without an onboarding enabler....182
Onboard a Java REST APIs with Spring Boot....188
Onboard a Java REST API service without Spring Boot....200
Prerequisites....201
Onboard a REST API without code changes required....210
API Mediation Layer Message Service Component....217
API Mediation Layer onboarding configuration....219
Developing for Zowe CLI....224
Developing for Zowe CLI....224
Setting up your development environment....225
Installing the sample plug-in....226
Extending a plug-in....228
Developing a new plug-in....231
Implementing profiles in a plug-in....236
Developing for Zowe Application Framework....237
Overview....237
Plug-ins definition and structure....239
Building plugin apps....242
Installing Plugins....243
Embedding plugins....244
Dataservices....246
Authentication API....252
Internationalizing applications....254
Zowe Desktop and window management....258
Configuration Dataservice....261
URI Broker....267
Application-to-application communication....268
Configuring IFrame communication....272
Error reporting UI....273
Logging utility....275
Zowe Conformance Program....279
Introduction....279
How to participate....279

Chapter 4: Troubleshooting..... 281

Troubleshooting....282
Zowe API Mediation Layer.... 282
Troubleshooting API ML.... 282
Error Message Codes....288
Zowe Application Framework....301
Troubleshooting Zowe Application Framework.... 301
Gathering information to troubleshoot Zowe Application Framework....301
Known Zowe Application Framework issues.... 303
Raising a Zowe Application Framework issue on GitHub.... 306
Troubleshooting z/OS Services.... 307
z/OS Services are unavailable.... 307
Zowe CLI....308
Troubleshooting Zowe CLI.... 308
Gathering information to troubleshoot Zowe CLI.... 308
z/OSMF troubleshooting....311
Known Zowe CLI issues....311
Raising a CLI issue on GitHub....313
Troubleshooting Zowe through Zowe Open Community....313
Contact Zowe Open Community to Troubleshoot Zowe....314

Chapter 1

Getting Started

Topics:

- Zowe overview
- Release notes
- Zowe CLI quick start
- Frequently Asked Questions

Zowe overview

Zowe overview

Zowe™ is an open source project created to host technologies that benefit the IBM Z platform for all members of the Z community, including Integrated Software Vendors (ISVs), System Integrators, and z/OS consumers. Zowe, like Mac or Windows, comes with a set of APIs and OS capabilities that applications build on and also includes some applications out of the box. Zowe offers modern interfaces to interact with z/OS and allows you to work with z/OS in a way that is similar to what you experience on cloud platforms today. You can use these interfaces as delivered or through plug-ins and extensions that are created by clients or third-party vendors.

Zowe Demo Video

Watch this video to see a quick demo of Zowe.

Component Overview

Zowe consists of the following components:

Zowe Application Framework

A web user interface (UI) that provides a virtual desktop containing a number of apps allowing access to z/OS function. Base Zowe includes apps for traditional access such as a 3270 terminal and a VT Terminal, as well as an editor and explorers for working with JES, MVS Data Sets and Unix System Services.

Learn more

The Zowe Application Framework modernizes and simplifies working on the mainframe. With the Zowe Application Framework, you can create applications to suit your specific needs. The Zowe Application Framework contains a web UI that has the following features:

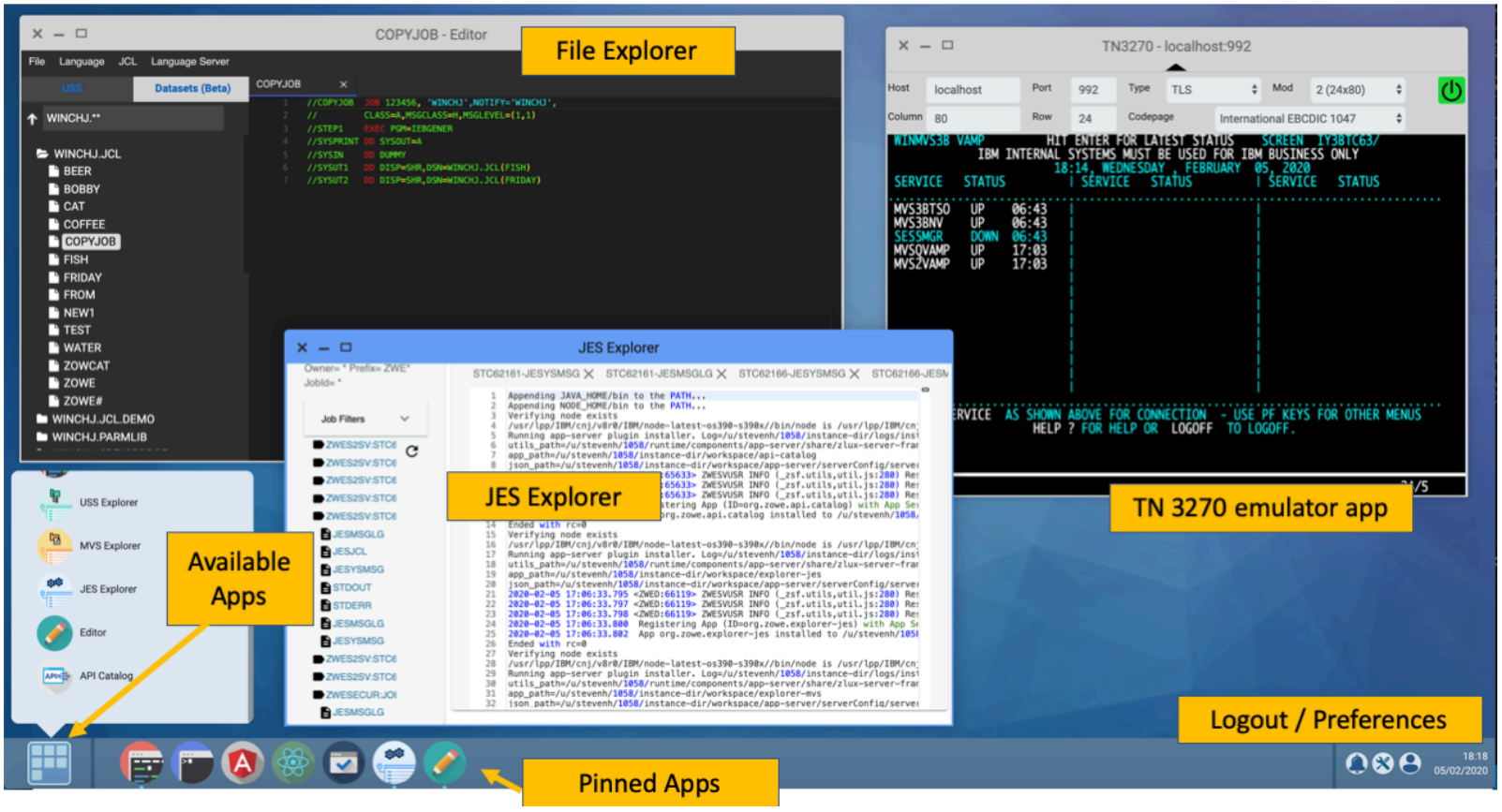
- The web UI works with the underlying REST APIs for data, jobs, and subsystem, but presents the information in a full screen mode as compared to the command line interface.
- The web UI makes use of leading-edge web presentation technology and is also extensible through web UI plug-ins to capture and present a wide variety of information.
- The web UI facilitates common z/OS developer or system programmer tasks by providing an editor for common text-based files like REXX or JCL along with general purpose data set actions for both Unix System Services (USS) and Partitioned Data Sets (PDS) plus Job Entry System (JES) logs.

The Zowe Application Framework consists of the following components:

- **Zowe Desktop**

  The desktop, accessed through a browser. The desktop contains a number of applications, including a TN3270 emulator for traditional Telnet or TLS terminal access to z/OS, a VT Termnial for SSH commands, as well as rich web GUI applications including a JES Explorer for working with jobs and spool output, a File Editor for working with USS directories and files and MVS data sets and members. The Zowe desktop is extensible and allows vendors to provide their own appications to run within the desktop. See Overview on page 237. The

following screen capture of a Zowe desktop shows some of its composition as well as the TN3270 app, the JES Explorer, and the File Editor open and in use.



- **Zowe Application Server**

  The Zowe Application Server runs the Zowe Application Framework. It consists of the Node.js server plus the Express.js as a webservices framework, and the proxy applications that communicate with the z/OS services and components.
- **ZSS Server**

  The ZSS Server provides secure REST services to support the Zowe Application Server. For services that need to run as APF authorized code, Zowe uses an angel process that the ZSS Server calls using cross memory

communication. During installation and configuration of Zowe, you will see the steps needed to configure and launch the cross memory server.

- **Application plug-ins**

  Several application-type plug-ins are provided. For more information, see Zowe Desktop application plug-ins on page 146.

z/OS Services

Provides a range of APIs for the management of z/OS JES jobs and MVS data set services.

Learn more

Zowe provides a z/OS® RESTful web service and deployment architecture for z/OS microservices. Zowe contains the following core z/OS services:

- **z/OS Datasets services**

  Get a list of data sets, retrieve content from a member, create a data set, and more.
- **z/OS Jobs services**

  Get a list of jobs, get content from a job file output, submit a job from a data set, and more.

You can view the full list of capabilities of the RESTful APIs from the API catalog that displays the Open API Specification for their capabilities.

- These APIs are described by the Open API Specification allowing them to be incorporated to any standard-based REST API developer tool or API management process.
- These APIs can be exploited by off-platform applications with proper security controls.

As a deployment architecture, the z/OS Services are running as microservices with a Springboot embedded Tomcat stack.

Zowe CLI

Zowe CLI is a command-line interface that lets application developers interact with the mainframe in a familiar, off-platform format. Zowe CLI helps to increase overall productivity, reduce the learning curve for developing mainframe applications, and exploit the ease-of-use of off-platform tools. Zowe CLI lets application developers use common tools such as Integrated Development Environments (IDEs), shell commands, bash scripts, and build tools for mainframe development. It provides a set of utilities and services for application developers that want to become efficient in supporting and building z/OS applications quickly.

Learn more

Zowe CLI provides the following benefits:

- Enables and encourages developers with limited z/OS expertise to build, modify, and debug z/OS applications.
- Fosters the development of new and innovative tools from a computer that can interact with z/OS. Some Zowe extensions are powered by Zowe CLI, for example the Zowe Explorer Extension for VSCode on page 161.
- Ensure that business critical applications running on z/OS can be maintained and supported by existing and generally available software development resources.
- Provides a more streamlined way to build software that integrates with z/OS.

Note: For information about software requirements, installing, and upgrading Zowe CLI, see Introduction on page 36.

Zowe CLI capabilities

With Zowe CLI, you can interact with z/OS remotely in the following ways:

- **Interact with mainframe files:** Create, edit, download, and upload mainframe files (data sets) directly from Zowe CLI.
- **Submit jobs:** Submit JCL from data sets or local storage, monitor the status, and view and download the output automatically.

- **Issue TSO and z/OS console commands:** Issue TSO and console commands to the mainframe directly from Zowe CLI.
- **Integrate z/OS actions into scripts:** Build local scripts that accomplish both mainframe and local tasks.
- **Produce responses as JSON documents:** Return data in JSON format on request for consumption in other programming languages.

For detailed information about the available functionality in Zowe CLI, see Zowe CLI Command Groups.

For information about extending the functionality of Zowe CLI by installing plug-ins, see Extending Zowe CLI on page 154.

More Information:

- System requirements on page 36
- Installing Zowe CLI on page 104

API Mediation Layer

Provides a gateway that acts as a reverse proxy for z/OS services, together with a catalog of REST APIs and a dynamic discovery capability. Base Zowe provides core services for working with MVS Data Sets, JES, as well as working with z/OSMF REST APIs. The API Mediation Layer also provides a framework for Single Sign On (SSO).

Learn more

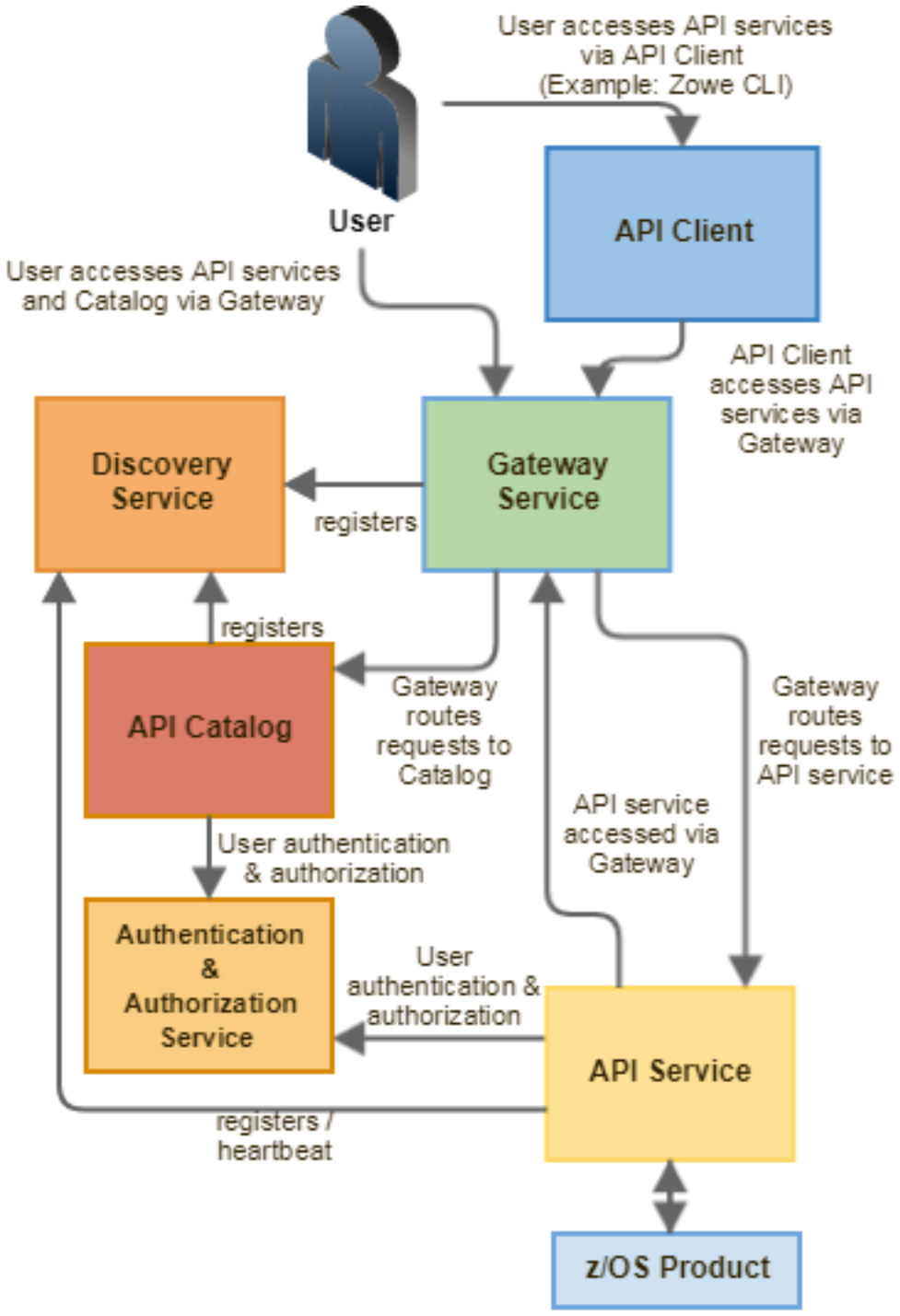
The API Mediation Layer provides a single point of access for mainframe service REST APIs. The layer offers enterprise, cloud-like features such as high-availability, scalability, dynamic API discovery, consistent security, a single sign-on experience, and documentation. The API Mediation Layer facilitates secure communication across loosely coupled microservices through the API Gateway. The API Mediation Layer consists of three components: the Gateway, the Discovery Service, and the Catalog. The Gateway provides secure communication across loosely coupled API services. The Discovery Service enables you to determine the location and status of service instances running inside the API ML ecosystem. The Catalog provides an easy-to-use interface to view all discovered services, their associated APIs, and Swagger documentation in a user-friendly manner.

Key features

- Consistent Access: API routing and standardization of API service URLs through the Gateway component provides users with a consistent way to access mainframe APIs at a predefined address.
- Dynamic Discovery: The Discovery Service automatically determines the location and status of API services.
- High-Availability: API Mediation Layer is designed with high-availability of services and scalability in mind.
- Redundancy and Scalability: API service throughput is easily increased by starting multiple API service instances without the need to change configuration.
- Presentation of Services: The API Catalog component provides easy access to discovered API services and their associated documentation in a user-friendly manner. Access to the contents of the API Catalog is controlled through a z/OS security facility.
- Encrypted Communication: API ML facilitates secure and trusted communication across both internal components and discovered API services.

API Mediation Layer architecture

The following diagram illustrates the single point of access through the Gateway, and the interactions between API ML components and services:



Components

The API Layer consists of the following key components:

API Gateway

Services that comprise the API ML service ecosystem are located behind a gateway (reverse proxy). All end users and API client applications interact through the Gateway. Each service is assigned a unique service ID that is used in the access URL. Based on the service ID, the Gateway forwards incoming API requests to the appropriate service. Multiple Gateway instances can be started to achieve high-availability. The Gateway access URL remains unchanged. The Gateway is built using Netflix Zuul and Spring Boot technologies.

Discovery Service

The Discovery Service is the central repository of active services in the API ML ecosystem. The Discovery Service continuously collects and aggregates service information and serves as a repository of active services. When a service is started, it sends its metadata, such as the original URL, assigned serviceId, and status information to the Discovery Service. Back-end microservices register with this service either directly or by using a Eureka client. Multiple enablers are available to help with service on-boarding of various application architectures including plain Java applications and Java applications that use the Spring Boot framework. The Discovery Service is built on Eureka and Spring Boot technology.

Discovery Service TLS/SSL

HTTPS protocol can be enabled during API ML configuration and is highly recommended. Beyond encrypting communication, the HTTPS configuration for the Discovery Service enables hightened security for service registration. Without HTTPS, services provide a username and password to register in the API ML ecosystem. When using HTTPS, only trusted services that provide HTTPS certificates signed by a trusted certificate authority can be registered.

API Catalog

The API Catalog is the catalog of published API services and their associated documentation. The Catalog provides both the REST APIs and a web user interface (UI) to access them. The web UI follows the industry standard Swagger UI component to visualize API documentation in OpenAPI JSON format for each service. A service can be implemented by one or more service instances, which provide exactly the same service for high-availability or scalability.

Catalog Security

Access to the API Catalog can be protected with an Enterprise z/OS Security Manager such as IBM RACF, CA ACF2, or CA Top Secret. Only users who provide proper mainframe credentials can access the Catalog. Client authentication is implemented through the z/OSMF API.

Onboarding APIs

Essential to the API Mediation Layer ecosystem is the API services that expose their useful APIs. Use the following topics to discover more about adding new APIs to the API Mediation Layer and using the API Catalog:

- Onboarding Overview on page 164
- Onboard a Java REST APIs with Spring Boot on page 188
- API Catalog on page 151

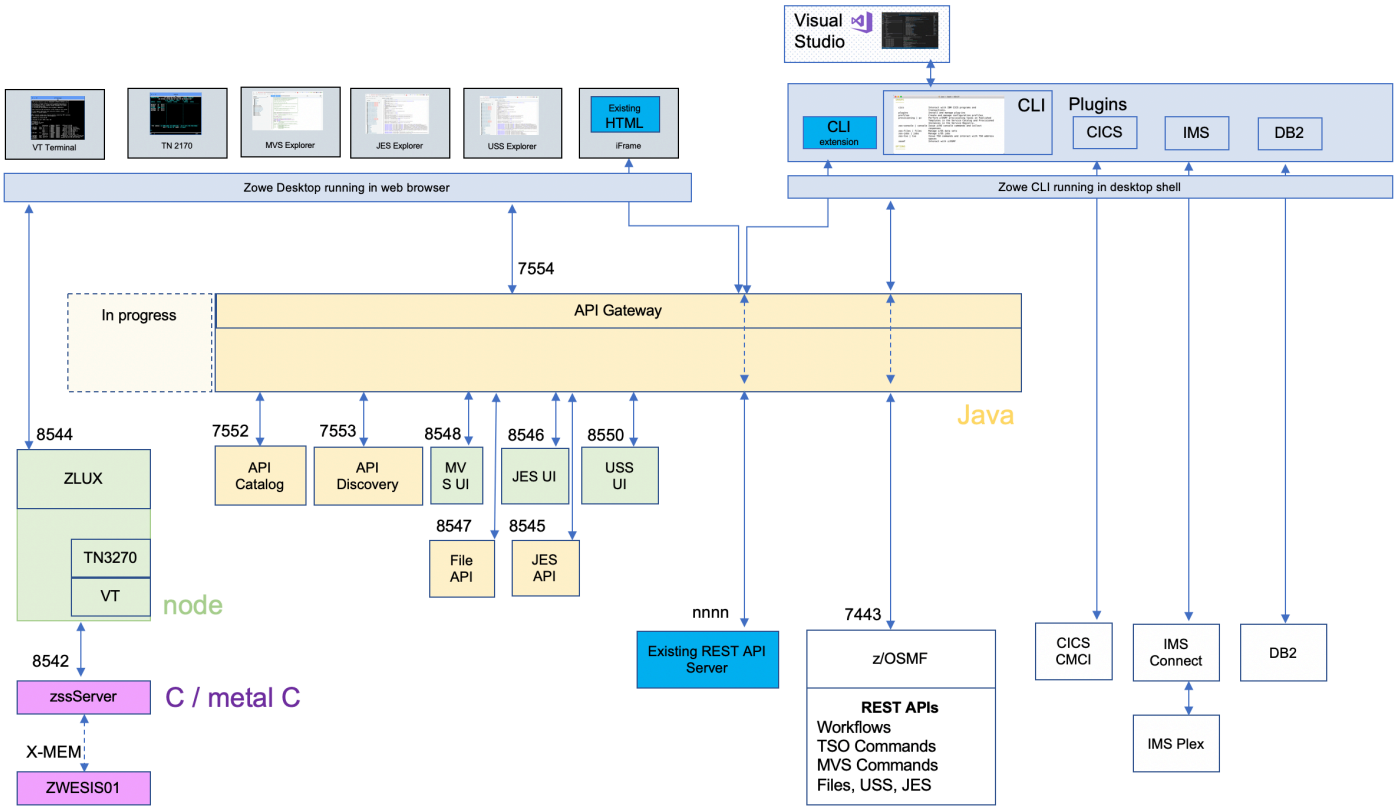
Zowe Third-Party Software Requirements and Bill of Materials

- Third-Party Software Requirements (TPSR)
- Bill of Materials (BOM)

Zowe architecture

Zowe™ is a collection of components that together form a framework that allows Z based functionality to be accessible across an organization. This includes exposing Z based components such as z/OSMF as Rest APIs. The framework provides an environment where other components can be included and exposed to a broader non-Z based audience.

The following diagram depicts the high level Zowe architecture.



Release notes

Learn about what is new, changed, or removed in Zowe™.

Zowe Version 1.8.0 and later releases include the following enhancements, release by release.

- Version 1.8.0 (February 2020)
- Version 1.7.1 (December 2019)
- Version 1.7.0 (November 2019)
- Version 1.6.0 (October 2019)
- Version 1.5.0 (September 2019)
- Zowe SMP/E Alpha (August 2019)
- Version 1.4.0 (August 2019)
- Version 1.3.0 (June 2019)
- Version 1.2.0 (May 2019)
- Version 1.1.0 (April 2019)
- Version 1.0.1 (March 2019)
- Version 1.0.0 (February 2019)

Version 1.8.0 (January 2020)

New features and enhancements

The following features and enhancements were added.

Installation of Zowe z/OS components

- The installation now just needs two parameters configured: the USS location of the runtime directory and a data set prefix where a SAMPLIB and LOADLIB will be created. The runtime directory permissions are set to 755 and when Zowe is run, no data is written to the runtime directory.
- The way to configure Zowe is changed. Previously, you configured Zowe at installation time with the `zowe-install.yaml` file. This file has been removed and is no longer used in this release.
- A new directory `zowe-instance-dir` has been introduced that contains configuration data used to launch Zowe. This allows more than one Zowe instance to be started from the same Zowe runtime directory. A new file `zowe-instance.env` within each `zowe-instance-dir` directory controls which ports are allocated to the Zowe servers as well as location of any dependencies such as Java, z/OSMF or node. No configuration data is specified at install time. The data is only read, validated and used at launch time. The `zowe-instance.env` file contains a parameter value `LAUNCH_COMPONENT_GROUPS` that allows you to control which Zowe subsystems to launch, for example you can run the Zowe desktop and not the API Mediation Layer, or vice-versa; you can run just the API Mediation Layer and not the Zowe desktop. The `zowe-instance-dir` directory is also where log files are collected. Static extensions to the API Mediation Layer are recorded in the Zowe instance directory as well as any plug-in extensions to the Zowe desktop. This allows the runtime directory to be fully replaced during PTF upgrades or moving to later Zowe releases while preserving configuration data and extension definitions that are held in the instance directory.
- A new directory `keystore-directory` has been introduced outside of the Zowe runtime directory which is where the Zowe certificate is held, as well as the truststore for public certificates from z/OS services that Zowe communicates to (such as z/OSMF). A keystore directory can be shared between multiple Zowe instances and across multiple Zowe runtimes.
- All configuration of z/OS security that was done by Unix shell scripts during installation and configuration has been removed. A JCL member `ZWESECUR` is provided that contains all of the JCL needed to configure permissions, user IDs and groups, and other steps to prepare and configure a z/OS environment to successfully run Zowe. Code is included for RACF, Top Secret, and ACF/2.
- The Zowe cross memory server installation script `zowe-install-apf-server.sh` is removed. In this release, the steps for configuring z/OS security are included in the `ZWESECUR` JCL member.
- Previously, Zowe runs its two started tasks under the user ID of `IZUSVR` and admin of `IZUADMIN`. These belong to z/OSMF and are no longer used in this release. Instead, Zowe includes two new user IDs of `ZWESVUSR` (for the main Zowe started task), `ZWESIUSR` (for the cross memory server), and `ZWEADMIN` as a group. These user IDs are defaults and different ones can be used depending on site preferences.
- Previously, the main Zowe started task is called `ZOWESVR`. Now it is called `ZWESVSTC`.
- Previously, the cross memory started task is called `ZWESIS01`. Now it is called `ZWESISTC`.
- The script `zowe-verify.sh` is no longer included with Zowe. Now the verification is done at launch time and dependent on the launch configuration parameters. It is no longer done with a generic script function that `zowe-verify.sh` used to provide.

For more information about how to install Zowe z/OS components, see Installation roadmap on page 59.

API Mediation Layer

- The API Catalog backend has been modified to support the OpenAPI 3.0 version. The API Catalog now supports the display of API documentation in the OpenAPI 3.0 format.
- A new Eureka metadata definition has been developed to enable service registration that does not require using existing pre-prepared enablers. Both new and old metadata versions are supported by the Discovery Service. Corresponding documentation to onboard a service with the Zowe API ML without an onboarding enabler has also been refactored.
- The plain Java enabler has been redesigned for simple and straight-forward API service configuration. Configuration parameters have been refactored to remove duplicates and unused parameters, and improve consistency with other parameters. Documentation to Onboard a REST API service with the Plain Java Enabler (PJE) has also been refactored.

Zowe App Server

- The app server now issues a message indicating it is ready, how many plug-ins loaded, and where it can be accessed from #355

- Restructured the App server directories to separate writable configuration items from read-only install content #911 #627 #87 #43
- Move install-app script to instance directory bin folder for ease of use #966
- Access control for app visibility 216
- The following features and enhancements were made in the default apps:
  - UI changes for write support for datasets in editor #340
  - Support for QSAM and VSAM deletion in the ZSS dataset REST API #339
  - Editor: Dataset deletion capability#229
  - Editor: File deletion UI changes #338
  - Editor fix: When saving a new file use the opened directory in the dialog #233
  - Editor fix: Disable text area for datasets in the absence of write ability #342
  - Editor fix: When saving a new file use the opened directory in the dialog #233

Zowe CLI

- The Zowe CLI REST API now supports the following capabilities for managing data sets:
  - Rename sequential and partitioned data sets. #571
  - Migrate data sets. #558
  - Copy data sets to another data set and copy members to another member. #578
- The Zowe CLI REST API now supports HTTP ETags in response data. The ETag mechanism allows client applications to cache data more efficiently. ETAgs can also prevent simultaneous, conflicting updates to a resource. #598

Zowe Explorer

Review the Zowe Explorer Change Log to learn about the latest features, enhancements, and fixes.

You can install the latest version of the extension from the Visual Studio Code Marketplace .

Check the new "Getting Started with Zowe Explorer" video to learn how to install and get started with the extension. For more information, see Zowe Explorer Extension for VSCode.

Bug fixes

The following bugs were fixed.

Zowe App Server

- Use of environment variables (_TAG_REDIR_XXX) required to run Zowe with node v12 #333
- Install-app.sh sh would not work without first server run, improper permissions #373

Zowe CLI

- Fixed an issue where `zowe zos-jobs submit stdin` command returned an error when handling data from standard in. #601
- Updated dependencies to address potential vulnerabilities. Most notably, Yargs is upgraded from v8.0.2 to v15.0.2. #333

Version 1.7.1 (December 2019)

New features and enhancements

The following features and enhancements were added.

Zowe App Server

- A backup routine for when a non-administrator tries to access the API. Instead of executing privileged commands and failing, it will execute a command to get their profile, and return only the information in their scope. This is a feature that most people won't need, since you'd ideally want to be an administrator if you were using this API, but the functionality is there. (#114)

- The ability to retrieve profiles only by prefix. This can be done by looking for a profile with a "." at the end. This will act as a wildcard which extracts everything matching that prefix. (#114)

Zowe SMP/E installation

The pre-release of the Zowe SMP/E build is updated to be based on Zowe Version 1.7.1.

Bug fixes

The following bugs were fixed.

Zowe App Server

- Fixed a bug where the end of an acid is cut off when getting the access list of a group, resulting in invalid output in the response.(#114)
- Fixed a bug where all of the different administrator suffixes weren't defined, so it was incorrectly returning administrators. (#114)

Version 1.7.0 (November 2019)

New features and enhancements

The following features and enhancements were added.

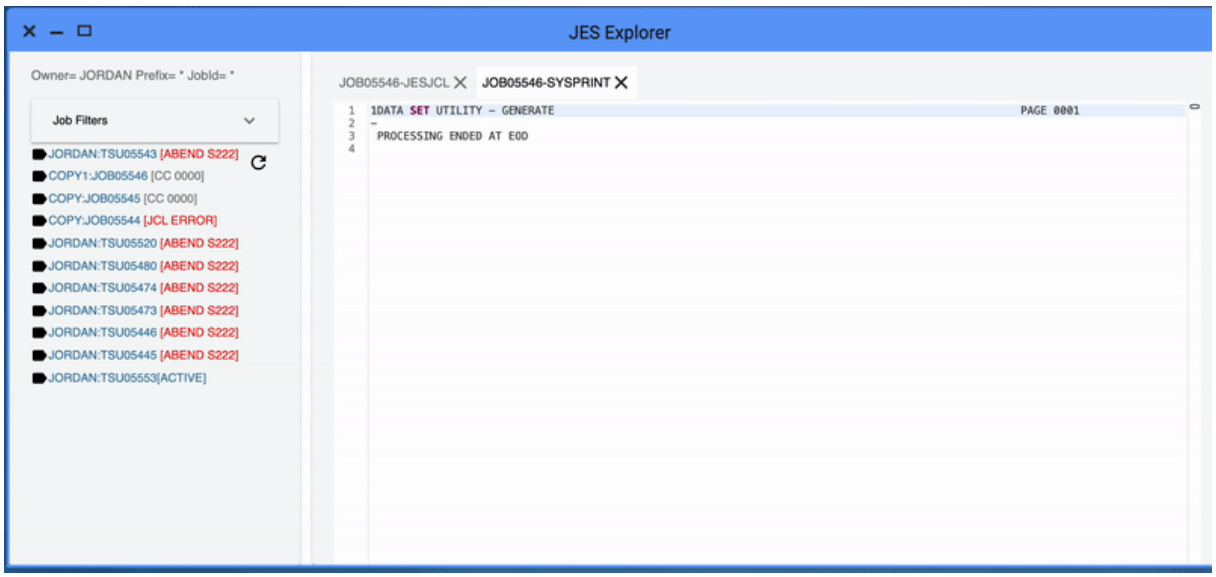
API Mediation Layer

- Cleanup Gateway dependency logs (#413)
- Cleanup Gateway - our code (#417)
- Cleanup Discovery Service dependency logs (#403)
- Cleanup Discovery Service - our code (#407)
- External option to activate DEBUG mode for APIML (#410)

Zowe App Server

- Introduced the "SJ" feature to the JES Explorer application (#282)

  You can now right-click a job label and click "Get JCL" to retrieve the JCL used to submit the job. This JCL can then be edited and resubmitted.



- File Explorer now offers a right click Delete option for files and folders (#43)

- Prevented creation/deletion of files and folders queued for deletion. (#48)
- Updated back-end API to give more accurate delete responses. (#93)
- IFrame adapter: added support for plugin definition, logger, and launch metadata. (#174)
- IFrame app-to-app communication support (#174)
- Removed unnecessary warning suppression (#23)
- Dispatcher always sends message, even when context doesn't exist (#174)
- Support constructor injectibles via Iframe adapter (#174)
- Browser tab for the desktop now includes opened app name. (#175)
- File Explorer now offers a right click file and folder Properties menu. (#180)
- File Explorer now offers a right click dataset Properties menu. (#49)
- Made it possible to specify config properties via command line arguments for the App server. (#81)
- Allow override of configuration attributes using a -D argument syntax. (#154)
- Allow specifying environment variables that can be interpreted as JSON structures. (#156)

Zowe Explorer (Extension for VSCode)

- The name of the extension was changed from "VSCode Extension for Zowe" to "Zowe Explorer".
- The VSCode Extention for Zowe contains various changes in this this release. For more information, see the VSCode Change Log.

Bug fixes

The following bugs were fixed.

API Mediation Layer

Fixed a typo in Gateway startup script. (#427)

Zowe App Server

Fixed notification click, time stamp, inconsistent notification manager pop up clicks, empty notification bubbles, and safari issue. (#171)

Zowe CLI

This version of Zowe CLI contains various bug fixes that address vulnerabilities.

Version 1.6.0 (October 2019)

No changes were made to API ML or Zowe CLI in this release.

What's new in the Zowe App Server

The following features and enhancements are added:

- Added two NodeJS issues to the App Framework Troubleshooting section. #786
- Added a REST API for new core dataservices to administer the servers and plugins. #82
- Added pass through express router ws patcher in case plug-ins need it. #152, #149
- Updated security plugins to manage proxied headers so that unnecessary things are not put into the browser.#152, #26
- Clear cookie on complete logout.#152

What's new in Zowe CLI

The following enhancement was added:

- The `--wait-for-output` and the `--wait-for-active` options were added. You can append these options to a `zowe zos-jobs submit` command to either wait for the job to be active, or wait for the job to complete and enter OUTPUT status. If you do not specify `--vasc`, you can use these options to check job return codes without issuing `zowe zos-jobs view job-status-by-jobid <jobid>`.

What's new in the Visual Studio Code (VSC) Extension for Zowe

The Visual Studio Code (VSC) Extension for Zowe lets you interact with data sets and USS files from a convenient graphical interface. Review the Change Log to learn about the latest improvements to the extension.

You can download the latest version from the VSC Marketplace.

Version 1.5.0 (September 2019)

What's new in API Mediation Layer

The following features and enhancements are added:

- The Discovery Service UI now enables the user to log in using mainframe credentials or by providing a valid client certificate.
- API Catalog REST endpoints now accept basic authentication by requiring the user to provide a username and password.

The following bugs are fixed:

- A defect has been resolved where previously an authentication message was thrown in the service detail page in the API Catalog when the swagger JSON document link was clicked. The message requires the user to provide mainframe credentials but did not link to an option to authenticate. Now, a link is included to provide the user with the option to authenticate.

What's new in the Zowe App Server

The following features and enhancements are added:

- Adds dynamic logging functionality for plugins (#60, #63)
- Top Secret updates to the security lookup API (#71, #72, #74)
- Accept basic auth header as an option for login (#80)
- JSON parsing enhancements for UTF8, and printing to buffer (#67)
- Optimization, memory bugfix and improved tracing for authentication (#72)
- Performance optimization for app thumbnail snapshots: Fixed a bug causing slowdown relative to number of apps open (#131)
- Translations: Added missing language translations about session lifecycle (#137)
- Logger reorganized for Zowe-wide log format unification. Includes i18n-able message ID support & new info. See #90 (#17, #119, #116, #142, #35, #19, #132, #146, #126, #139, #67, #133, #21)
- Establish rules & recommendations for conformance (#142)
- Launchbar menu of apps now has same context menu properties as pinned apps (#140)
- Properties App now shows the ID of the chosen plugin (#140)
- Added group permission for plugin access when installing via install script (#125)
- Updated URIBroker include new parameter for searching datasets with included trailing qualifiers (#34, #138)
- App2App communication now allows you to target a specific app instance, as well as to request minimization or maximization (#38, #148)
- Configuration Dataservice now can load plugin defaults from the plugin's own folder (#129)
- Configuration Dataservice can now support GET like HEAD (#140)
- Configuration Dataservice can now utilize binaries as opposed to JSON. This mode does not process the objects, just stores & retrieves. (#130)
- Added a notification menu, popup & API where messages can be sent by administrators to individual or all end users (#36, #144)
- Doc: Configuration Dataservice Swagger document updated for new features (#136)
- Desktop now supports loading a custom wallpaper, and the launchbar & maximized window style has been changed to improve screen real estate (#151)
- The App Server configuration and log verbosity can now be viewed and updated on-the-fly via a REST API (#66, #128)

- The App Server environment parameters and log output can now be viewed via a REST API (#66, #128)
- The App Server can now have Application plugins added, removed, and upgraded on-the-fly via a REST API (#137, #69)
- A dataservice can now import another import dataservice, as long as this chain eventually resolves to a non-import dataservice (#139)
- You can now open any Zowe App in its own browser tab by right clicking its icon and choosing "Open in new browser window" (#149, #150)
- Icons improved for datasets that are migrated/archived (#30)
- Support App2App to open a given dataset (#87, #35)
- Navigate the editor menu bar via keyboard (#85)
- Add keyboard shortcuts to open and close tabs (#81)
- Add loading indicator for dataset loading (#34)
- Compress the terminals with gzip for improved initial load time, same as was done with the editor previously (#22, #23)
- Made the following enhancements to the JES Explorer App
  - Add ability to open and view multiple Spool files at once (#99)
  - Migrate from V0 to V1 of Material UI (#98)
  - Migrate from V15 to V16 of React (#98)

The following bugs are fixed:

- New directories/files from Unix file API would have no permissions (#75)
- Properties App can now be reused when clicking property of a second app (#140)
- Logout did not clear dispatcher App instance tracking (#32)
- Iframe Apps were not gaining mouse focus correctly (#37, #145)
- Remove placeholder swagger from swagger response when plugin-provided swagger is found (#139)
- ZSS Dataservices could fail due to incorrect impersonation environment variable setting (_BPX_SHAREAS) (#68)
- Restore focus of text on window restore (#84)
- Reposition menu from menu bar on edge/firefox (#82)
- Could not open the SSH terminal in single window mode (#21)

What's new in Zowe CLI and Plug-ins

The following commands and enhancements are added:

- You can append `--help-web` to launch interactive command help in your Web browser. For more information, see Interactive Web Help. (#238)

Zowe SMP/E Alpha (August 2019)

A pre-release of the Zowe SMP/E build is now available. This alpha release is based on Zowe Version 1.4.0. Do not use this alpha release in production environment.

- To obtain the SMP/E build, go to the Zowe Download website.
- For more information, see Installing Zowe SMP/E Alpha on page 65.

Version 1.4.0 (August 2019)

What's new in API Mediation Layer

This release of Zowe API ML contains the following improvements:

- JWT token configuration
  - RS256 is used as a token encryption algorithm
  - JWT secret string is generated at the time of installation and exported as a `.pem` file for use by other services
  - JWT secret string is stored in a key store in PKCS 11 format under "jwtsecret" name
- SonarQube problems fixed
  - Various fixes from SonarQube scan
- API Mediation Layer log format aligned with other Zowe services:

```
%d{yyyy-MM-dd HH:mm:ss.SSS,UTC} %clr(&lt;${logbackService:-
${logbackServiceName}}:%thread:${PID:- }&gt;){magenta} %X{userid:-}
 %clr(%-5level) %clr(\(%logger{15},%file:%line\)){cyan} %msg%n
```

- Added an NPM command to register certificates on Windows. The following command installs the certificate to trusted root certification authorities:

```
npm run register-certificates-win
```

- Cookie persistence changed
  - Changed the API Mediation Layer cookie from persistent to session. The cookie gets cleared between browser sessions.
- Fixed high CPU usage occurrence replicated in Broadcom (#282)
  - Changed configuration of LatencyUtils to decrease idle CPU consumption by API ML services
- API Mediation layer now builds using OpenJDK with OpenJ9 JVM

What's new in the Zowe App Server

Made the following fixes and enhancements:

- Added the ability for the App Server Framework to defer to managers for dataservices that are not written in NodeJS or C. The first implementation is a manager of Java servlet type dataservices, where the App Server manages Tomcat instances when Tomcat is present. (#158)
- Added a tomcat xml configuration file with substitutions for values (ports, keys, certificates) necessary for the App Server to manage one or more instances of Tomcat for hosting servlet dataservices. Also added a new section to the zluxserver.json file to describe dataservice providers such as the aforementioned Tomcat Java Servlet one. (#49)
- Added Swagger API documentation support. Application developers can include a Swagger 2.0 JSON or YAML file in the app's /doc/swagger directory for each REST data service. Each file must have the same name as the data service. Developers can then reference the files at runtime using a new app route: /ZLUX/plugins/PLUGINID/catalogs/swagger. They can reference individual services at: /ZLUX/plugins/PLUGINID/catalogs/swagger/SERVICENAME. If swagger documents are not present, the server will use contextual knowledge to show some default values. (#159)
- The following new REST and cross-memory services have been added (#32):
  - Extract RACF user profiles
  - Define/delete/permit general RACF resource profiles (limited to a single class)
  - Add/remove RACF groups
  - Connect users to RACF groups (for a limited set of group prefixes)
  - Check RACF user access levels (limited to a single class)
- Fixed multiple issues in the File Editor App. (#88)
- Fixed multiple ZSS file and dataset API issues (#49 #42 #40 #44 #45)
- Remove several CSS styles from the Desktop to prevent bleed-in of styles to Apps (#117)
- Fixed incorrect count of open Apps upon logging in more than once per browser session (#123) Add OMVS information API to uribroker (#116)
- Enhanced auth plugin structure for application framework that lists auth capabilities (#118 #14 #19)
- Improved searching for node libraries for dataservices within an plugin (#114)

- Editor & File Explorer Widget Changes
  - Unix directory listing now starts in the user's home directory (#16)
  - JCL syntax coloring revision (#73)
  - Cursor, scroll position and text selection is now kept while switching tabs in editor (#71)
  - Editor now scrolls tab bar to newest tab when opening, and tab scrolling improved when closing tabs (#69)
  - Tab name, tooltip, and scroll fixes (#55 #60 #63)
  - Change in double and single click behavior of file explorer widget (#21)
  - Fix to show language menu on new file (#62)
  - Fix to keep language menu within the bounds of app window (#59)
  - Fix to the delete file prompt (#61)
  - Fix to allow closing of multiple editor instances (#22)
  - Fix to query datasets correctly by making queries uppercase (#65)
- Fixed issue where the cascading position of new windows were wrong when that application was maximized. (#102)
- Fixed issue where the file tabs in File Editor app were vertically scrollable, and where the close button would not be accessible for long file names. (#170)
- Updated the package lock files in all repositories to fix vulnerable dependencies. (#163)
- Fixed an issue where the Desktop used the roboto-latin-regular font for all text, which would not display well with non-latin languages. Now the fallback font is sans-serif. (#118)

What's new in Zowe CLI and Plug-ins

You can now explore the Zowe CLI command help in an interactive online format. See Zowe CLI Web Help.

The following new commands and enhancements are added:

- The VSCode Extension for Zowe now supports manipulation of USS files. (#32)
- You can now archive z/OS workflows using a wildcard. (#435)
- The z/OS Workflows functionality is now exported to an API. Developers can leverage the exported APIs to create applications and scripts without going through the CLI layer. (#482)
- The CLI now exploits all "z/OS data set and file REST interface" options that are provided in z/OSMF v2.3. (#491)

The following bugs are fixed:

- Fixed an issue where examples for `zowe files list uss-files` were slightly incorrect. (#440)
- Improved error message for `zowe db2 call procedure` command. (#22)

Version 1.3.0 (June 2019)

What's new in API Mediation Layer

This release of Zowe API ML contains the following user experience improvements:

- Added authentication endpoints (/login, /query) to the API Gateway
- Added the Gateway API Swagger document (#305)
  - Fixed the bug that causes JSON response to set incorrectly when unauthenticated
  - Fixed error messages shown when a home page cannot be modified
- Added a new e2e test for GW, and update the detail service tile (#309)
- Removed a dependency of integration-enabler-java on the gateway-common (#302)
- Removed access to the Discovery service UI with basic authentication (#313)
- Fixed the issue with the connection logic on headers to pass in the websocket (#275)
- Fixed the bug 264: Bypass the API Gateway when the server returns 302 (#276)
- Fixed the issue that causes the API ML Services display as UP, and makes the API doc available in the Catalog regardless whether the API ML Services stop (#287)

- Fixed the issue that prevents the API Catalog to load under zLux 9 (314)

What's new in the Zowe App Server

Made the following fixes and enhancements:

- Added internationalization to the Angular and React sample applications. (#133)
- Made the following enhancements to the ZSS server:
  - Added support for Zowe on z/OS version 2.4. (#15)
  - Updated documentation for query parameter to file API. (#48)
- Made the following enhancements to security:
  - App Server session cookie is now a browser session cookie rather than having an expiration date. Expiration is now tracked on the server side. (#132, #97, #81)
  - Added a "mode=base64" option to the unixfile API. (#127)
- Added a port to the cookie name to differentiate multiple servers on same domain. (#95)
- Made the following fixes and enhancements to the Code Editor application:
  - Added a menu framework to provide options specific to the current file/data set type. (#131)
  - Added ISPF-like syntax highlighting for JCL. (#48)
  - Fixed an issue by notifying users if the editor cannot open a file or data set. (#148)
  - Fixed an issue with event behavior when a tab is closed. (#135)
  - Fixed an issue with not showing the content of files in Chrome and Safari. (#100)
  - Fixed an issue with files shown without alphabetical sorting. (#85)
- Made the following fixes and enhancements to the TN3270 application (#96):
  - Fixed an issue where the application could not be configured to default to a TLS connection.
  - Fixed an issue where it could not handle a TN3270 connection, only TN3270E. Improved preference saving. Administrators can now store default values for terminal mod type, codepage, and screen dimensions.
- Made the following fixes and enhancements for App2App for IFrames (#24, #107):
  - Fixed an issue with an exception when handling App2App communication with IFrames.
  - Added experimental support for App2App communication with an IFrame application as destination.
- Made the following enhancements to support TopSecret:
  - Added a user-profiles endpoint. (#113)
  - Added an endpoint extraction for groups. (#129)
- Fixed an issue with app names not being internationalized when translations were present. (#85)
- Fixed Russian language errors in translation files. (#100)
- Fixed several issues with using the Application Server as a proxy. (#93)
- Fixed an issue with the App Server throwing exceptions when authorization plugins were installed but not requested. (#94)
- Fixed an issue with ZSS consuming excessive CPU during download. (#147)
- Fixed documentation issue by replacing "zLUX" with "Zowe Application Framework" and "MVD" with "Zowe Desktop." (#214)
- Fixed an issue with an incorrect translation for word "Japanese" in Japanese. (#108)

What's new in Zowe CLI and Plug-ins

The following new commands and enhancements are added:

- Return a list of archived z/OSMF workflows with the `zowe zos-workflows list arw` command. (#391)
- Return a list of systems that are defined to a z/OSMF instance with the `zowe zosmf list systems` command. (#348)
- The `zowe uss issue ssh` command now returns the exit code of the shell command that you issued. (#359)
- The `zowe files upload dtu` command now supports the metadata file named `.zosattributes`. (#366)

The following bugs are fixed:

- Fixed an issue where `zowe workflow ls aw` commands with the `--wn` option failed if there was a space in the workflow name. (#356)
- Fixed an issue where `zowe zowe-files delete uss` command could fail when resource URL includes a leading forward-slash. (#343).

Version 1.2.0 (May 2019)

Version 1.2.0 contains the following changes since Version 1.1.0.

What's new in the Zowe installer

- Made the following installer improvements:
  - Check whether ICSF is configured before checking Node version to avoid runaway CPU.
  - Warn if the host name that is determined by the installer is not a valid IP address.
  - Fixed a bug where a numeric value is specified in ZOWE_HOST_NAME causing errors generating the Zowe certificate.
- Made the following improvements to the `zowe-check-prereqs.sh` script:
  - Improvements for checking and validating the telnet and ssh port required by the Zowe Desktop applications.

What's new in API Mediation Layer

This release of Zowe API ML contains the following user experience improvements:

- Prevented the Swagger UI container on the service detail page from spilling.
- Added a check for the availability of the z/OSMF URL contained in the configuration. z/OSMF is used to verify users logging into the Catalog.
- Made *PageNotFound* error visible only in the debug log level.
- Added zD&T-compatible ciphers and the TLS protocol restricted to 1.2.
- Introduced support for VSCode development.
- Introduced a common cipher configuration property.
- Fixed URL transformation defects.
- Fixed reporting that the Catalog is down when it is started before the Discovery Service.
- Removed the bean overriding error message from the log.
- Fixed the state manipulation mechanism in the Catalog. As a result, no restoring of the application state is performed.
- Fixed the Catalog routing mechanism for a users who is already logged in so that the user is not prompted to log in again.
- A timeout has been set for Catalog login when z/OSMF is not responding.
- A tile change in the Catalog is now propagated to the UI.
- Fixed a problem with an incorrect service homepage link in the Catalog.
- The Catalog Login button has been disabled when the login request is in progress.

What's new in the Zowe App Server

- Improved security by adding support for RBAC (Role Based Access Control) to enable Zowe to determine whether a user is authorized to access a dataservice.
- Added Zowe Desktop settings feature for specifying the Zowe desktop language.
- Added German language files.
- Fixed a bug by adding missing language files.
- Enabled faster load times by adding support for serving the Zowe Application Framework core components, such as the Desktop, as compressed files in gzip format.
- Added support for application plug-ins to serve static content, such as HTML, JavaScript, and images, to browsers in gzip and brotli compressed files.
- Fixed a Code Editor bug by separating browsing of files and data sets.

What's new in Zowe CLI and Plug-ins

The Zowe CLI core component contains the following improvements and fixes:

- The `zos-uss` command group is added to the core CLI. The commands let you issue Unix System Services shell commands by establishing an SSH connection to an SSH server. For more information, see #unique_92.
- The zowe `zos-workflows` command group now contains the following `active-workflow-details` options:
  - `--steps-summary-only | --sso (boolean)`: An optional parameter that lets you list (only) the steps summary.
  - `--skip-workflow-summary | --sws (boolean)`: An optional parameter that lets you skip the default workflow summary.
- Zowe CLI was updated to correct an issue where the `zowe zos-workflows start` command ignored the `-- workflow-name` argument.
- Updated and clarified the description the `-- overwrite` option for the `zowe zos-workflows create workflow-from-data-set` command and the `Zowe zos-workflows create workflow-from-uss-file` command.
- The CLI Reference Guide is featured on the Zowe Docs home page. The document is a comprehensive guide to commands and options in Zowe CLI.
- You can now click the links on the Welcome to Zowe help section and open the URL in a browser window. Note that the shell application must support the capability to display and click hyperlinks.

What's new in Zowe USS API

Made the following enhancements:

- Chtag detection and ascii/ebcdic conversion on GET & PUT requests. For details, see this issue.
- New optional header on GET Unix file content request to force conversion from ebcdic to ascii. For details, see this issue.
- New response header on GET Unix file content requests: E-Tag for overwrite detection and validation. For details, see this issue.
- Reintroduced PUT (update) Unix file content endpoint. For details, see this issue.
- Reintroduced DELETE Unix file content endpoint. For details, see this issue.
- Reintroduced POST (create) Unix file or directory endpoint. For details, see this issue.
- Fixed a problem with incorrect return error when the user requests to view contents of a USS folder they do not have permission to. Now it returns a 403 (Forbidden) error. For details, see this issue.

Version 1.1.0 (April 2019)

Version 1.1.0 contains the following changes since the last 1.0.x version.

What's new in Zowe system requirements

z/OSMF Lite is now available for non-production use such as development, proof-of-concept, demo and so on. It simplifies the setup of z/OSMF with only a minimal amount of z/OS customization, but provides key functions that are required. For more information, see Configuring z/OSMF Lite (for non-production use) on page 42.

What's new in the Zowe App Server

- Made the following user experience improvements:
  - Enabled the Desktop to react to session expiration information from the Zowe Application Server. If a user is active the Desktop renews their session before it expires. If a user appears inactive they are prompted and can click to renew the session. If they don't click, they are logged out with a session expired message.
  - Added the ability to programmatically dismiss popups created with the "zlux-widgets" popup manager.

- Made the following security improvements:
  - Encoded URIs shown in the App Server 404 handler, which prevents some browsers from loading malicious scripts.
  - Documented and support configuring HTTPS on ZSS.
  - For ZSS API callers, added HTTP response headers to instruct clients not to cache HTTPS responses from potentially sensitive APIs.
- Improved the Zowe Editor App by adding app2app communication support that allows the application to open requested directories, dataset listings, and files.
- Improved the Zowe App API by allowing subscription to close events on viewports instead of windows, which allows applications to better support Single App Mode.
- Fixed a bug that generated an extraneous RACF audit message when you started ZSS.
- Fixed a bug that would sometimes move application windows when you attempted to resized them.
- Fixed a bug in the "Getting started with the ZOWE WebUi" tutorial documentation.
- Fixed a bug that caused applications that made ZSS service requests to fail with an HTTP 401 error because of dropped session cookies.

What's new in the Zowe CLI and Plug-ins

This release of Zowe CLI contains the following new and improved capabilities:

- Added APIs to allow the definition of workflows
- Added the option `max-concurrent-requests` to the `zowe zos-files upload dir-to-uss` command
- Added the option `overwrite` to the `zowe zos-workflows create` commands
- Added the option `workflow-name` to the `zowe zos-workflows` commands
- Added the following commands along with their APIs:
  - `zowe zos-workflows archive active-workflow`
  - `zowe zos-workflows create workflow-from-data-set`
  - `zowe zos-workflows create workflow-from-uss-file`
  - `zowe zos-workflows delete active-workflow`
  - `zowe zos-files list uss-files`

This release of the Plug-in for IBM DB2 Database contains the following new and improved capabilities:

- Implemented command line precedence, which lets users issue commands without the need of a DB2 profile.
- The DB2 plug-in can now be influenced by the `ZOWE_OPT_` environment variables.

What's new in API Mediation Layer

- Made the following user experience improvements:
  - Documented the procedure for changing the log level of individual code components in *Troubleshooting API ML*.
  - Documented a known issue when the API ML stops accepting connections after z/OS TCP/IP is recycled in the *Troubleshooting API ML*.

Version 1.0.1 (March 2019)

Version 1.0.1 contains the following changes since the last version.

What's new in Zowe installation on z/OS

During product operation of the Zowe Cross Memory Server which was introduced in V1.0.0, the z/OSMF user ID IZUSVR or its equivalent must have UPDATE access to the BPX.SERVER and BPX.DAEMON FACILITY classes. The install script will do this automatically if the installing user has enough authority, or provide the commands to be issued manually if not. For more information, see Installing the Zowe Cross Memory Server on z/OS

What's new in the Zowe App Server

- Made the following improvements to security:
  - Removed the insecure SHA1 cipher from the Zowe App Server's supported ciphers list.
  - Added instructions to REST APIs to not cache potentially sensitive response contents.
  - Set secure attributes to desktop and z/OSMF session cookies.
- Fixed a bug that caused the configuration data service to mishandle PUT operations with bodies that were not JSON.
- Fixed a bug that prevented IFrame applications from being selected by clicking on their contents.
- Fixed various bugs in the File Explorer and updated it to use newer API changes.
- Fixed a bug in which App2App Communication Actions could be duplicated upon logging in a second time on the same desktop.

What's new in Zowe CLI

- Create and Manage z/OSMF Workflows using the new `zos-workflows` command group. For more information, see Zowe CLI command groups.
- Use the `@lts-incremental` tag when you install and update Zowe CLI core or plug-ins. The tag ensures that you don't consume breaking changes that affect your existing scripts. Installation procedures are updated to reflect this change.
- A Zowe CLI quick start on page 29 is now available for users who are familiar with command-line tools and want to get up and running quickly.
- IBM CICS Plug-in for Zowe CLI was updated to support communication over HTTPS. Users can enable https by specifying `--protocol https` when creating a profile or issuing a command. For backwards compatibility, HTTP remains the default protocol.

What's new in the Zowe REST APIs

Introduced new Unix files APIs that reside in the renamed API catalog tile `z/OS Datasets and Unix files service` (previously named `z/OS Datasets service`). You can use these APIs to:

- List the children of a Unix directory
- Get the contents of a Unix file

What's changed

- **Zowe explorer apps**
  - JES Explorer: Enhanced Info/Error messages to better help users diagnose problems.
  - MVS Explorer: Fixed an issue where Info/Error messages were not displayed when loading a Dataset/ Members contents.

Version 1.0.0 (February 2019)

Version 1.0.0 contains the following changes since the Open Beta release.

What's new in API Mediation Layer

- HTTPs is now supported on all Java enablers for onboarding API microservices with the API ML.
- SSO authentication using z/OSMF has been implemented for the API Catalog login. Mainframe credentials are required for access.

What's new in Zowe CLI

- **Breaking change to Zowe CLI**: The `--pass` command option is changed to `--password` for all core Zowe CLI commands for clarity and to be consistent with plug-ins. If you have zosmf profiles that you created prior to January 11, 2019, you must recreate them to use the `--password` option. The aliases `--pw` and `--pass` still function when you issue commands as they did prior to this breaking change. You do not need to modify scripts that use `--pass`.

- The `@next` npm tag used to install Zowe CLI is deprecated. Use the `@latest` npm tag to install the product with the online registry method.

What's new in the Zowe Desktop

- You can now obtain information about an application by right-clicking on an application icon and then clicking **Properties**.
- To view version information for the desktop, click the avatar in the lower right corner of the desktop.
- Additional information was added for the Workflow application.
- The titlebar of the active window is now colored to give an at-a-glance indication of which window is in the foreground.
- Window titlebar maximize button now changes style to indicate whether a window is maximized.
- Windows now have a slight border around them to help see boundaries and determine which window is active.
- Multiple instances of the same application can be opened and tracked from the launchbar. To open multiple instances, right-click and choose **Open New**. Once multiple instances are open, you can click the application icon to select which application to bring to the foreground. The number of orbs below the application icon relates to the number of instances of the application that is open.
- Desktop framework logging trimmed and formalized to the Zowe App Logger. For more information, see https://github.com/zowe/zlux/wiki/Logging.
- The UriBroker was updated to support dataservice versioning and UNIX file API updates.
- Removed error messages about missing `components.js` by making this optional component explicitly declared within an application. By using the property "webContent.hasComponents = true/false".
- Set the maximum username and password length for login to 100 characters each.
- Applications can now list webContent.framework = "angular" as an alias for "angular2".
- Fixed a bug where the desktop might not load on high latency networks.

What's new in the Zowe App Server

- HTTP support was disabled in favor of HTTPS-only hosting.
- The server can be configured to bind to specific IPs or to hostnames. Previously, the server would listen on all interfaces. For more information, see https://github.com/zowe/zlux-app-server/pull/30.
- The core logger prefixes for the Zowe App Server were changed from "_unp" to "_zsf".
- Dataservices are now versioned, and dataservices can depend on specific versions of other dataservices. A plug-in can include more than one version of a dataservice for compatibility. For more information, see https://github.com/zowe/zlux/wiki/ZLUX-Dataservices.
- Support to communicate with the API Mediation Layer with the use of keys and certificates was added.
- Trimmed and corrected error messages regarding unconfigured proxies for clarity and understanding. For more information, see https://github.com/zowe/zlux-server-framework/pull/33.
- Fixed the `nodeCluster.sh` script to have its logging and environment variable behavior consistent with `nodeServer.sh`.
- Removed the "swaggerui" plug-in in favor of the API Catalog.
- Bugfix for `/plugins` API to not show the installation location of the plug-in.
- Bugfix to print a warning if the server finds two plug-ins with the same name.
- Added the ability to conditionally add HTTP headers for secure services to instruct the browser not to cache the responses. For more information, see https://github.com/zowe/zlux-server-framework/issues/36.
- Added a startup check to confirm that ZSS is running as a prerequisite of the Zowe App Server.
- Bugfix for sending HTTP 404 response when content is missing, instead of a request hanging.
- Added tracing of login, logout, and HTTP routing so that administrators can track access.

What's changed

- Previously, APIs for z/OS Jobs services and z/OS Data Set services are provided unsing an IBM WebSphere Liberty web application server. In this release, they are provided using a Tomcat web application server. You can view the associated API documentation corresponding to the z/OS services through the API Catalog.

- References to `zlux-example-server` were changed to `zlux-app-server` and references to `zlux-proxy-server` were changed to `zlux-server-framework`.

Known issues

Paste operations from the Zowe Desktop TN3270 and VT Terminal applications

TN3270 App - If you are using Firefox, the option to use Ctrl+V to paste is not available. Instead, press Shift + right-click to access the paste option through the context menu.

Pressing Ctrl+V will perform paste for the TN3270 App on other browsers.

VT Terminal App - In the VT Terminal App, Ctrl+V will not perform a paste operation for any browser.

Note: In both terminals, press Shift + right-click to access copy and paste options through the context menu.

z/OS Subsystems App - The z/OS Subsystems application is being removed temporarily for the 1.0 release. The reason is that as the ZSS has transitioned from closed to open source some APIs needed to be re-worked and are not complete yet. Look for the return of the application in a future update.

Zowe CLI quick start

Get started with Zowe™ CLI quickly and easily.

Note: This section assumes some prerequisite knowledge of command-line tools and writing scripts. If you prefer more detailed instructions, see Installing Zowe CLI on page 104

- Installing on page 29
  - Installing Zowe CLI core on page 29
  - Installing CLI plug-ins on page 29
- Issuing your first commands on page 30
  - Listing all data sets under a high-level qualifier (HLQ) on page 30
  - Downloading a partitioned data-set (PDS) member to local file on page 30
- Using profiles on page 30
  - Profile types on page 30
  - Creating a zosmf profile on page 30
  - Using a zosmf profile on page 30
- Writing scripts on page 30
  - Example: on page 31
- Next Steps on page 31

Installing

Before you install Zowe CLI, download and install Node.js and npm.

Installing Zowe CLI core

```
npm config set @brightside:registry https://api.bintray.com/npm/ca/
brightside
```

```
npm install @brightside/core@lts-incremental -g
```

Installing CLI plug-ins

```
zowe plugins install @brightside/cics@lts-incremental @brightside/db2@lts-
incremental
```

The command installs the IBM CICS plug-in, but the IBM Db2 plug-in requires Installing on page 159.

For more information, see Installing Zowe CLI plug-ins on page 154.

Issuing your first commands

Issue `zowe --help` to display full command help. Append `--help` (alias `-h`) to any command to see available command actions and options.

To interact with the mainframe, type `zowe` followed by a command group, action, and object. Use options to specify your connection details such as password and system name.

Listing all data sets under a high-level qualifier (HLQ)

```
zowe zos-files list data-set "MY.DATASET.*" --host my.company.com --port 123
 --user myusername123 --pass mypassword123
```

Downloading a partitioned data-set (PDS) member to local file

```
zowe zos-files download data-set "MY.DATA.SET(member)" -f "mylocalfile.txt"
 --host my.company.com --port 123 --user myusername123 --pass mypassword123
```

See Command Groups for a list of available functionality.

Using profiles

Zowe profiles let you store configuration details such as username, password, host, and port for a mainframe system. Switch between profiles to quickly target different subsystems and avoid typing connection details on every command.

Profile types

Most command groups require a `zosmf-profile`, but some plug-ins add their own profile types. For example, the CICS plug-in has a `cics-profile`. The profile type that a command requires is defined in the `PROFILE OPTIONS` section of the help response.

Tip: The first `zosmf` profile that you create becomes your default profile. If you don't specify any options on a command, the default profile is used. Issue `zowe profiles -h` to learn about listing profiles and setting defaults.

Creating a zosmf profile

```
zowe profiles create zosmf-profile myprofile123 --host my.company.com --port
 123 --user myusername123 --password mypassword123
```

Note: The port defaults to 443 if you omit the `--port` option. Specify a different port if your host system does not use port 443.

Using a zosmf profile

```
zowe zos-files download data-set "MY.DATA.SET(member)" -f "mylocalfile.txt"
 --zosmf-profile myprofile123
```

For detailed information about issuing commands, using profiles, and storing variables as environment variables, see Defining Zowe CLI connection details on page 122

Writing scripts

You can write Zowe CLI scripts to streamline your daily development processes or conduct mainframe actions from an off-platform automation tool such as Jenkins or TravisCI.

Example:

You want to delete a list of temporary datasets. Use Zowe CLI to download the list, loop through the list, and delete each data set using the `zowe zos-files delete` command.

```
#!/bin/bash

set -e

# Obtain the list of temporary project data sets
dslist=$(zowe zos-files list dataset "my.project.ds*")

# Delete each data set in the list
IFS=$'\n'
for ds in $dslist
do
     echo "Deleting Temporary Project Dataset: $ds"
     zowe files delete ds "$ds" -f
done
```

For more information, see Writing scripts to automate mainframe actions.

Next Steps

You successfully installed Zowe CLI, issued your first commands, and wrote a simple script! Next, you might want to:

- Review Command Groups to learn what functionality is available, and explore the in-product help.
- Learn about Defining Environment Variables on page 124 to store configuration options.
- Integrate your scripts with an automation server like Jenkins.
- See what Extending Zowe CLI on page 154 for the CLI.
- Learn about Developing a new plug-in on page 231 (contributing to core and developing plug-ins).

Frequently Asked Questions

Check out the following FAQs to learn more about the purpose and function of Zowe™.

- Zowe FAQ on page 31
- Zowe CLI FAQ on page 33

Zowe FAQ

What is Zowe?

Click to show answer

Zowe is an open source project within the Open Mainframe Project that is part of The Linux Foundation. The Zowe project provides modern software interfaces on IBM z/OS to address the needs of a variety of modern users. These interfaces include a new web graphical user interface, a script-able command-line interface, extensions to existing REST APIs, and new REST APIs on z/OS.

Who is the target audience for using Zowe?

Click to show answer

Zowe technology can be used by a variety of mainframe IT and non-IT professionals. The target audience is primarily application developers and system programmers, but the Zowe Application Framework is the basis for developing web browser interactions with z/OS that can be used by anyone.

What language is Zowe written in?

Click to show answer

Zowe consists of several components. The primary languages are Java and JavaScript. Zowe CLI is written in TypeScript.

What is the licensing for Zowe?
Click to show answer

Zowe source code is licensed under EPL2.0. For license text click here and for additional information click here.

In the simplest terms (taken from the FAQs above) - "...if you have modified EPL-2.0 licensed source code and you distribute that code or binaries built from that code outside your company, you must make the source code available under the EPL-2.0."

Why is Zowe licensed using EPL2.0?
Click to show answer

The Open Mainframe Project wants to encourage adoption and innovation, and also let the community share new source code across the Zowe ecosystem. The open source code can be used by anyone, provided that they adhere to the licensing terms.

What are some examples of how Zowe technology might be used by z/OS products and applications?
Click to show answer

The Zowe Desktop (web user interface) can be used in many ways, such as to provide custom graphical dashboards that monitor data for z/OS products and applications.

Zowe CLI can also be used in many ways, such as for simple job submission, data set manipulation, or for writing complex scripts for use in mainframe-based DevOps pipelines.

The increased capabilities of RESTful APIs on z/OS allows APIs to be used in programmable ways to interact with z/OS services.

What is the best way to get started with Zowe?
Click to show answer

Zowe provides a convenience build that includes the components released-to-date, as well as IP being considered for contribution, in an easy to install package on Zowe.org. The convenience build can be easily installed and the Zowe capabilities seen in action.

To install the complete Zowe solution, see Introduction on page 36.

To get up and running with the Zowe CLI component quickly, see Zowe CLI quick start on page 29.

What are the prerequisites for Zowe?
Click to show answer

The primary prerequisites is Java on z/OS and the z/OS Management Facility enabled and configured. For a complete list of software requirements listed by component, see System requirements on page 36.

How is access security managed on z/OS?
Click to show answer

Zowe components use typical z/OS System authorization facility (SAF) calls for security.

How is access to the Zowe open source managed?
Click to show answer

The source code for Zowe is maintained on an Open Mainframe Project GitHub server. Everyone has read access. "Committers" on the project have authority to alter the source code to make fixes or enhancements. A list of Committers is documented in Committers to the Zowe project.

How do I get involved in the open source development?
Click to show answer

The best way to get started is to join a Zowe Slack channel and/or email distribution list and begin learning about the current capabilities, then contribute to future development.

For more information about emailing lists, community calendar, meeting minutes, and more, see the Zowe Community GitHub repo.

For information and tutorials about extending Zowe with a new plug-in or application, see Onboarding Overview on page 164 on Zowe Docs.

When will Zowe be completed?

Click to show answer

Zowe will continue to evolve in the coming years based on new ideas and new contributions from a growing community.

Can I try Zowe without a z/OS instance?

Click to show answer

IBM has contributed a free hands-on tutorial for Zowe. Visit the Zowe Tutorial page to learn about adding new applications to the Zowe Desktop and and how to enable communication with other Zowe components.

The Zowe community is also currently working to provide a vendor-neutral site for an open z/OS build and sandbox environment.

Zowe is also compatible with IBM z/OSMF Lite for non-production use. For more information, see Configuring z/OSMF Lite (for non-production use) on page 42 on Zowe Docs.

Zowe CLI FAQ

Why might I use Zowe CLI versus a traditional ISPF interface to perform mainframe tasks?

Click to show answer

For developers new to the mainframe, command-line interfaces might be more familiar than an ISPF interface. Zowe CLI lets developers be productive from day-one by using familiar tools. Zowe CLI also lets developers write scripts that automate a sequence of mainframe actions. The scripts can then be executed from off-platform automation tools such as Jenkins automation server, or manually during development.

With what tools is Zowe CLI compatible?

Click to show answer

Zowe CLI is very flexible; developers can integrate with modern tools that work best for them. It can work in conjunction with popular build and testing tools such as Gulp, Gradle, Mocha, and Junit. Zowe CLI runs on a variety of operating systems, including Windows, macOS, and Linux. Zowe CLI scripts can be abstracted into automation tools such as Jenkins and TravisCI.

Where can I use the CLI?

Click to show answer

Usage Scenario	Example
Interactive use, in a command prompt or bash terminal.	Perform one-off tasks such as submitting a batch job.
Interactive use, in an IDE terminal	Download a data set, make local changes in your editor, then upload the changed dataset back to the mainframe.
Scripting, to simplify repetitive tasks	Write a shell script that submits a job, waits for the job to complete, then returns the output.
Scripting, for use in automated pipelines	Add a script to your Jenkins (or other automation tool) pipeline to move artifacts from a mainframe development system to a test system.

Which method should I use to install Zowe CLI?

Click to show answer

You can install Zowe CLI using the following methods:

- **Local package installation:** The local package method lets you install Zowe CLI from a zipped file that contains the core application and all plug-ins. When you use the local package method, you can install Zowe CLI in an offline environment. We recommend that you download the package and distribute it internally if your site does not have internet access.
- **Online NPM registry:** The online NPM (Node Package Manager) registry method unpacks all of the files that are necessary to install Zowe CLI using the command line. When you use the online registry method, you need an internet connection to install Zowe CLI

How can I get help with using Zowe CLI?

Click to show answer

- You can get help for any command, action, or option in Zowe CLI by issuing the command 'zowe --help'.
- For information about the available commands in Zowe CLI, see Command Groups.
- If you have questions, the Zowe Slack space is the place to ask our community!

How can I use Zowe CLI to automate mainframe actions?

Click to show answer

- You can automate a sequence of Zowe CLI commands by writing bash scripts. You can then run your scripts in an automation server such as Jenkins. For example, you might write a script that moves your Cobol code to a mainframe test system before another script runs the automated tests.
- Zowe CLI lets you manipulate data sets, submit jobs, provision test environments, and interact with mainframe systems and source control management, all of which can help you develop robust continuous integration/ delivery.

How can I contribute to Zowe CLI?

Click to show answer

As a developer, you can extend Zowe CLI in the following ways:

- Build a plug-in for Zowe CLI
- Contribute code to the core Zowe CLI
- Fix bugs in Zowe CLI or plug-in code, submit enhancement requests via GitHub issues, and raise your ideas with the community in Slack.

Note: For more information, see How can I contribute? on page 224.

Chapter 2

User Guide

Topics:

- Planning and preparing the installation
- Installing Zowe z/OS components
- Installing Zowe CLI
- Advanced Zowe configuration
- Using Zowe
- Zowe CLI extensions and plug-ins

Planning and preparing the installation

Introduction

The installation of Zowe™ consists of two independent processes: installing the Zowe runtime on z/OS and installing Zowe CLI on a desktop computer.

The Zowe z/OS runtime provides a web desktop that runs in a web browser providing a number of applications for z/OS users, together with an API mediation layer that provides capabilities useful for z/OS developers.

Zowe CLI can connect to z/OS servers and allows tasks to be performed through a command line interface.

- A desktop computer that accesses the Zowe z/OS runtime through a web browser or REST API client does not need to have Zowe CLI installed.
- The z/OS servers that Zowe CLI connects to does not require the Zowe z/OS components to be installed on those servers.
- A desktop computer that uses Zowe CLI does not require the Zowe z/OS runtime to be installed on the z/OS server.

Before you start the installation, review the information on system requirements and other considerations.

Planning the installation of Zowe z/OS components

The following information is required during the installation process of the Zowe z/OS components.

- The zFS directory where you will install the Zowe runtime files and folders.
- A HLQ that the installation can create a load library and samplib containing load modules and JCL samples required to run Zowe.
- Multiple instances of Zowe can be started from the same Zowe z/OS runtime. Each launch of Zowe has its own zFS directory that is known as an instance directory.
- Zowe uses a zFS directory to contain its northbound certificate keys as well as a truststore for its southbound keys. Northbound keys are one presented to clients of the Zowe desktop or Zowe API Gateway, and southbound keys are for servers that the Zowe API gateway connects to. The certificate directory is not part of the Zowe runtime so that it can be shared between multiple Zowe runtimes and have its permissions secured independently.
- Zowe has two started tasks.
  - `ZWESVSVR` brings up the Zowe runtime containing the Zowe desktop, the API mediation layer and a number of Zowe applications.
  - `ZWESISTC` is a cross memory server that the Zowe desktop uses to perform APF-authorized code. More details on the cross memory server are described in Installing and configuring the Zowe cross memory server (ZWESISTC) on page 97.

    In order for the two started tasks to run correctly, security manager configuration needs to be performed. This is documented in Configuring the z/OS system for Zowe on page 83 and a sample JCL member `ZWESECUR` is shipped with Zowe that contains commands for RACF, TopSecret, and ACF2 security managers.

System requirements

Before installing Zowe™, ensure that your environment meets the prerequisites.

- Common sytem requirements
- Zowe Application Framework requirements
- Zowe CLI requirements

Common z/OS system requirements (host)

- z/OS Version 2.2 or later.

- IBM z/OS Management Facility (z/OSMF) Version 2.2, Version 2.3 or Version 2.4.

  z/OSMF is an optional prerequisite for Zowe. It is recomended that z/OSMF is present to fully exploit Zowe's capabilities.

  ::: tip

  - For non-production use of Zowe (such as development, proof-of-concept, demo), you can customize the configuration of z/OSMF to create what is known as "z/OS MF Lite" that simplifies the setup of z/OSMF. As z/OS MF Lite only supports selected REST services (JES, DataSet/File, TSO and Workflow), you will observe considerable improvements in start up time as well as a reduction in the efforts involved in setting up z/OSMF. For information about how to set up z/OSMF Lite, see Configuring z/OSMF Lite (for non-production use) on page 42
  - For production use of Zowe, see Configuring z/OSMF on page 39. :::

Zowe Application Framework requirements (host)

- Node.js versions between v6.14.4 and v8.x *on the z/OS host* where you install the Zowe Application Server. To install Node.js on z/OS, follow the instructions in Installing Node.js on z/OS on page 37.
- IBM SDK for Java Technology Edition V8 or later
- 833 MB of zFS file space
- (client) Supported browsers:
  - Google Chrome V66 or later
  - Mozilla Firefox V60 or later
  - Safari V12.0 or later
  - Microsoft Edge 17 (Windows 10)

  Each release of the Zowe Application Framework is tested to work on the current releases of Chrome, Firefox, Safari, and Edge, as well as the oldest release within a 1 year time span, unless the current release is also older than 1 year. For Firefox, the oldest supported release will also be from the Extended Support Release (ESR) version of Firefox, to ensure compatibility in those enterprise environments. This scheme for browser support is to ensure that Zowe works on the vast majority of browsers that people are currently using, while still allowing for use of new features and security that browsers constantly add.

Zowe CLI requirements (client)

Zowe CLI is supported on platforms where Node.js 8.0 or 10 is available, including Windows, Linux, and Mac operating systems.

- **Node.js V8.0 or later** on your computer

  Tip: You might need to restart the command prompt after installing Node.js. Issue the command `node --version` to verify that Node.js is installed. As a best practice, we recommend that you update Node.js regularly to the latest Long Term Support (LTS) version.
- **Node Package Manager V5.0 or later** on your computer.

  npm is included with the Node.js installation. Issue the command `npm --version` to verify that npm is installed.

Free disk space

Zowe CLI requires approximately **100 MB** of free disk space. The actual quantity of free disk space consumed might vary depending on the operating system where you install Zowe CLI.

Installing Node.js on z/OS

Before you install Zowe™, you must install IBM SDK for Node.js on the same z/OS server that hosts the Zowe Application Server. To install Node.js for Zowe, you can follow the steps in this topic or in the IBM SDK for Node.js - z/OS documentation.

Note: If you follow the steps in the Node.js documentation to install Node.js, you do **NOT** need to install Python, Make, Perl, or C/C++ runtime or compiler, which might be listed as prerequisites there. These software packages are

NOT required by Zowe. If you can execute `node --version` successfully, you have installed the prerequisites required by Zowe.

How to obtain IBM SDK for Node.js - z/OS

You can obtain IBM SDK for Node.js - z/OS for free in one of the following ways:

- Order the SMP/E version through your IBM representative for production use
- Use the pax evaluation for non-production deployments

For details, see the blog "How to obtain IBM SDK for Node.js - z/OS, at no charge".

Known issue: There is a known issue with node.js v8.16.1 and Zowe desktop encoding. See https://github.com/ibmruntimes/node/issues/142 for details.

Workaround: Use node.js v8.16.2 which is available at https://www.ibm.com/ca-en/marketplace/sdk-nodejs-compiler-zos. Download the `pax.Z` file under **IBM SDK for Node.js - z/OS, V8 - (v8.16.2 - November 2019 Build)**. Note that node.js v12 is not yet supported on Zowe.

Hardware and software requirements

To install Node.js for Zowe, the following requirements must be met.

Hardware:

IBM zEnterprise® 196 (z196) or newer

Software:

- Node.js Version 6 (see IBM Knowledge Center for all prerequisites):
  - z/OS V2R2 with PTF UI46658 or z/OS V2R3
- Node.js Version 8 (see IBM Knowledge Center for all prerequisites):
  - z/OS 2.2: PTFs UI62788, UI46658, UI62416 (APARs PH10606, PI79959, PH10740)
  - z/OS 2.3: PTFs UI61308, UI61376, and UI61747 (APARs PH07107, PH08353, and PH09543)
- z/OS UNIX System Services enabled
- Integrated Cryptographic Service Facility (ICSF) configured and started.

Installing the PAX evaluation version of Node.js -z/OS

Follow these steps to installing the PAX evaluation version of Node.js - z/OS to run Zowe.

1. Download the pax.Z file from the Download section to a z/OS machine.
2. Extract the pax.Z file inside an installation directory of your choice. For example:

   ```
   pax -rf <path_to_pax.Z_file> -x pax
   ```
3. Add the full path of your installation directory to your PATH environment variable:

   ```
   export PATH=<installation_directory>/bin/:$PATH
   ```
4. Run the following command from the command line to verify the installation.

   ```
   node --version
   ```

   If Node.js is installed correctly, the version of Node.js is displayed.
5. After you install Node.js, set the *NODE_HOME* environment variable to the directory where Node.js is installed. For example, `NODE_HOME=/proj/mvd/node/installs/node-v6.14.4-os390-s390x`.

To troubleshoot or install the SMP/E version of Node.js, see the documentation for IBM SDK for Node.js - z/OS. Remember that the software packages Perl, Python, Make, or C/C++ runtime or compiler that the Node.js documentation might mention are **NOT** needed by Zowe.

Configuring z/OSMF

The following information contains procedures and tips for meeting z/OSMF requirements. For complete information, go to IBM Knowledge Center and read the following documents.

- IBM z/OS Management Facility Configuration Guide
- IBM z/OS Management Facility Help

z/OS requirements

Ensure that the z/OS system meets the following requirements:

Requirements	Description	Resources in IBM Knowledge Center
Integrated Cryptographic Service Facility (ICSF)	On z/OS, Node requires ICSF to be installed, configured and started.	N/A
AXR (System REXX)	z/OS uses AXR (System REXX) component to perform Incident Log tasks. The component enables REXX executable files to run outside of conventional TSO and batch environments.	System REXX
Common Event Adapter (CEA) server	The CEA server, which is a co-requisite of the Common Information Model (CIM) server, enables the ability for z/OSMF to deliver z/OS events to C-language clients.	Customizing for CEA
Common Information Model (CIM) server	z/OSMF uses the CIM server to perform capacity-provisioning and workload-management tasks. Start the CIM server before you start z/OSMF (the IZU* started tasks).	Reviewing your CIM server setup
CONSOLE and CONSPROF commands	The CONSOLE and CONSPROF commands must exist in the authorized command table.	Customizing the CONSOLE and CONSPROF commands
Java level	IBM® 64-bit SDK for z/OS®, Java Technology Edition V8 or later is required.	Software prerequisites for z/OSMF
TSO region size	To prevent **exceeds maximum region size** errors, verify that the TSO maximum region size is a minimum of 65536 KB for the z/OS system.	N/A
User IDs	User IDs require a TSO segment (access) and an OMVS segment. During workflow processing and REST API requests, z/OSMF might start one or more TSO address spaces under the following job names: userid; substr(userid, 1, 6) CN (Console).	N/A

Configuring z/OSMF

Follow these steps:

1. From the console, issue the following command to verify the version of z/OS:

   ```
   /D IPLINFO
   ```

   Part of the output contains the release, for example,

   ```
   RELEASE z/OS 02.02.00.
   ```

2. Configure z/OSMF.

   z/OSMF is a base element of z/OS V2.2 and V2.3, so it is already installed. But it might not be configured and running on every z/OS V2.2 and V2.3 system.

   In short, to configure an instance of z/OSMF, run the IBM-supplied jobs IZUSEC and IZUMKFS, and then start the z/OSMF server. The z/OSMF configuration process occurs in three stages, and in the following order:

   - Stage 1 - Security setup
   - Stage 2 - Configuration
   - Stage 3 - Server initialization

   This stage sequence is critical to a successful configuration. For complete information about how to configure z/OSMF, see Configuring z/OSMF if you use z/OS V2.2 or Setting up z/OSMF for the first time if V2.3.

Note: In z/OS V2.3, the base element z/OSMF is started by default at system initial program load (IPL). Therefore, z/OSMF is available for use as soon as you set up the system. If you prefer not to start z/OSMF automatically, disable the autostart function by checking for `START` commands for the z/OSMF started procedures in the *COMMNDxx parmlib* member.

The z/OS Operator Consoles task is new in Version 2.3. Applications that depend on access to the operator console such as Zowe™ CLI's RestConsoles API require Version 2.3.

1. Verify that the z/OSMF server and angel processes are running. From the command line, issue the following command:

   ```
   /D A,IZU*
   ```

   If jobs IZUANG1 and IZUSVR1 are not active, issue the following command to start the angel process:

   ```
   /S IZUANG1
   ```

   After you see the message ""CWWKB0056I INITIALIZATION COMPLETE FOR ANGEL"", issue the following command to start the server:

   ```
   /S IZUSVR1
   ```

   The server might take a few minutes to initialize. The z/OSMF server is available when the message ""CWWKF0011I: The server zosmfServer is ready to run a smarter planet."" is displayed.

2. Issue the following command to find the startup messages in the SDSF log of the z/OSMF server:

```
f IZUG349I
```

You could see a message similar to the following message, which indicates the port number:

```
IZUG349I: The z/OSMF STANDALONE Server home page can be accessed at
 https://mvs.hursley.ibm.com:443/zosmf after the z/OSMF server is started
 on your system.
```

In this example, the port number is 443. You will need this port number later.

Point your browser at the nominated z/OSMF STANDALONE Server home page and you should see its Welcome Page where you can log in.

Note: If your implementation uses an external security manager other than RACF (for example, CA Top Secret for z/OS or CA ACF2 for z/OS), you provide equivalent commands for your environment. For more information, see the following product documentation:

- Configure z/OS Management Facility for CA Top Secret
- Configure z/OS Management Facility for CA ACF2

z/OSMF REST services for the Zowe CLI

The Zowe CLI uses z/OSMF Representational State Transfer (REST) APIs to work with system resources and extract system data. Ensure that the following REST services are configured and available.

z/OSMF REST services	Requirements	Resources in IBM knowledge Center
Cloud provisioning services	Cloud provisioning services are required for the Zowe CLI CICS and Db2 command groups. Endpoints begin with `/zosmf/provisioning/`	Cloud provisioning services
TSO/E address space services	TSO/E address space services are required to issue TSO commands in the Zowe CLI. Endpoints begin with `/zosmf/tsoApp`	TSO/E address space services
z/OS console services	z/OS console services are required to issue console commands in the Zowe CLI. Endpoints begin with `/zosmf/restconsoles/`	z/OS console
z/OS data set and file REST interface	z/OS data set and file REST interface is required to work with mainframe data sets and UNIX System Services files in the Zowe CLI. Endpoints begin with `/zosmf/restfiles/`	z/OS data set and file interface
z/OS jobs REST interface	z/OS jobs REST interface is required to use the zos-jobs command group in the Zowe CLI. Endpoints begin with `/zosmf/restjobs/`	z/OS jobs interface

z/OSMF REST services	Requirements	Resources in IBM knowledge Center
z/OSMF workflow services	z/OSMF workflow services is required to create and manage z/OSMF workflows on a z/OS system. Endpoints begin with `/zosmf/workflow/`	z/OSMF workflow services

Zowe uses symbolic links to the z/OSMF `bootstrap.properties`, `jvm.security.override.properties`, and `ltpa.keys` files. Zowe reuses SAF, SSL, and LTPA configurations; therefore, they must be valid and complete.

For more information, see Using the z/OSMF REST services in IBM z/OSMF documentation.

To verify that z/OSMF REST services are configured correctly in your environment, enter the REST endpoint into your browser. For example: https://mvs.ibm.com:443/zosmf/restjobs/jobs

Notes:

- Browsing z/OSMF endpoints requests your user ID and password for defaultRealm; these are your TSO user credentials.
- The browser returns the status code 200 and a list of all jobs on the z/OS system. The list is in raw JSON format.

Configuring z/OSMF Lite (for non-production use)

This section provides information about requirements for z/OSMF Lite configuration.

Disclaimer: z/OSMF Lite can be used in a non-production environment such as development, proof-of-concept, demo and so on. It is not for use in a production environment. To use z/OSMF in a production environment, see Configuring z/OSMF on page 39.

1. Introduction on page 43
2. Assumptions on page 43
3. Software Requirements on page 43
    a. Minimum Java level on page 43
    b. WebSphere® Liberty profile (z/OSMF V2R3 and later)
    c. System settings on page 44
    d. Web browser on page 44
4. Creating a z/OSMF nucleus on your system
    a. Running job IZUNUSEC to create security on page 45
    b. Running job IZUMKFS to create the z/OSMF user file system
    c. Copying the IBM procedures into JES PROCLIB on page 48
    d. Starting the z/OSMF server
    e. Accessing the z/OSMF Welcome page
    f. Mounting the z/OSMF user file system at IPL time
5. Adding the required REST services on page 52
    a. Enabling the z/OSMF JOB REST services
    b. Enabling the TSO REST services on page 53
    c. Enabling the z/OSMF data set and file REST services
    d. Enabling the z/OSMF Workflow REST services and Workflows task UI
6. Troubleshooting problems on page 56
    a. Common problems and scenarios on page 56
    b. Tools and techniques for troubleshooting on page 56

- Appendix A. Creating an IZUPRMxx parmlib member on page 56

- Appendix B. Modifying IZUSVR1 settings on page 58
- Appendix C. Adding more users to z/OSMF

Introduction

IBM® z/OS® Management Facility (z/OSMF) provides extensive system management functions in a task-oriented, web browser-based user interface with integrated user assistance, so that you can more easily manage the day-to-day operations and administration of your mainframe z/OS systems.

By following the steps in this guide, you can quickly enable z/OSMF on your z/OS system. This simplified approach to set-up, known as "z/OSMF Lite", requires only a minimal amount of z/OS customization, but provides the key functions that are required by many exploiters, such as the open mainframe project (Zowe™).

A z/OSMF Lite configuration is applicable to any future expansions you make to z/OSMF, such as adding more optional services and plug-ins.

It takes 2-3 hours to set up z/OSMF Lite. Some steps might require the assistance of your security administrator.

For detailed information about various aspects of z/OSMF configuration such as enabling the optional plug-ins and services, see the IBM publication *z/OSMF Configuration Guide*.

Assumptions

This document is intended for a first time z/OSMF setup. If z/OSMF is already configured on your system, you do not need to create a z/OSMF Lite configuration.

This document is designed for use with a single z/OS system, not a z/OS sysplex. If you plan to run z/OSMF in a sysplex, see *z/OSMF Configuration Guide* for multi-system considerations.

It is assumed that a basic level of security for z/OSMF is sufficient on the z/OS system. IBM provides a program, IZUNUSEC, to help you set up basic security for a z/OSMF Lite configuration.

System defaults are used for the z/OSMF environmental settings. Wherever possible, it is recommended that you use the default values. If necessary, however, you can override the defaults by supplying an IZUPRMxx member, as described in Appendix A. Creating an IZUPRMxx parmlib member on page 56.

It is recommended that you use the following procedures as provided by IBM:

- Started procedures IZUSVR1 and IZUANG1
- Logon procedure IZUFPROC

Information about installing these procedures is provided in Copying the IBM procedures into JES PROCLIB on page 48.

Software Requirements

Setting up z/OSMF Lite requires that you have access to a z/OS V2R2 system or later. Also, your z/OS system must meet the following minimum software requirements:

- Minimum Java level on page 43
- WebSphere® Liberty profile (z/OSMF V2R3 and later)
- System settings on page 44
- Web browser on page 44

Minimum Java level

Java™ must be installed and operational on your z/OS system, at the required minimum level. See the table that follows for the minimum level and default location. If you installed Java in another location, you must specify the JAVA_HOME statement in your IZUPRMxx parmlib member, as described in Appendix A. Creating an IZUPRMxx parmlib member on page 56.

z/OS Version	Minimum level of Java™	Recommended level of Java	Default location
z/OS V2R2	IBM® 64-bit SDK for z/OS®, Java Technology Edition V7.1 (SR3), with the PTFs for APAR PI71018 and APAR PI71019 applied **OR** IBM® 64-bit SDK for z/OS®, Java Technology Edition V8, with the PTF for APAR PI72601 applied.	IBM® 64-bit SDK for z/OS®, Java™ Technology Edition, V8 SR6 (5655-DGH)	`/usr/lpp/java/J7.1_64`
z/OS V2R3	IBM® 64-bit SDK for z/OS®, Java™ Technology Edition, V8 SR4 FP10 (5655-DGH)	IBM® 64-bit SDK for z/OS®, Java™ Technology Edition, V8 SR6 (5655-DGH)	`/usr/lpp/java/J8.0_64`

WebSphere® Liberty profile (z/OSMF V2R3 and later)

z/OSMF V2R3 uses the Liberty Profile that is supplied with z/OS, rather than its own copy of Liberty. The WebSphere Liberty profile must be mounted on your z/OS system. The default mount point is: `/usr/lpp/liberty_zos`. To determine whether WebSphere® Liberty profile is mounted, check for the existence of the mount point directory on your z/OS system.

If WebSphere® Liberty profile is mounted at a non-default location, you need to specify the location in the IZUSVR1 started procedure on the keyword **WLPDIR=**. For details, see Appendix B. Modifying IZUSVR1 settings on page 58.

Note: Whenever you apply PTFs for z/OSMF, you might be prompted to install outstanding WebSphere Liberty service. It is recommended that you do so to maintain z/OSMF functionality.

System settings

Ensure that the z/OS host system meets the following requirements:

- Port 443 (default port) is available for use.
- The system host name is unique and maps to the system on which z/OSMF Lite will be configured.

Otherwise, you might encounter errors later in the process. If you encounter errors, see Troubleshooting problems on page 56 for the corrective actions to take.

Web browser

For the best results with z/OSMF, use one of the following web browsers on your workstation:

- Microsoft Internet Explorer Version 11 or later
- Microsoft Edge (Windows 10)
- Mozilla Firefox ESR Version 52 or later.

To check your web browser's level, click **About** in the web browser.

Creating a z/OSMF nucleus on your system

The following system changes are described in this chapter:

- Running job IZUNUSEC to create security on page 45
- Running job IZUMKFS to create the z/OSMF user file system
- Copying the IBM procedures into JES PROCLIB on page 48
- Starting the z/OSMF server
- Accessing the z/OSMF Welcome page

- Mounting the z/OSMF user file system at IPL time

The following sample jobs that you might use are included in the package and available for download:

- IZUAUTH
- IZUICSEC
- IZUNUSEC_V2R2
- IZUNUSEC_V2R3
- IZUPRM00
- IZURFSEC
- IZUTSSEC
- IZUWFSEC

Download sample jobs

Check out the video for a demo of the process:

Running job IZUNUSEC to create security

The security job IZUNUSEC contains a minimal set of RACF® commands for creating security profiles for the z/OSMF nucleus. The profiles are used to protect the resources that are used by the z/OSMF server, and to grant users access to the z/OSMF core functions. IZUNUSEC is a simplified version of the sample job IZUSEC, which is intended for a more complete installation of z/OSMF.

Note: If your implementation uses an external security manager other than RACF (for example, CA Top Secret or CA ACF2), provide equivalent commands for your environment. For more information, see the following CA Technologies product documentation:

- Configure z/OS Management Facility for CA Top Secret
- Configure z/OS Management Facility for CA ACF2

Before you begin

In most cases, you can run the IZUNUSEC security job without modification. To verify that the job is okay to run as is, ask your security administrator to review the job and modify it as necessary for your security environment. If security is not a concern for the host system, you can run the job without modification.

Procedure

1. If you run z/OS V2R2 or V2R3, download job IZUNUSEC in the sample jobs package and upload this job to z/OS. If you run z/OS V2R4, locate job IZUNUSEC at SYS1.SAMPLIB.
2. Review and edit the job, if necessary.
3. Submit IZUNUSEC as a batch job on your z/OS system.
4. Connect your user ID to IZUADMIN group.
   a. Download job IZUAUTH in the sample jobs package and customize it.
   b. Replace the 'userid' with your z/OSMF user ID.
   c. Submit the job on your z/OS system.

Results

Ensure the IZUNUSEC job completes with return code `0000`.

To verify, check the results of the job execution in the job log. For example, you can use SDSF to examine the job log:

1. In the SDSF primary option menu, select Option ST.
2. On the SDSF Status Display, enter **S** next to the job that you submitted.
3. Check the return code of the job. The job succeeds if '0000' is returned.

Common errors

Review the following messages and the corresponding resolutions as needed:

Symptom	Cause	Resolution
Message IKJ56702I: INVALID data is issued	The job is submitted more than once.	You can ignore this message.
Job fails with an authorization error.	Your user ID lacks superuser authority.	Contact your security admin to run IZUNUSEC. If you are using RACF®, select a user ID with SPECIAL attribute which can issue all RACF® commands.
Job fails with an authorization error.	Your installation uses the RACF PROTECT-ALL option.	See Troubleshooting problems on page 56.
ADDGROUP and ADDUSER commands are not executed.	The automatic GID and UID assignment is required.	Define SHARED.IDS and BPX.NEXT.USER profiles to enable the use of AUTOUID and AUTOGID.

Running job IZUMKFS to create the z/OSMF user file system

The job IZUMKFS initializes the z/OSMF user file system, which contains configuration settings and persistence information for z/OSMF.

The job mounts the file system. On a z/OS V2R3 system with the PTF for APAR PI92211 installed, the job uses mount point `/global/zosmf`. Otherwise, for an earlier system, the job mounts the file system at mount point `/var/zosmf`.

Before you begin

To perform this step, you need a user ID with "superuser" authority on the z/OS host system. For more information about how to define a user with superuser authority, see the publication *z/OS UNIX System Services*.

Procedure

1. In the system library `SYS1.SAMPLIB`, locate job IZUMKFS.
2. Copy the job.
3. Review and edit the job:
   - Modify the job information so that the job can run on your system.
   - You must specify a volume serial (VOLSER) to be used for allocating a data set for the z/OSMF data directory.
4. Submit IZUMKFS as a batch job on your z/OS system.

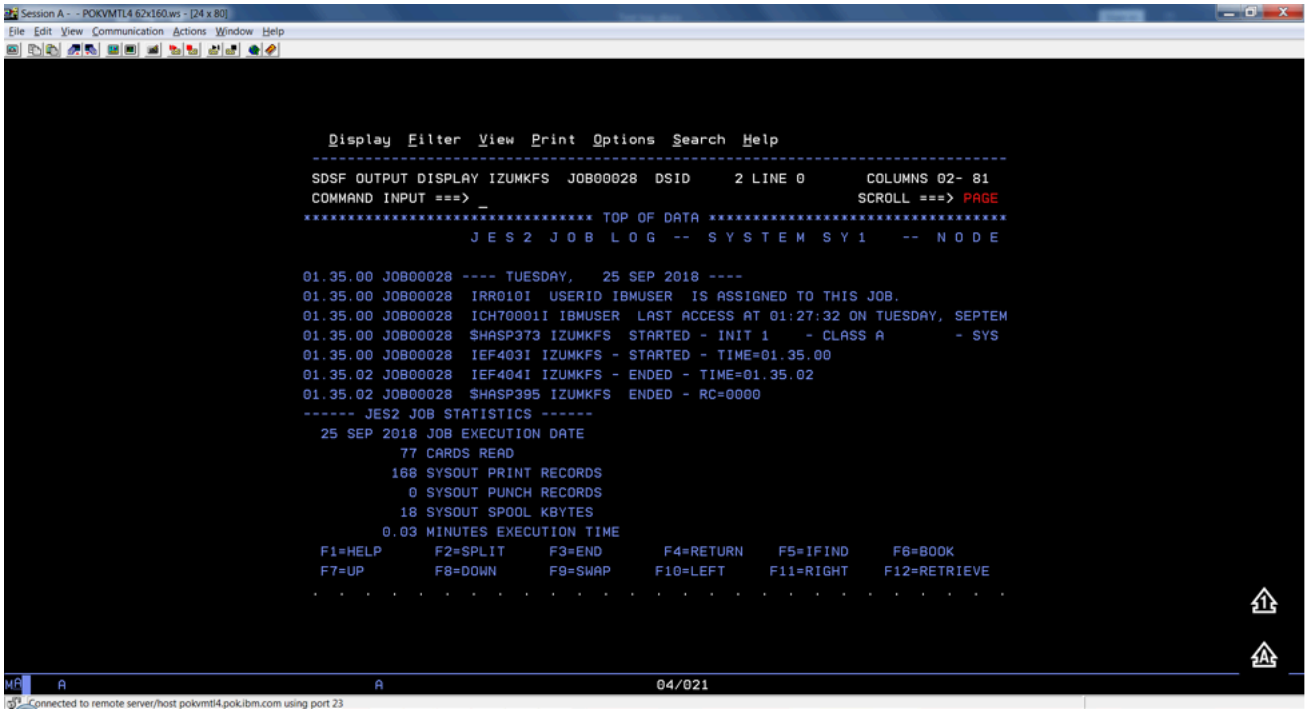
Results

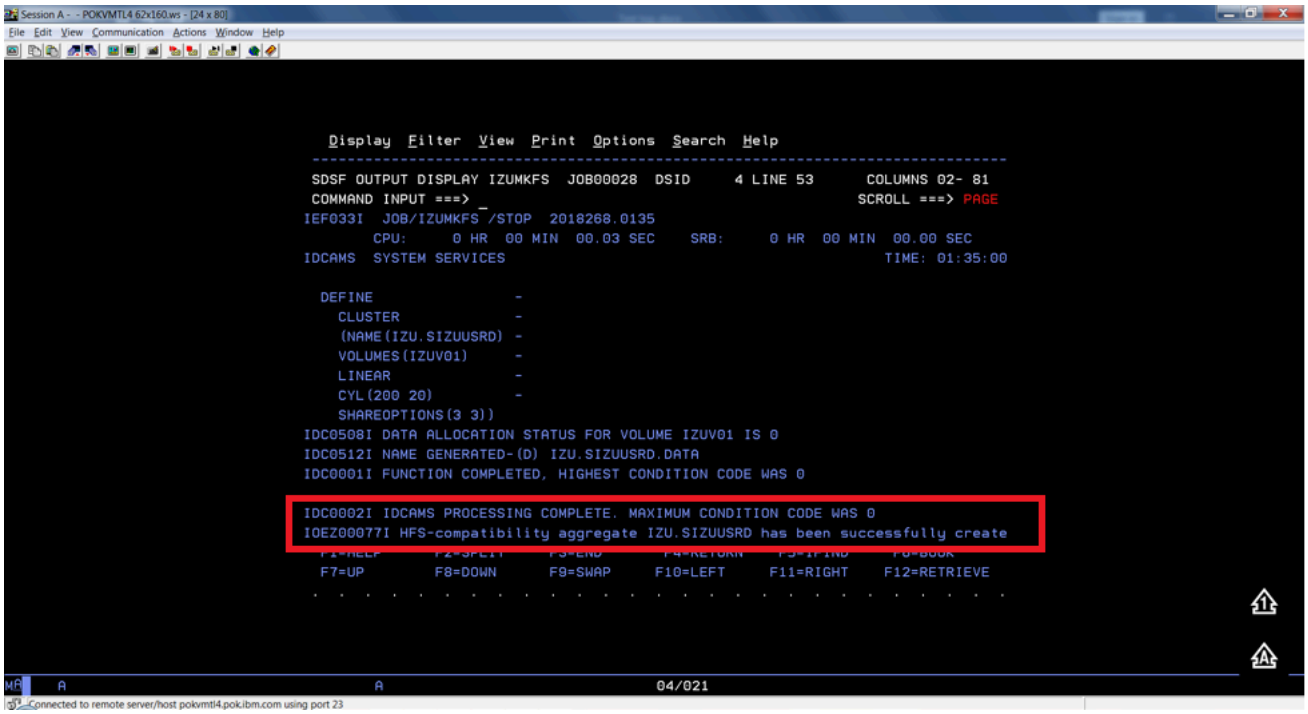
The z/OSMF file system is allocated, formatted, and mounted, and the necessary directories are created.

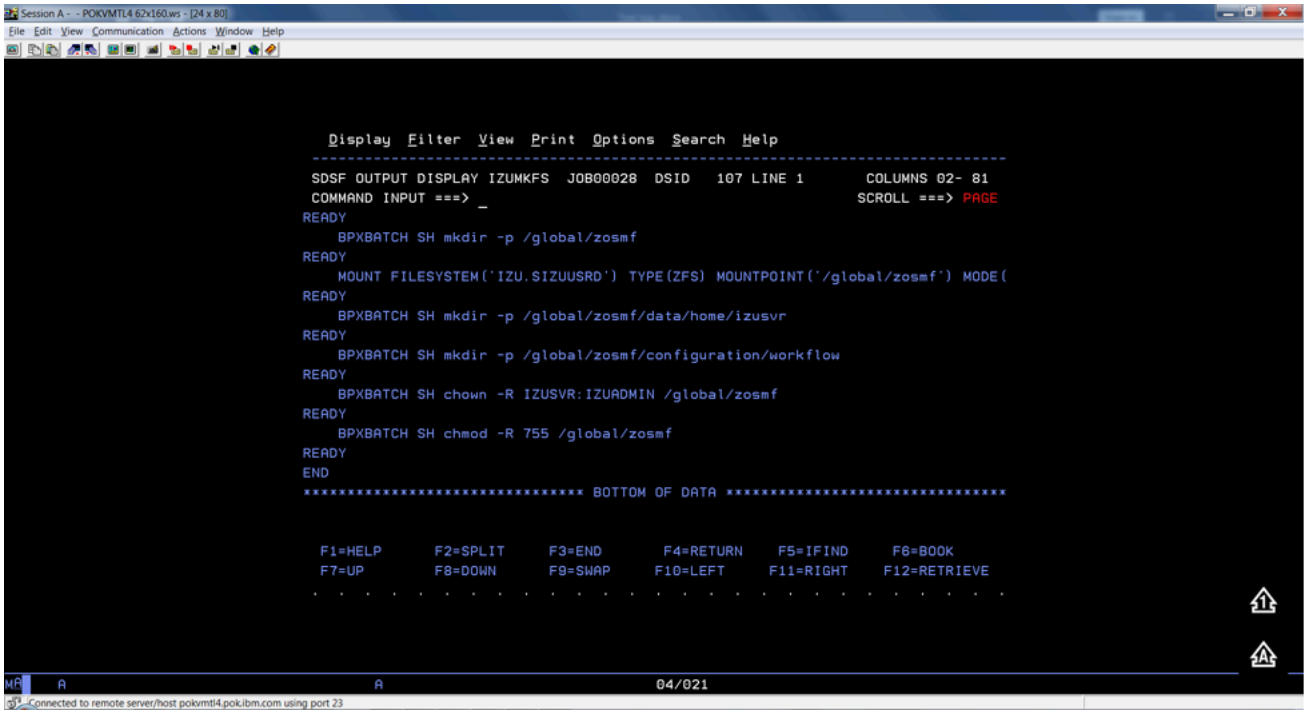
To verify if the file system is allocated, formatted, locate the following messages in IZUMKFS job output.

```
IDC0002I IDCAMS PROCESSING COMPLETE. MAX CONDITION CODE WAS 0.

IOEZ00077I HFS-compatibility aggregate izu.sizuusrd has been successfully
 created.
```

Sample output:


Session A - - POKVMTL4 62x160.ws - [24 x 80]
File Edit View Communication Actions Window Help
Display Filter View Print Options Search Help
SDSF OUTPUT DISPLAY IZUMKFS JOB00028 DSID 2 LINE 0 COLUMNS 02- 81
COMMAND INPUT ===> _ SCROLL ===> PAGE
******************************** TOP OF DATA ********************************
J E S 2 J O B L O G -- S Y S T E M S Y 1 -- N O D E
01.35.00 JOB00028 ---- TUESDAY, 25 SEP 2018 ----
01.35.00 JOB00028 IRR010I USERID IBMUSER IS ASSIGNED TO THIS JOB.
01.35.00 JOB00028 ICH70001I IBMUSER LAST ACCESS AT 01:27:32 ON TUESDAY, SEPTEM
01.35.00 JOB00028 $HASP373 IZUMKFS STARTED - INIT 1 - CLASS A - SYS
01.35.00 JOB00028 IEF403I IZUMKFS - STARTED - TIME=01.35.00
01.35.02 JOB00028 IEF404I IZUMKFS - ENDED - TIME=01.35.02
01.35.02 JOB00028 $HASP395 IZUMKFS ENDED - RC=0000
------ JES2 JOB STATISTICS ------
25 SEP 2018 JOB EXECUTION DATE
77 CARDS READ
168 SYSOUT PRINT RECORDS
0 SYSOUT PUNCH RECORDS
18 SYSOUT SPOOL KBYTES
0.03 MINUTES EXECUTION TIME
F1=HELP F2=SPLIT F3=END F4=RETURN F5=IFIND F6=BOOK
F7=UP F8=DOWN F9=SWAP F10=LEFT F11=RIGHT F12=RETRIEVE
04/021
Connected to remote server/host pokvmtl4.pok.ibm.com using port 23


Session A - - POKVMTL4 62x160.ws - [24 x 80]
File Edit View Communication Actions Window Help
Display Filter View Print Options Search Help
SDSF OUTPUT DISPLAY IZUMKFS JOB00028 DSID 4 LINE 53 COLUMNS 02- 81
COMMAND INPUT ===> _ SCROLL ===> PAGE
IEF033I JOB/IZUMKFS /STOP 2018268.0135
CPU: 0 HR 00 MIN 00.03 SEC SRB: 0 HR 00 MIN 00.00 SEC
IDCAMS SYSTEM SERVICES TIME: 01:35:00
DEFINE -
CLUSTER -
(NAME(IZU.SIZUUSRD) -
VOLUMES(IZUV01) -
LINEAR -
CYL(200 20) -
SHAREOPTIONS(3 3))
IDC0508I DATA ALLOCATION STATUS FOR VOLUME IZUV01 IS 0
IDC0512I NAME GENERATED-(D) IZU.SIZUUSRD.DATA
IDC0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0
IDC0002I IDCAMS PROCESSING COMPLETE. MAXIMUM CONDITION CODE WAS 0
IOEZ00077I HFS-compatibility aggregate IZU.SIZUUSRD has been successfully create
F7=UP F8=DOWN F9=SWAP F10=LEFT F11=RIGHT F12=RETRIEVE
04/021
Connected to remote server/host pokvmtl4.pok.ibm.com using port 23



Common errors

Review the following messages and the corresponding resolutions as needed

Symptom	Cause	Resolution
Job fails with FSM error.	Your user ID lacks superuser authority.	For more information about how to define a user with superuser authority, see the publication *z/OS UNIX System Services*.
Job fails with an authorization error.	Job statement errors.	See Troubleshooting problems on page 56.

Copying the IBM procedures into JES PROCLIB

Copy the z/OSMF started procedures and logon procedure from SYS1.PROCLIB into your JES concatenation. Use `$D PROCLIB` command to display your JES2 PROCLIB definitions.

Before you begin

Locate the IBM procedures. IBM supplies procedures for z/OSMF in your z/OS order:

- ServerPac and CustomPac orders: IBM supplies the z/OSMF procedures in the SMP/E managed proclib data set. In ServerPac and SystemPac, the default name for the data set is SYS1.IBM.PROCLIB.
- CBPDO orders: For a CBPDO order, the SMP/E-managed proclib data set is named as SYS1.PROCLIB.
- Application Development CD.

Procedure

Use ISPF option 3.3 or 3.4 to copy the procedures from SYS1.PROCLIB into your JES concatenation.

- IZUSVR1
- IZUANG1
- IZUFPROC

Results

The procedures now reside in your JES PROCLIB.

Common errors

Review the following messages and the corresponding resolutions as needed

Symptom	Cause	Resolution
Not authorized to copy into PROCLIB.	Your user ID doesn't have the permission to modify PROCLIB.	Contact your security administrator.
Abend code B37 or E37.	The data set runs out of space.	Use IEBCOPY utility to compress PROCLIB dataset before you copy it.

Starting the z/OSMF server

z/OSMF processing is managed through the z/OSMF server, which runs as the started tasks IZUANG1 and IZUSVR1. z/OSMF is started with the START command.

Before you begin

Ensure that you have access to the operations console and can enter the START command.

Procedure

In the operations console, enter the START commands sequentially:

```
S IZUANG1

S IZUSVR1
```

Note: The z/OSMF angel (IZUANG1) must be started before the z/OSMF server (IZUSVR1).

You must enter these commands manually at subsequent IPLs. If necessary, you can stop z/OSMF processing by entering the STOP command for each of the started tasks IZUANG1 and IZUSVR1.

Note: z/OSMF offers an autostart function, which you can configure to have the z/OSMF server started automatically. For more information about the autostart capability, see *z/OSMF Configuration Guide*.

Results

When the z/OSMF server is initialized, you can see the following messages displayed in the operations console:

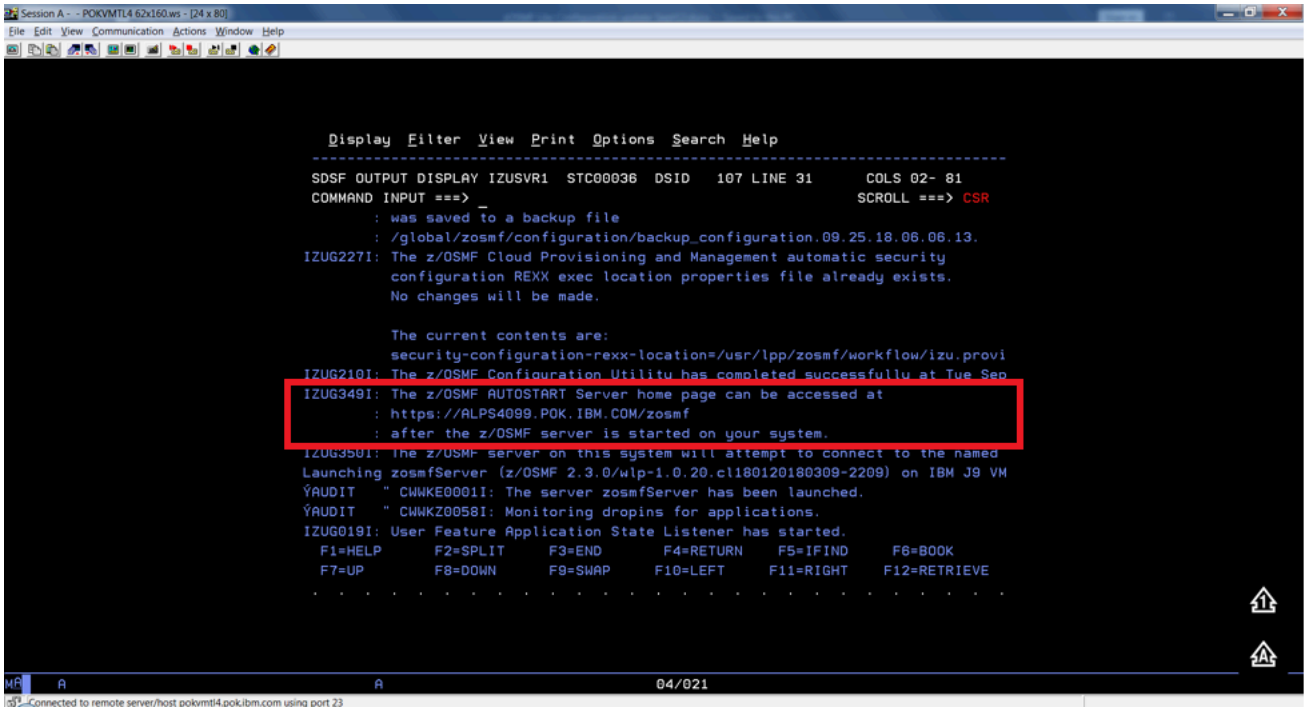
```
CWWKB0069I: INITIALIZATION IS COMPLETE FOR THE IZUANG1 ANGEL PROCESS.

IZUG400I: The z/OSMF Web application services are initialized.

CWWKF0011I: The server zosmfServer is ready to run a smarter planet.
```

Accessing the z/OSMF Welcome page

At the end of the z/OSMF configuration process, you can verify the results of your work by opening a web browser to the Welcome page.

Before you begin

To find the URL of the Welcome page, look for message IZUG349I in the z/OSMF server job log.



Procedure

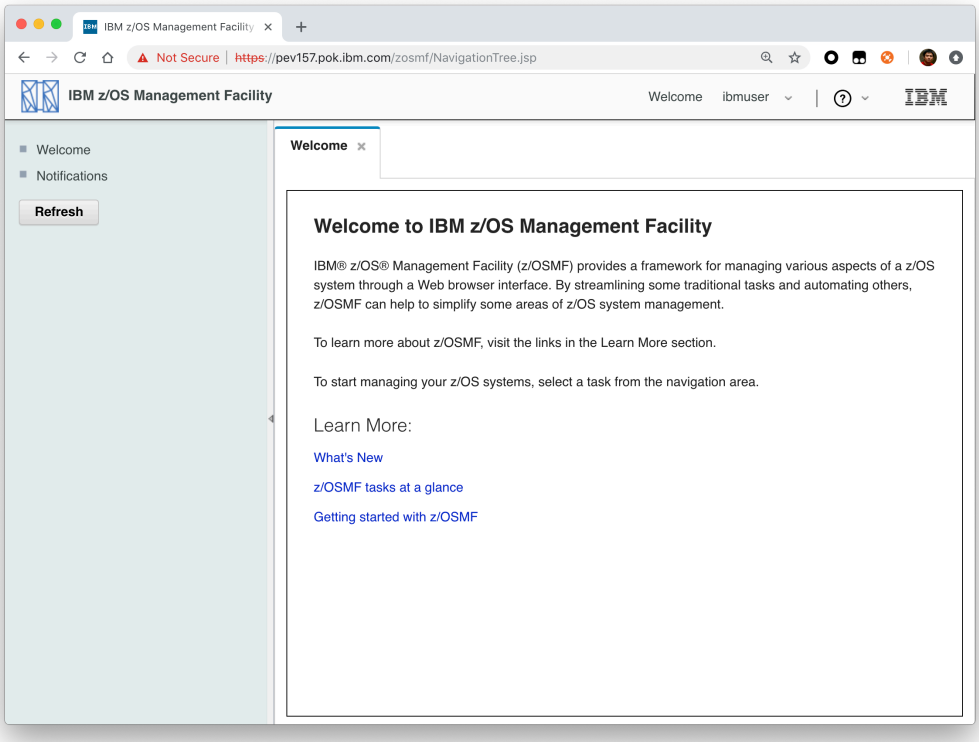
1. Open a web browser to the z/OSMF Welcome page. The URL for the Welcome page has the following format: https://hostname:port/zosmf/

   Where:

   - *hostname* is the host name or IP address of the system in which z/OSMF is installed.
   - *port* is the secure port for the z/OSMF configuration. If you specified a secure port for SSL encrypted traffic during the configuration process through parmlib statement HTTP_SSL_PORT, port is required to log in. Otherwise, it is assumed that you use the default port 443.
2. In the z/OS USER ID field on the Welcome page, enter the z/OS user ID that you use to configure z/OSMF.
3. In the z/OS PASSWORD field, enter the password or pass phrase that is associated with the z/OS user ID.
4. Select the style of UI for z/OSMF. To use the desktop interface, select this option. Otherwise, leave this option unselected to use the tree view UI.
5. Click **Log In**.

Results

If the user ID and password or pass phrase are valid, you are authenticated to z/OSMF. The Welcome page of IBM z/OS Management Facility tab opens in the main area. At the top right of the screen, Welcome <*your_user_ID*> is displayed. In the UI, only the options you are allowed to use are displayed.



You have successfully configured the z/OSMF nucleus.

Common errors

The following errors might occur during this step:

Symptom	Cause	Resolution
z/OSMF welcome page does not load in your web browser.	The SSL handshake was not successful. This problem can be related to the browser certificate.	See Certificate error in the Mozilla Firefox browser.
To log into z/OSMF, enter a valid z/OS user ID and password. Your account might be locked after too many incorrect log-in attempts.	The user ID is not connected to the IZUADMIN group.	Connect your user ID to the IZUADMIN group.
To log into z/OSMF, enter a valid z/OS user ID and password. Your account might be locked after too many incorrect log-in attempts.	The password is expired.	Log on to TSO using your z/OS User ID and password, you will be asked to change your password if it's expired.

Mounting the z/OSMF user file system at IPL time

Previously, in Running job IZUMKFS to create the z/OSMF user file system, you ran job IZUMKFS to create and mount the z/OSMF user file system. Now you should ensure that the z/OSMF user file system is mounted automatically for subsequent IPLs. To do so, update the BPXPRMxx parmlib member on your z/OS system.

Before you begin

By default, the z/OSMF file system uses the name IZU.SIZUUSRD, and is mounted in read/write mode. It is recommended that this file system is mounted automatically at IPL time.

If you do not know which BPXPRMxx member is active, follow these steps to find out:

1. In the operations console, enter the following command to see which parmlib members are included in the parmlib concatenation on your system:

   ```
   D PARMLIB
   ```
2. Make a note of the BPXPRMxx member suffixes that you see.
3. To determine which BPXPRMxx member takes precedence, enter the following command:

   ```
   D OMVS
   ```

The output of this command should be similar to the following:

```
BPXO042I 04.01.03 DISPLAY OMVS 391

OMVS 000F ACTIVE OMVS=(ST,3T)
```

In this example, the member BPXPRMST takes precedence. If BPXPRMST is not present in the concatenation, member BPXPRM3T is used.

Procedure

Add a MOUNT command for the z/OSMF user file system to your currently active BPXPRMxx parmlib member. For example:

On a z/OS V2R3 system with the PTF for APAR PI92211 installed:

```
MOUNT FILESYSTEM('IZU.SIZUUSRD') TYPE(ZFS) MODE(RDWR)

MOUNTPOINT('/global/zosmf') PARM('AGGRGROW') UNMOUNT
```

On a z/OS V2R2 or V2R3 system without PTF for APAR PI92211 installed:

```
MOUNT FILESYSTEM('IZU.SIZUUSRD') TYPE(ZFS) MODE(RDWR)

MOUNTPOINT('/var/zosmf') PARM('AGGRGROW') UNMOUNT
```

Results

The BPXPRMxx member is updated. At the next system IPL, the following message is issued to indicate that the z/OSMF file system is mounted automatically.

```
BPXF013I FILE SYSTEM IZU.SIZUUSRD WAS SUCCESSFULLY MOUNTED.
```

Adding the required REST services

You must enable a set of z/OSMF REST services for the Zowe framework.

The following system changes are described in this topic:

- Enabling the z/OSMF JOB REST services
- Enabling the TSO REST services on page 53
- Enabling the z/OSMF data set and file REST services
- Enabling the z/OSMF Workflow REST services and Workflows task UI

Enabling the z/OSMF JOB REST services

The Zowe framework requires that you enable the z/OSMF JOB REST services, as described in this topic.

Procedure

None

Results

To verify if the z/OSMF JOB REST services are enabled, open a web browser to our z/OS system (host name and port) and add the following REST call to the URL:

```
GET /zosmf/restjobs/jobs
```

The result is a list of the jobs that are owned by your user ID. For more information about the z/OSMF JOB REST services, see *z/OSMF Programming Guide*.

Common errors

Review the following messages and the corresponding resolutions as needed:

Symptom 1

401 Unauthorized

Cause

The user ID is not connected to IZUADMIN or IZUUSER.

Resolution

Connect your user ID to IZUADMIN or IZUUSER.

Symptom 2

HTTP/1.1 500 Internal Server Error {"rc":16,"reason":-1,"stack":"JesException: CATEGORY_CIM rc=16 reason=-1 cause=com.ibm.zoszmf.util.eis.EisConnectionException: IZUG911I: Connection to \"http://null:5988\" cannot be established, or was lost and cannot be re-established using protocol \"CIM\".....Caused by: WBEMException: CIM_ERR_FAILED (JNI Exception type CannotConnectException:\nCannot connect to local CIM server. Connection failed.)

Cause

For JES2, you may have performed one of the following "Modify" operations: Hold a job, Release a job, Change the job class, Cancel a job, Delete a job (Cancel a job and purge its output), or you are running JES3 without configuring CIM Server.

Resolution

If you are running JES2, you can use *synchronous support for job modify operations* which does not required CIM. If you are running JES3, follow the *CIM setup instructions* to configure CIM on your system.

Enabling the TSO REST services

The Zowe framework requires that you enable the TSO REST services, as described in this topic.

Before you begin

Ensure that the common event adapter component (CEA) of z/OS is running in full function mode.

1. To check if the CEA address space is active, enter the following command:

```
D A,CEA
```

1. If not, start CEA in full function mode. For detailed instructions, see *System prerequisites for the CEA TSO/E address space services*.
2. To verify that CEA is running in full function mode, enter the following command:

```
F CEA,D
```

The output should look like the following:

```
CEA0004I COMMON EVENT ADAPTER 399
STATUS: ACTIVE-FULL CLIENTS: 0 INTERNAL: 0
EVENTS BY TYPE: \#WTO: 0 \#ENF: 0 \#PGM: 0
TSOASMGR: ALLOWED: 50 IN USE: 0 HIGHCNT: 0
```

Procedure

1. If you run z/OS V2R2 and V2R3, download job IZUTSSEC in the sample jobs package and upload this Job to z/OS. If you run z/OS V2R4, locate job IZUTSSEC at `SYS1.SAMPLIB`.
2. Review and edit job IZUTSSEC before you submit. You can review the IZUTSSEC section below for more details.
3. Submit IZUTSSEC as a batch job on your z/OS system.

IZUTSSEC

IBM provides a set of jobs in `SYS1.SAMPLIB` with sample RACF commands to help with your z/OSMF configuration and its prerequisites. The IZUTSSEC job represents the authorizations that are needed for the z/OSMF TSO/E address space service. Your security administrator can edit and run the job. Generally, your z/OSMF user ID requires the same authorizations for using the TSO/E address space services as when you perform these operations through a TSO/E session on the z/OS system. For example, to start an application in a TSO/E address space requires that your user ID be authorized to operate that application. In addition, to use TSO/E address space services, you must have:

- READ access to the account resource in class ACCTNUM, where account is the value specified in the COMMON_TSO ACCT option in parmlib.
- READ access to the CEA.CEATSO.TSOREQUEST resource in class SERVAUTH.
- READ access to the proc resource in class TSOPROC, where proc is the value specified with the COMMON_TSO PROC option in parmlib.
- READ access to the <SAF_PREFIX>.*.izuUsers profile in the EJBROLE class. Or, at a minimum, READ access to the <SAF_PREFIX>.IzuManagementFacilityTsoServices.izuUsers resource name in the EJBROLE class. You must also ensure that the z/OSMF started task user ID, which is IZUSVR by default, has READ access to the CEA.CEATSO.TSOREQUEST resource in class SERVAUTH. To create a TSO/E address space on a remote system, you require the following authorizations:
- You must be authorized to the SAF resource profile that controls the ability to send data to the remote system (systemname), as indicated: CEA.CEATSO.FLOW.systemname
- To flow data between different systems in the sysplex, you must be authorized to do so by your external security manager, such as a RACF database with sysplex-wide scope. For example, to flow data between System A and System B, you must be permitted to the following resource profiles:
  - CEA.CEATSO.FLOW.SYSTEMA
  - CEA.CEATSO.FLOW.SYSTEMB

Results

The IZUTSSEC job should complete with return code 0000.

Enabling the z/OSMF data set and file REST services

The Zowe framework requires that you enable the z/OSMF data set and file REST services.

Before you begin

1. Ensure that the message queue size is set to a large enough value. It is recommended that you specify an IPCMSGQBYTES value of at least 20971520 (20M) in BPXPRMxx.

   Issue command `D OMVS,O` to see the current value of IPCMSGQBYTES, if it is not large enough, use the `SETOMVS` command to set a large value. To set this value dynamically, you can enter the following operator command:

   ```
   SETOMVS IPCMSGQBYTES=20971520
   ```
2. Ensure that the TSO REST services are enabled.
3. Ensure that IZUFPROC is in your JES concatenation.
4. Ensure that your user ID has a TSO segment defined. To do so, enter the following command from TSO/E command prompt:

   ```
   LU userid TSO
   ```

Where *userid* is your z/OS user ID.

The output from this command must include the section called **TSO information**, as shown in the following example:

```
TSO LU ZOSMFAD TSO NORACF

4:57:17 AM: USER=ZOSMFAD

TSO INFORMATION

---------------

ACCTNUM= 123412345

PROC= OMVSPROC

SIZE= 02096128

MAXSIZE= 00000000

USERDATA= 0000

 ***
```

Procedure

1. If you run z/OS V2R2 and V2R3, download job IZURFSEC in the sample jobs package and upload it to z/OS. If you run z/OS V2R4, locate job IZURFSEC at `SYS1.SAMPLIB`.
2. Copy the job.
3. Examine the contents of the job.
4. Modify the contents as needed so that the job will run on your system.
5. From the TSO/E command line, run the IZURFSEC job.

Results

Ensure that the IZURFSEC job completes with return code `0000`.

To verify if this setup is complete, try issuing a REST service. See the example in *List data sets* in the z/OSMF programming guide.

Common errors

Review the following messages and the corresponding resolutions as needed:

Symptom	Cause	Resolution
REST API doesn't return expected data with rc=12, rsn=3, message: message queue size "SIZE" is less than minimum: 20M	The message queue size for CEA is too small.	Ensure that the message queue size is set to a large enough value. It is recommended that you specify an IPCMSGQBYTES value of at least 20971520 (20M) in BPXPRMx.

Enabling the z/OSMF Workflow REST services and Workflows task UI

The Zowe framework requires that you enable the z/OSMF Workflow REST services and Workflows task UI.

Before you begin

1. Ensure that the JOB REST services are enabled.
2. Ensure that the TSO REST services are enabled.
3. Ensure that the dataset and file REST services are enabled.

Procedure

1. If you run z/OS V2R2 and V2R3, download job IZUWFSEC in the sample jobs package and upload this job to z/OS. If you run z/OS V2R4, locate job IZUWFSEC at `SYS1.SAMPLIB`.
2. Copy the job.
3. Examine the contents of the job.
4. Modify the contents as needed so that the job will run on your system.
5. From the TSO/E command line, run the IZUWFSEC job.

Results

Ensure the IZUWFSEC job completes with return code `0000`.

To verify, log on to z/OSMF (or refresh it) and verify that the Workflows task appears in the z/OSMF UI.

At this point, you have completed the setup of z/OSMF Lite.

Optionally, you can add more users to z/OSMF, as described in Appendix C. Adding more users to z/OSMF.

Troubleshooting problems

This section provides tips and techniques for troubleshooting problems you might encounter when creating a z/OSMF Lite configuration. For other types of problems that might occur, see *z/OSMF Configuration Guide*.

Common problems and scenarios

This section discusses troubleshooting topics, procedures, and tools for recovering from a set of known issues.

System setup requirements not met

This document assumes that the following is true of the z/OS host system:

- Port 443 is available for use. To check this, issue either TSO command `NETSTAT SOCKET` or TSO command `NETSTAT BYTE` to determine if the port is being used.
- The system host name is unique and maps to the system on which z/OSMF Lite is being installed. To retrieve this value, enter either "hostname" z/OS UNIX command or TSO command "HOMETEST". If your system uses another method of assigning the system name, such as a multi-home stack, dynamic VIPA, or System Director, see *z/OSMF Configuration Guide*.
- The global mount point exists. On a z/OS 2.3 system, the system includes this directory by default. On a z/OS 2.2 system, you must create the global directory at the following location: `/global/zosmf/`.

If you find that a different value is used on your z/OS system, you can edit the IZUPRMxx parmlib member to specify the correct setting. For details, see Appendix A. Creating an IZUPRMxx parmlib member on page 56.

Tools and techniques for troubleshooting

For information about working with z/OSMF log files, see *z/OSMF Configuration Guide*.

Common messages

```
ICH420I PROGRAM CELQLIB FROM LIBRARY CEE.SCEERUN2 CAUSED THE ENVIRONMENT
 TO BECOME UNCONTROLLED.

BPXP014I ENVIRONMENT MUST BE CONTROLLED FOR DAEMON (BPX.DAEMON)
PROCESSING.
```

If you see above error messages, check if your IZUANG0 procedure is up to date.

For descriptions of all the z/OSMF messages, see *z/OSMF messages* in IBM Knowledge Center.

Appendix A. Creating an IZUPRMxx parmlib member

If z/OSMF requires customization, you can modify the applicable settings by using the IZUPRMxx parmlib member. To see a sample member, locate the IZUPRM00 member in the SYS1.SAMPLIB data set. IZUPRM00 contains settings that match the z/OSMF defaults.

Using IZUPRM00 as a model, you can create a customized IZUPRMxx parmlib member for your environment and copy it to SYS1.PARMLIB to override the defaults.

The following IZUPRMxx settings are required for the z/OSMF nucleus:

- HOSTNAME
- HTTP_SSL_PORT
- JAVA_HOME.

The following setting is needed for the TSO/E REST services:

- COMMON_TSO ACCT(IZUACCT) REGION(50000) PROC(IZUFPROC)

Descriptions of these settings are provided in the table below. For complete details about the IZUPRMxx settings and the proper syntax for updating the member, see *z/OSMF Configuration Guide*.

If you change values in the IZUPRMxx member, you might need to customize the started procedure IZUSVR1, accordingly. For details, see Appendix B. Modifying IZUSVR1 settings on page 58.

To create an IZUPRMxx parmlib member, follow these steps:

1. Copy the sample parmlib member into the desired parmlib data set with the desired suffix.
2. Update the parmlib member as needed.
3. Specify the IZUPRMxx parmlib member or members that you want the system to use on the IZU parameter of IEASYSxx. Or, code a value for IZUPRM= in the IZUSVR1 started procedure. If you specify both IZU= in IEASYSxx and IZUPARM= in IZUSVR1, the system uses the IZUPRM= value you specify in the started procedure.

Setting	**Purpose**	**Rules**	**Default**
HOSTNAME(*hostname*)	Specifies the host name, as defined by DNS, where the z/OSMF server is located. To use the local host name, enter asterisk (*), which is equivalent to \@HOSTNAME from previous releases. If you plan to use z/OSMF in a multisystem sysplex, IBM recommends using a dynamic virtual IP address (DVIPA) that resolves to the correct IP address if the z/OSMF server is moved to a different system.	Must be a valid TCP/IP HOSTNAME or an asterisk (*).	Default: *

Setting	Purpose	Rules	Default
HTTP_SSL_PORT(nnn)	Identifies the port number that is associated with the z/OSMF server. This port is used for SSL encrypted traffic from your z/OSMF configuration. The default value, 443, follows the Internet Engineering Task Force (IETF) standard. **Note:** By default, the z/OSMF server uses the SSL protocol SSL_TLSv2 for secure TCP/IP communications. As a result, the server can accept incoming connections that use SSL V3.0 and the TLS 1.0, 1.1 and 1.2 protocols.	Must be a valid TCP/IP port number. Value range: 1 - 65535 (up to 5 digits)	Default: 443
COMMON_TSO ACCT(*account-number*) REGION(*region-size*) PROC(*proc-name*)	Specifies values for the TSO/E logon procedure that is used internally for various z/OSMF activities and by the Workflows task.	The valid ranges for each value are described in *z/OSMF Configuration Guide*.	Default: 443 ACCT(IZUACCT) REGION(50000) PROC(IZUFPROC)
USER_DIR=*filepath*	z/OSMF data directory path. By default, the z/OSMF data directory is located in `/global/zosmf`. If you want to use a different path for the z/OSMF data directory, specify that value here, for example: USER_DIR=`/the/new/config/dir`.	Must be a valid z/OS UNIX path name.	Default: `/global/zosmf/`

Appendix B. Modifying IZUSVR1 settings

You might need to customize the started procedure IZUSVR1 for z/OSMF Lite.

To modify the IZUSVR1 settings, follow these steps:

1. Make a copy
2. Apply your changes
3. Store your copy in PROCLIB.

Setting	Purpose	Rules	Default
WLPDIR='directory-path'	WebSphere Liberty server code path.	The directory path must: Be a valid z/OS UNIX path name Be a full or absolute path name Be enclosed in quotation marks Begin with a forward slash ('/').	Default: `/usr/lpp/zosmf/liberty`

Setting	Purpose	Rules	Default
USER_DIR=*filepath*	z/OSMF data directory path. By default, the z/OSMF data directory is located in /global/zosmf. If you want to use a different path for the z/OSMF data directory, specify that value here, for example: USER_DIR=`/the/new/config/dir`.	Must be a valid z/OS UNIX path name.	Default: `/global/zosmf/`

Appendix C. Adding more users to z/OSMF

Your security administrator can authorize more users to z/OSMF. Simply connect the required user IDs to the z/OSMF administrator group (IZUADMIN). This group is permitted to a default set of z/OSMF resources (tasks and services). For the specific group permissions, see Appendix A in *z/OSMF Configuration Guide*.

You can create more user groups as needed, for example, one group per z/OSMF task.

Before you Begin

Collect the z/OS user IDs that you want to add.

Procedure

1. On an RACF system, enter the CONNECT command for the user IDs to be granted authorization to z/OSMF resources:

```
CONNECT userid GROUP(IZUADMIN)
```

Results

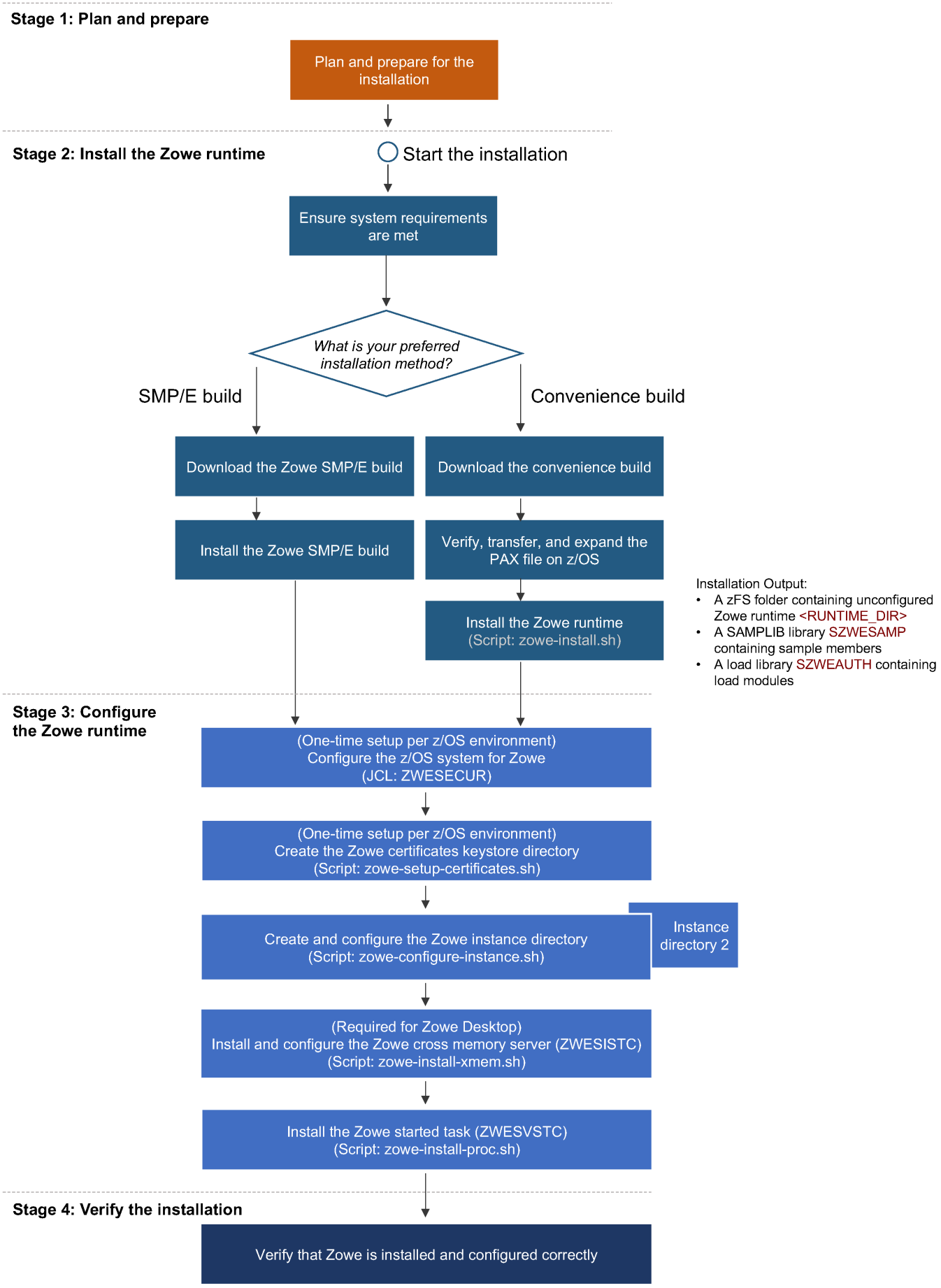
The user IDs can now access z/OSMF.

Installing Zowe z/OS components

Installation roadmap

To install Zowe™ on z/OS, there are two parts. The first part is the Zowe runtime that consists of three components: Zowe Application Framework, z/OS Explorer Services, and Zowe API Mediation Layer. The second part is the Zowe Cross Memory Server. This is an authorized server application that provides privileged services to Zowe in a secure manner.

Review the installation diagram and the introduction in this topic to see the general installation sequence and the most important tasks that are to be performed during installation and configuration. You can click each step on the diagram for detailed instructions.


Stage 1: Plan and prepare
Plan and prepare for the installation
Stage 2: Install the Zowe runtime
Start the installation
Ensure system requirements are met
What is your preferred installation method?
SMP/E build
Convenience build
Download the Zowe SMP/E build
Download the convenience build
Install the Zowe SMP/E build
Verify, transfer, and expand the PAX file on z/OS
Install the Zowe runtime (Script: zowe-install.sh)
Installation Output:
• A zFS folder containing unconfigured Zowe runtime <RUNTIME_DIR>
• A SAMPLIB library SZWESAMP containing sample members
• A load library SZWEAUTH containing load modules
Stage 3: Configure the Zowe runtime
(One-time setup per z/OS environment) Configure the z/OS system for Zowe (JCL: ZWESECUR)
(One-time setup per z/OS environment) Create the Zowe certificates keystore directory (Script: zowe-setup-certificates.sh)
Create and configure the Zowe instance directory (Script: zowe-configure-instance.sh)
Instance directory 2
(Required for Zowe Desktop) Install and configure the Zowe cross memory server (ZWESISTC) (Script: zowe-install-xmem.sh)
Install the Zowe started task (ZWESVSTC) (Script: zowe-install-proc.sh)
Stage 4: Verify the installation
Verify that Zowe is installed and configured correctly

Stage 1: Plan and prepare

Before you start the installation, review the information on hardware and software requirements and other considerations. See Introduction on page 36 for details.

Stage 2: Install the Zowe runtime

1. Ensure that the software requirements are met. The prerequisites are described in System requirements on page 36.
2. Choose the method of installing Zowe on z/OS.

   The Zowe z/OS binaries are distributed in the following formats. They contain the same contents but you install them by using different methods. You can choose which method to use depending on your needs.

   - **Convenience build**

     The Zowe z/OS binaries are packaged as a PAX file. You install this build by running shell script within a UNIX System Services (USS) shell. Convenience builds are full product installs.
   - **SMP/E build**

     The Zowe z/OS binaries are packaged as the following files that you can download. You install this build through SMP/E.

     - A pax.Z file, which contains an archive (compressed copy) of the FMIDs to be installed.
     - A readme file, which contains a sample job to decompress the pax.Z file, transform it into a format that SMP/E can process, and invoke SMP/E to extract and expand the compressed SMP/E input data sets.

   Note: The SMP/E build is currently in alpha, which means that it is available for early testing. You can provide any feedback about your experience with Zowe SMP/E as issues in the zowe-install-packaging GitHub repo.

   While the procedure to obtain and install the convenience build or SMP/E build are different, the procedure to configure a Zowe runtime are the same irrespective of how the build is obtained and installed.
3. Obtain and install the Zowe build.

   - For how to obtain the convenience build and install it, see Installing Zowe runtime from a convenience build on page 62.
   - For how to obtain the SMP/E build and install it, see Installing Zowe SMP/E Alpha on page 65.

After successful installation of either a convenience build or an SMP/E build, there will be a zFS folder that contains the unconfigured Zowe runtime `<RUNTIME_DIR>`, a SAMPLIB library `SZWESAMP` that contains sample members, and a load library `SZWEAUTH` that contains load modules. The steps to prepare the z/OS environment to launch Zowe are the same irrespective of the installation method.

Stage 3: Configure the Zowe runtime

1. Configure the z/OS security manager to prepare for launching the Zowe started tasks. For instructions, see Configuring the z/OS system for Zowe on page 83.

   A SAMPLIB JCL member `ZWESECUR` is provided to assist with the configuration. You can submit the `ZWESECUR` JCL member as-is or customize it depending on site preferences.

   If Zowe has already been launched on the z/OS system from a previous release of Version 1.8 or later, then you are applying a newer Zowe build. You can skip this security configuration step unless told otherwise in the release documentation.

2. Configure the Zowe certificates keystore and truststore directory. For instructions, see Configuring Zowe certificates on page 91.

   If you have already created a keystore directory from a previous release of Version 1.8 or later, then you may reuse the existing keystore directory.

   The Zowe keystore directory contains the key used by the Zowe desktop and the Zowe API mediation layer to secure its TLS communication with clients (such as web browsers or REST AI clients). The keystore directory also has a truststore where public keys of any servers that Zowe communicates to (such as z/OSMF) are held.

   A keystore directory needs to be created for a Zowe instance to be launched successfully, and a keystore directory can be shared between Zowe instances and between Zowe runtimes, including between different Zowe releases, unless specified otherwise in the release documentation.
3. Create and customize an instance directory that contains configuration data required to launch a Zowe runtime and is where log files are stored. For instructions, see Creating and configuring the Zowe instance directory on page 94.

   A single Zowe runtime can be launched multiple times from different instance directories, each specifying different port ranges, applications to include at start-up, paths of associated runtimes (Java, Node, z/OSMF).

   Next, you will install and configure the Zowe started tasks. Zowe has two high level started tasks: `ZWESVSTC` that launches the Zowe desktop and API mediation layer address spaces, and `ZWESISTC` that is a cross memory server that runs all of the APF-authorized code. The JCLs for the tasks are included in the PDS SAMPLIB `SZWESAMP` installed by Zowe and the load modules for the cross memory server are included in the PDS load library `SZWEAUTH`.
4. (Only required for launching the Zowe desktop) Configure the `ZWESISTC` cross memory server and install the load libraries. For instructions, see Installing and configuring the Zowe cross memory server (ZWESISTC) on page 97.

   The cross memory server is only required if you want to use the Zowe desktop. The cross memory server is not used by API Mediation Layer. If you want to use Zowe API Mediation Layer only, you can skip this step.

   Which components of Zowe are started is determined by the `LAUNCH_COMPONENT_GROUPS` value in the `instance.env` file in the Zowe instance directory, see Component groups on page 94.
5. Configure the `ZWESVSTC` started task. For instructions, see Installing the Zowe started task (ZWESVSTC) on page 100.

Stage 4: Verify the installation

Verify that Zowe is installed correctly on z/OS. See Verifying Zowe installation on z/OS on page 102.

Looking for troubleshooting help?

If you encounter unexpected behavior when installing or verifying the Zowe runtime on z/OS, see the Troubleshooting on page 282 section for tips.

Installing Zowe runtime from a convenience build

You install the Zowe™ convenience build by running shell script within a UNIX System Services (USS) shell.

Obtaining and preparing the convenience build

The Zowe installation file for Zowe z/OS components is distributed as a PAX file that contains the runtimes and the scripts to install and launch the z/OS runtime. For each release, there is a PAX file named `zowe-v.r.m.pax`, where

- `v` indicates the version
- `r` indicates the release number
- `m` indicates the modification number

The numbers are incremented each time a release is created so the higher the numbers, the later the release.

To download the PAX file, open your web browser and click the **Zowe z/OS Components** button on the Zowe Download website to save it to a folder on your desktop. After you download the PAX file, follow the instructions to verify the PAX file and prepare it to install the Zowe runtime.

Follow these steps:

1. Verify the integrity of the PAX file to ensure that the file you download is officially distributed by the Zowe project.

   Follow the instructions in the **Verify Hash and Signature of Zowe Binary** section on the post-download page `https://d1xozlojgf8voe.cloudfront.net/post_download.html?version=v.r.m` after you download the official build. For example, the post-download page for Version 1.4.0 is https://d1xozlojgf8voe.cloudfront.net/post_download.html?version=1.4.0.
2. Transfer the PAX file to z/OS.

   Follow these steps:

   a. Open a terminal in Mac OS/Linux, or command prompt in Windows OS, and navigate to the directory where you downloaded the Zowe PAX file.

   b. Connect to z/OS using SFTP. Issue the following command:

   ```
   sftp <userID@ip.of.zos.box>
   ```

   If SFTP is not available or if you prefer to use FTP, you can issue the following command instead:

   ```
   ftp <userID@ip.of.zos.box>
   ```

   Note: When you use FTP, switch to binary file transfer mode by issuing the following command:

   ```
   bin
   ```

   c. Navigate to the target directory that you want to transfer the Zowe PAX file into on z/OS.

   Note: After you connect to z/OS and enter your password, you enter the UNIX file system. The following commands are useful:

   - To see what directory you are in, type `pwd`.
   - To switch directory, type `cd`.
   - To list the contents of a directory, type `ls`.
   - To create a directory, type `mkdir`.

   d. When you are in the directory you want to transfer the Zowe PAX file into, issue the following command:

   ```
   put <zowe-v.r.m>.pax
   ```

   Where *zowe-v.r.m* is a variable that indicates the name of the PAX file you downloaded.

   Note: When your terminal is connected to z/OS through FTP or SFTP, you can prepend commands with `l` to have them issued against your desktop. To list the contents of a directory on your desktop, type `lls` where `ls` lists contents of a directory on z/OS.
3. When the PAX file is transferred, expand the PAX file by issuing the following command in an SSH session:

   ```
   pax -ppx -rf <zowe-v.r.m>.pax
   ```

   Where *zowe-v.r.m* is a variable that indicates the name of the PAX file you downloaded.

   This will expand to a file structure.

   ```
   /bin
   /files
   /install
   ```

```
   /scripts
   ...
```

Note: The PAX file will expand into the current directory. A good practice is to keep the installation directory apart from the directory that contains the PAX file. To do this, you can create a directory such as `/zowe/paxes` that contains the PAX files, and another such as `/zowe/builds`. Use SFTP to transfer the Zowe PAX file into the `/zowe/paxes` directory, use the `cd` command to switch into `/zowe/builds` and issue the command `pax -ppx -rf ../paxes/<zowe-v.r.m>.pax`. The `/install` folder will be created inside the `zowe/builds` directory from where the installation can be launched.

Installing the Zowe runtime

The first installation step is to create a USS folder that contains the Zowe runtime artifacts. This is known as the `<RUNTIME_DIR>`.

Step 1: Locate the install directory

Navigate to the directory where the installation archive is extracted. Locate the `/install` directory.

```
/install
  /zowe-install.sh
```

Step 2: Choose a runtime USS folder

For Zowe to execute, it must be installed into a runtime directory or `<RUNTIME_DIR>`. This directory will be created during the installation process and the user who performs the installation must have write permission for the installation to succeed.

If you are installing an upgrade of Zowe, the runtime directory used should be the existing `<RUNTIME_DIR>` of where the previous Zowe was installed. Upgrading Zowe is only supported for Version 1.8 or later.

For an enterprise installation of Zowe, a `<RUNTIME_DIR>` could be `/usr/lpp/zowe/v1`. For users who test Zowe for themselves, it could be `~/zowe/v1`.

Step 3: Choose a dataset HLQ for the SAMPLIB and LOADLIB

During installation, two PDS data sets are created: the `SZWESAMP` data set and the `SZWEAUTH` data set. These are not used at runtime and there is a further step needed to promote these to the z/OS execution environment but they contain required JCL and load modules.

You must know the `<DATA_SET_PREFIX>` into which to create the `SZWESAMP` and the `SZWEAUTH` PDS data sets. If a `<DATA_SET_PREFIX>` of `OPENSRC.ZWE` is specified, the PDS data sets `OPENSRC.ZWE.SZWESAMP` and `OPENSRC.ZWE.SZWEAUTH` will be created during installation.

The `SZWESAMP` data set is fixed block 90 samplib containing the following members.

Member name	Purpose
ZWESECUR	JCL member to configure z/OS user IDs and permissions required to run Zowe
ZWENOSEC	JCL member to undo the configuration steps performed in ZWESECUR and revert z/OS environment changes.
ZWESVSTC	JCL to start Zowe
ZWEXMSTC	JCL to start the Zowe cross memory server
ZWESIP00	Parmlib member for the cross memory server
ZWESASTC	Started task JCL for the cross memory Auxillary server
ZWESIPRG	Console commands to APF authorize the cross memory server load library

Member name	Purpose
ZWESISCH	PPT entries required by Cross memory server and its Auxiliary address spaces to run in Key(4)

The `SZWEAUTH` data set is a load library containing the following members.

Member name	Purpose
ZWESIS01	Load module for the cross memory server
ZWESAUX	Load module for the cross memory server's auxiliary address space

Step 4: Install the Zowe runtime

You install the Zowe runtime by executing the `zowe-install.sh` script passing in the arguments for the USS runtime directory and the prefix for the SAMPLIB and loadlib PDS members.

```
zowe-install.sh -i <RUNTIME_DIR> -h <DATASET_PREFIX>
```

In this documentation, the steps of creating the runtime directory and configuring the runtime directory are described separately. The configuration step is the same for a Zowe runtime whether it is installed from a convenience build or from an SMP/E distribution.

Next steps

For a z/OS system where you install Zowe 1.8 or later for the first time, follow the instructions in Stage 3: Configure the Zowe runtime on page 61 that describes how to Configuring the z/OS system for Zowe on page 83 and Configuring Zowe certificates on page 91.

If you have previously installed Zowe 1.8 or later, then you already have an instance directory that needs to be updated. If you have not installed Zowe 1.8 or later before, you will need to create an instance directory to be able to launch Zowe. For instructions, see Creating and configuring the Zowe instance directory on page 94.

Zowe has two started tasks that need to be installed and configured ready to be started. These are the Zowe server, see Installing the Zowe started task (ZWESVSTC) on page 100 and the Zowe cross memory server, see Installing and configuring the Zowe cross memory server (ZWESISTC) on page 97.

Installing Zowe SMP/E Alpha

Contents

- Introduction on page 66
  - Zowe description on page 67
  - Zowe FMIDs on page 67
- Program materials on page 67
  - Basic machine-readable material on page 67
  - Program publications on page 67
  - Program source materials on page 67
  - Publications useful during installation on page 67
- Program support on page 67
  - Statement of support procedures on page 67
- Program and service level information on page 68
  - Program level information on page 68
  - Service level information on page 68

- Installation requirements and considerations on page 68
  - Driving system requirements on page 68
    - Driving system machine requirements on page 68
    - Driving system programming requirements on page 68
  - Target system requirements on page 68
    - Target system machine requirements on page 69
    - Target system programming requirements on page 69
    - DASD storage requirements on page 69
  - FMIDs deleted on page 72
- Installation instructions on page 72
  - SMP/E considerations for installing Zowe
  - SMP/E options subentry values
  - Overview of the installation steps on page 73
  - Download the Zowe SMP/E package
  - Allocate file system to hold the download package on page 73
  - Upload the download package to the host on page 74
  - Extract and expand the compressed SMPMCS and RELFILEs on page 75
    - GIMUNZIP on page 77
  - Sample installation jobs on page 77
  - Create SMP/E environment (optional)
  - Perform SMP/E RECEIVE
  - Allocate SMP/E Target and Distributions Libraries
  - Allocate, create and mount ZSF files (Optional) on page 80
  - Allocate z/OS UNIX Paths
  - Create DDDEF entries on page 81
  - Perform SMP/E APPLY
  - Perform SMP/E ACCEPT
  - Run REPORT CROSSZONE on page 83
  - Cleaning up obsolete data sets, paths, and DDDEFs on page 83
- Activating Zowe on page 83
  - File system execution on page 83
- Zowe customization on page 83

Introduction

This program directory is intended for system programmers who are responsible for program installation and maintenance. It contains information about the material and procedures associated with the installation of Zowe Open Source Project (Base). This publication refers to Zowe Open Source Project (Base) as Zowe.

The Program Directory contains the following sections:

- **Program materials on page 67** identifies the basic program materials and documentation for Zowe.
- **Program support on page 67** describes the support available for Zowe.
- **Program and service level information on page 68** lists the APARs (program level) and PTFs (service level) that have been incorporated into Zowe.
- **Installation requirements and considerations on page 68** identifies the resources and considerations that are required for installing and using Zowe.
- **Installation instructions on page 72** provides detailed installation instructions for Zowe. It also describes the procedures for activating the functions of Zowe, or refers to appropriate publications.

Zowe description

Zowe™ is an open source project created to host technologies that benefit the Z platform. It is a sub-project of Open Mainframe Project which is part of the Linux Foundation. More information about Zowe is available at https://zowe.org.

Zowe FMIDs

Zowe consists of the following FMIDs:

- AZWE001

Program materials

Basic Machine-Readable Materials are materials that are supplied under the base license and are required for the use of the product.

Basic machine-readable material

The distribution medium for this program is via downloadable files. This program is in SMP/E RELFILE format and is installed using SMP/E. See Installation instructions on page 72 for more information about how to install the program.

Program publications

You can obtain the Zowe documentation from the Zowe doc site at https://docs.zowe.org/. No optional publications are provided for Zowe.

Program source materials

No program source materials or viewable program listings are provided for Zowe in the SMP/E installation package. However, program source materials can be downloaded from the Zowe GitHub repositories at https://github.com/zowe/.

Publications useful during installation

Publications listed below are helpful during the installation of Zowe.

Publication Title	Form Number
IBM SMP/E for z/OS User's Guide	SA23-2277
IBM SMP/E for z/OS Commands	SA23-2275
IBM SMP/E for z/OS Reference	SA23-2276
IBM SMP/E for z/OS Messages, Codes, and Diagnosis	GA32-0883

These and other publications can be obtained from https://www.ibm.com/shop/publications/order.

Program support

This section describes the support available for Zowe.

Because this is an alpha release of the Zowe FMID package for early testing and adoption, no formal support is offered. Support is available through the Zowe community. See Community Engagement for details. Slack is the preferred interaction channel.

Additional support may be available through other entities outside of the Open Mainframe Project and Linux Foundation which offers no warranty and provides the package under the terms of the EPL v2.0 license.

Statement of support procedures

Report any problems which you feel might be an error in the product materials to the Zowe community via the Zowe GitHub community repo at https://github.com/zowe/community/issues/new/choose. You may be asked to gather and submit additional diagnostics to assist the Zowe Community for analysis and resolution.

Program and service level information

This section identifies the program and relevant service levels of Zowe. The program level refers to the APAR fixes that have been incorporated into the program. The service level refers to the PTFs that have been incorporated into the program.

Program level information

All issues of previous releases of Zowe that were resolved before August 2019 have been incorporated into this packaging of Zowe.

Service level information

Since this is the first release of the SMP/E package, no PTFs have been created.

Installation requirements and considerations

The following sections identify the system requirements for installing and activating Zowe. The following terminology is used:

- *Driving System:* the system on which SMP/E is executed to install the program.
- *Target system:* the system on which the program is configured and run.

Use separate driving and target systems in the following situations:

- When you install a new level of a product that is already installed, the new level of the product will replace the old one. By installing the new level onto a separate target system, you can test the new level and keep the old one in production at the same time.
- When you install a product that shares libraries or load modules with other products, the installation can disrupt the other products. By installing the product onto a separate target system, you can assess these impacts without disrupting your production system.

Driving system requirements

This section describes the environment of the driving system required to install Zowe.

Driving system machine requirements

The driving system can be run in any hardware environment that supports the required software.

Driving system programming requirements

Program Number	Product Name	Minimum VRM	Minimum Service Level will satisfy these APARs	Included in the shipped product?
5650-ZOS	z/OS	V2.2.0 or later	N/A	No

Notes:

- SMP/E is a requirement for Installation and is an element of z/OS but can also be ordered as a separate product, 5655-G44, minimally V03.06.00.
- Installation might require migration to a new z/OS release to be service supported. See https://www-01.ibm.com/software/support/lifecycle/index_z.html.

Zowe is installed into a file system, either HFS or zFS. Before installing Zowe, you must ensure that the target system file system data sets are available for processing on the driving system. OMVS must be active on the driving system and the target system file data sets must be mounted on the driving system.

If you plan to install Zowe in a zFS file system, this requires that zFS be active on the driving system. Information on activating and using zFS can be found in z/OS Distributed File Service zSeries File System Administration (SC24-5989).

Target system requirements

This section describes the environment of the target system required to install and use Zowe.

Zowe installs in the z/OS (Z038) SREL.

Target system machine requirements

The target system can run in any hardware environment that supports the required software.

Target system programming requirements

Installation requisites

Installation requisites identify products that are required and must be present on the system or products that are not required but should be present on the system for the successful installation of Zowe.

Mandatory installation requisites identify products that are required on the system for the successful installation of Zowe. These products are specified as PREs or REQs.

Zowe has no mandatory installation requisites.

Conditional installation requisites identify products that are not required for successful installation of Zowe but can resolve such things as certain warning messages at installation time. These products are specified as IF REQs.

Zowe has no conditional installation requisites.

Operational requisites

Operational requisites are products that are required and must be present on the system, or, products that are not required but should be present on the system for Zowe to operate all or part of its functions.

Mandatory operational requisites identify products that are required for this product to operate its basic functions. The following table lists the target system mandatory operational requisites for Zowe.

Program Number	Product Name and Minimum VRM/Service Level
5650-ZOS	IBM z/OS Management Facility V2.2.0 or higher
5655-SDK	IBM SDK for Node.js - z/OS V8.16.0 or higher
5655-DGH	IBM 64-bit SDK for z/OS Java Technology Edition V8.0.0

Conditional operational requisites identify products that are not required for Zowe to operate its basic functions but are required at run time for Zowe to operate specific functions. These products are specified as IF REQs. Zowe has no conditional operational requisites.

Toleration/coexistence requisites

Toleration/coexistence requisites identify products that must be present on sharing systems. These systems can be other systems in a multi-system environment (not necessarily Parallel Sysplex™), a shared DASD environment (such as test and production), or systems that reuse the same DASD environment at different time intervals.

Zowe has no toleration/coexistence requisites.

Incompatibility (negative) requisites

Negative requisites identify products that must *not* be installed on the same system as Zowe.

Zowe has no negative requisites.

DASD storage requirements

Zowe libraries can reside on all supported DASD types.

Total DASD space required by Zowe

Library Type	Total Space Required in 3390 Trks	Description
Target	30 Tracks	/
Distribution	12030 Tracks	/

Library Type	Total Space Required in 3390 Trks	Description
File System(s)	9000 Tracks	/
Web Download	26111 Tracks	These are temporary data sets, which can be removed after the SMP/E install.

Notes:

1. For non-RECFM U data sets, we recommend using system-determined block sizes for efficient DASD utilization. For RECFM U data sets, we recommend using a block size of 32760, which is most efficient from the performance and DASD utilization perspective.
2. Abbreviations used for data set types are shown as follows.

   - **U** - Unique data set, allocated by this product and used by only this product. This table provides all the required information to determine the correct storage for this data set. You do not need to refer to other tables or program directories for the data set size.
   - **S** - Shared data set, allocated by this product and used by this product and other products. To determine the correct storage needed for this data set, add the storage size given in this table to those given in other tables (perhaps in other program directories). If the data set already exists, it must have enough free space to accommodate the storage size given in this table.
   - **E** - Existing shared data set, used by this product and other products. This data set is not allocated by this product. To determine the correct storage for this data set, add the storage size given in this table to those given in other tables (perhaps in other program directories). If the data set already exists, it must have enough free space to accommodate the storage size given in this table.

   If you currently have a previous release of Zowe installed in these libraries, the installation of this release will delete the old release and reclaim the space that was used by the old release and any service that had been installed. You can determine whether these libraries have enough space by deleting the old release with a dummy function, compressing the libraries, and comparing the space requirements with the free space in the libraries.

   For more information about the names and sizes of the required data sets, see Allocate SMP/E target and distribution libraries.
3. Abbreviations used for the file system path type are as follows.

   - **N** - New path, created by this product.
   - **X** - Path created by this product, but might already exist from a previous release.
   - **P** - Previously existing path, created by another product.
4. All target and distribution libraries listed have the following attributes:

   - The default name of the data set can be changed.
   - The default block size of the data set can be changed.
   - The data set can be merged with another data set that has equivalent characteristics.
   - The data set can be either a PDS or a PDSE, with some exceptions. If the value in the "ORG" column specifies "PDS", the data set must be a PDS. If the value in "DIR Blks" column specifies "N/A", the data set must be a PDSE.
5. All target libraries listed have the following attributes:

   - These data sets can be SMS-managed, but they are not required to be SMS-managed.
   - These data sets are not required to reside on the IPL volume.
   - The values in the "Member Type" column are not necessarily the actual SMP/E element types that are identified in the SMPMCS.

6. All target libraries that are listed and contain load modules have the following attributes:

- These data sets cannot be in the LPA, with some exceptions. If the value in the "Member Type" column specifies "LPA", it is advised to place the data set in the LPA.
- These data sets can be in the LNKLST.
- These data sets are not required to be APF-authorized, with some exceptions. If the value in the "Member Type" column specifies "APF", the data set must be APF-authorized.

Storage requirements for SMP/E work data sets

Library DDNAME	TYPE	ORG	RECFM	LRECL	No. of 3390 Trks	No. of DIR Blks
SMPWRK6	S	PDS	FB	80	(20,200)	50
SYSUT1	U	SEQ	--	--	(20,200)	0

In the table above, (20,200) specifies a primary allocation of 20 tracks, and a secondary allocation of 200 tracks.

Storage requirements for SMP/E data sets

Library DDNAME	TYPE	ORG	RECFM	LRECL	No. of 3390 Trks	No. of DIR Blks
SMPPTS	S	PDSE	FB	80	(12000,3000)	50

The following figures describe the target and distribution libraries and file system paths required to install Zowe. The storage requirements of Zowe must be added to the storage required by other programs that have data in the same library or path.

Note: Use the data in these tables to determine which libraries can be merged into common data sets. In addition, since some ALIAS names may not be unique, ensure that no naming conflicts will be introduced before merging libraries.

Storage requirements for Zowe target libraries

Note: These target libraries are not required for the initial alpha drop of Zowe SMP/E but will be required for subsequent drops so are included here for future reference.

Library DDNAME	Member Type	Target Volume	Type	Org	RECFM	LRECL	No. of 3390 Trks	No. of DIR Blks
SZWEAUTH	APF Load Modules	ANY	U	PDSE	U	0	15	N/A
SZWESAMP	Samples	ANY	U	PDSE	FB	80	15	5

Zowe file system paths

DDNAME	TYPE	Path Name
SZWEZFS	X	/usr/lpp/zowe/SMPE

Storage requirements for Zowe distribution libraries

Note: These target libraries are not required for the initial alpha drop of Zowe SMP/E but will be required for subsequent drops so are included here for future reference.

Library DDNAME	TYPE	ORG	RECFM	LRECL	No. of 3390 Trks	No. of DIR Blks
AZWEAUTH	U	PDSE	U	0	15	N/A

Library DDNAME	TYPE	ORG	RECFM	LRECL	No. of 3390 Trks	No. of DIR Blks
AZWESAMP	U	PDSE	FB	80	15	5
AZWEZFS	U	PDSE	VB	6995	12000	30

The following figures list data sets that are not used by Zowe, but are required as input for SMP/E.

Data Set Name	TYPE	ORG	RECFM	LRECL	No. of 3390 Trks	No. of DIR Blks
hlq.ZOWE.AZWE001.F1		PDSE	FB	80	5	N/A
hlq.ZOWE.AZWE001.F2		PDSE	FB	80	5	N/A
hlq.ZOWE.AZWE001.F4		PDSE	VB	6995	9000	N/A
hlq.ZOWE.AZWE001.SMPMCS		SEQ	FB	80	1	N/A
z/OS UNIX file system	U	zFS	N/A	N/A	17095	N/A

Note: These are temporary data sets, which can be removed after the SMP/E installation.

FMIDs deleted

Installing Zowe might result in the deletion of other FMIDs.

To see which FMIDs will be deleted, examine the ++VER statement in the SMPMCS of the product. If you do not want to delete these FMIDs now, install Zowe into separate SMP/E target and distribution zones.

Note: These FMIDs are not automatically deleted from the Global Zone. If you want to delete these FMIDs from the Global Zone, use the SMP/E REJECT NOFMID DELETEFMID command. See the SMP/E Commands book for details.

Special considerations

Zowe has no special considerations for the target system.

Installation instructions

This section describes the installation method and the step-by-step procedures to install and activate the functions of Zowe.

Notes:

- If you want to install Zowe into its own SMP/E environment, consult the SMP/E manuals for instructions on creating and initializing the SMPCSI and SMP/E control data sets.
- You can use the sample jobs that are provided to perform part or all of the installation tasks. The SMP/E jobs assume that all DDDEF entries that are required for SMP/E execution have been defined in appropriate zones.
- You can use the SMP/E dialogs instead of the sample jobs to accomplish the SMP/E installation steps.

SMP/E considerations for installing Zowe

Use the SMP/E RECEIVE, APPLY, and ACCEPT commands to install this release of Zowe.

SMP/E options subentry values

The recommended values for certain SMP/E CSI subentries are shown in the following table. Using values lower than the recommended values can result in failures in the installation. DSSPACE is a subentry in the GLOBAL options entry. PEMAX is a subentry of the GENERAL entry in the GLOBAL options entry. See the SMP/E manuals for instructions on updating the global zone.

Subentry	Value	Comment
DSSPACE	(1200,1200,1400)	Space allocation
PEMAX	SMP/E Default	IBM recommends using the SMP/E default for PEMAX.

Overview of the installation steps

Follow these high-level steps to download and install Zowe Open Source Project (Base).

1. Download the Zowe SMP/E package
2. Allocate file system to hold the download package on page 73
3. Upload the download package to the host on page 74
4. Extract and expand the compressed SMPMCS and RELFILEs on page 75
5. Sample installation jobs on page 77
6. Create SMP/E environment (optional)
7. Perform SMP/E RECEIVE
8. Allocate SMP/E target and distribution libraries
9. Allocate, create and mount ZSF files (Optional) on page 80
10. Allocate z/OS UNIX paths
11. Create DDDEF entries on page 81
12. Perform SMP/E APPLY
13. Perform SMP/E ACCEPT
14. Run REPORT CROSSZONE on page 83
15. Cleaning up obsolete data sets, paths, and DDDEFs on page 83

Download the Zowe SMP/E package

To download the Zowe SMP/E package, open your web browser and go to the Zowe Download website. Click the **Zowe SMP/E Alpha** button to save the files to a folder on your desktop.

You will receive two files on your desktop.

- **AZWE001.pax.Z (binary)**

  The SMP/E input data sets to install Zowe are provided as compressed files in AZWE001.pax.Z. This pax archive file holds the SMP/E MCS and RELFILEs.
- **AZWE001.readme.txt (text)**

  The README file AZWE001.readme.txt is a single JCL file containing a job with the job steps you need to begin the installation, including comprehensive comments on how to tailor them. There is a sample job step that executes the z/OS UNIX System Services pax command to extract package archives. This job also executes the GIMUNZIP program to expand the package archives so that the data sets can be processed by SMP/E.

  Review this file on your desktop and follow the instructions that apply to your system.

Allocate file system to hold the download package

You can either create a new z/OS UNIX file system (zFS) or create a new directory in an existing file system to place AZWE001.pax.Z. The directory that will contain the download package must reside on the z/OS system where the function will be installed.

To create a new file system, and directory, for the download package, you can use the following sample JCL (FILESYS).

Copy and paste the sample JCL into a separate data set, uncomment the job, and modify the job to update required parameters before submitting it.

```
//FILESYS  JOB <job parameters>
//*
//*************************************************************
```

```
//* This job must be updated to reflect your environment.
//* This sample:
//*   . Allocates a new z/OS UNIX file system
//*   . Creates a mount point directory
//*   . Mounts the file system
//*
//* - Provide valid job card information
//* - Change:
//*   @zfs_path@
//*   ----+----1----+----2----+----3----+----4----+----5
//*             - To the absolute z/OS UNIX path for the download
//*               package (starting with /)
//*             - Maximum length is 50 characters
//*             - Do not include a trailing /
//*   @zfs_dsn@
//*             - To your file system data set name
//*
//* Your userid MUST be defined as a SUPERUSER to successfully
//* run this job
//*
//*************************************************************
//*
//CREATE   EXEC PGM=IDCAMS,REGION=0M,COND=(0,LT)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  DEFINE CLUSTER ( -
         NAME(@zfs_dsn@) -
         TRK(#size) -
       /*VOLUME(volser)*/ -
         LINEAR -
         SHAREOPTIONS(3) -
         )
//*
//         SET ZFSDSN='@zfs_dsn@'
//FORMAT   EXEC PGM=IOEAGFMT,REGION=0M,COND=(0,LT),
//            PARM='-aggregate &ZFSDSN -compat'
//*STEPLIB  DD DISP=SHR,DSN=IOE.SIOELMOD        before z/OS 1.13
//*STEPLIB  DD DISP=SHR,DSN=SYS1.SIEALNKE       from z/OS 1.13
//SYSPRINT DD SYSOUT=*
//*
//MOUNT    EXEC PGM=IKJEFT01,REGION=0M,COND=(0,LT)
//SYSEXEC  DD DISP=SHR,DSN=SYS1.SBPXEXEC
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
  PROFILE MSGID WTPMSG
  oshell umask 0022; +
    mkdir -p @zfs_path@
  MOUNT +
    FILESYSTEM('@zfs_dsn@') +
    MOUNTPOINT('@zfs_path@') +
    MODE(RDWR) TYPE(ZFS) PARM('AGGRGROW')
//*
```

Expected Return Codes and Messages: You will receive a return code of 0 if this job runs correctly.

Upload the download package to the host

Upload the AZWE001.readme.txt file in text format and the AZWE001.pax.Z file in binary format from your workstation to the z/OS UNIX file system. The instructions in this section are also in the AZWE001.readme.txt file that you downloaded.

There are many ways to transfer the files or make them available to the z/OS system where the package will be installed. In the following sample dialog, we use FTP from a Microsoft Windows command line to do the transfer.

This assumes that the z/OS host is configured as an FTP host/server and that the workstation is an FTP client. Commands or other information entered by the user are in bold, and the following values are assumed.

User enters:	**Values**
mvsaddr	TCP/IP address or hostname of the z/OS system
tsouid	Your TSO user ID
tsopw	Your TSO password
d:	Location of the downloaded files
@zfs_path@	z/OS UNIX path where to store the files. This matches the @zfs_path@ variable you specified in the previous step.

Important! The AZWE001.pax.Z file must be uploaded to the z/OS driving system in binary format, or the subsequent UNPAX step will fail.

Sample FTP upload scenario:

*C:/>**ftp mvsaddr**Connected to mvsaddr.200-FTPD1 IBM FTP CS %version% at mvsaddr, %time% on %date %. 220 Connection will close if idle for more than 5 minutes.User (mvsaddr:(none)): **tsouid**331 Send password pleasePassword: **tsopw**230 tsouid is loaded on. Working directory is "tsouid.".ftp> **cd @zfs_path@**250 HFS directory @zfs_path@ is the current working directoryftp> **ascii**200 Representation type is Ascii NonPrintftp> **put c:/AZWE001.readme.txt**200 Port request OK.150 Storing data set @zfs_path@/AZWE001.readme.txt250 Transfer completed successfully.ftp: 0344 bytes sent in 0.01 sec. (1366.67 Kbs)ftp **binary**200 Representation type is Imageftp> **put c:\AZWE001.pax.Z**200 Port request OK.145 Storing data set @zfs_path@/AZWE001.pax.Z250 Transfer completed successfully.ftp: 524192256 bytes sent in 1.26 sec. (1040.52 Kbs)ftp: **quit**221 Quit command received. Goodbye.*

If you are unable to connect with ftp and only able to use sftp, the commands above are the same except that you will use *sftp* at the command prompt instead of *ftp*. Also, because *sftp* only supports binary file transfer, the ***ascii*** and ***binary*** commands should be omitted. After you transfer the AZWE001.readme.txt file, it will be in an ASCII codepage so you need to convert it to EBCDIC before it can be used. To convert AZWE001.readme.txt to EBCDIC, log in to the distribution system using ssh and run an ICONV command.

*C:>/**ssh tsouid@mvsaddr**tsouid@mvsaddr's password: **tsopw**/u/tsouid:>cd:@zfs_path@@zfs_path:>@zfs_path:>iconv -f ISO8859-1 -t IBM-1047 AZWE001.readme.txt > AZWE001.readme.EBCDIC@zfs_path:>rm AZWE001.readme.txt@zfs_path:>mv AZWE001.readme.EBCDIC AZWE001.readme.txt@zfs_path:>exitC:>/*

Extract and expand the compressed SMPMCS and RELFILEs

The AZWE001.readme.txt file uploaded in the previous step holds a sample JCL to expand the compressed SMPMCS and RELFILEs from the uploaded AZWE001.pax.Z file into data sets for use by the SMP/E RECEIVE job. The JCL is repeated here for your convenience.

- *@zfs_path@* matches the variable that you specified in the previous step.
- If the `oshell` command gets a RC=256 and message "pax: checksum error on tape (got ee2e, expected 0)", then the archive file was not uploaded to the host in binary format.
- GIMUNZIP allocates data sets to match the definitions of the original data sets. You might encounter errors if your SMS ACS routines alter the attributes used by GIMUNZIP. If this occurs, specify a non-SMS managed volume for the GINUMZIP allocation of the data sets. For example:

```
storclas-"storage_class" volume="data_set_volume"
newname-"..."/>
```

- Normally, your Automatic Class Selection (ACS) routines decide which volumes to use. Depending on your ACS configuration, and whether your system has constraints on disk space, units, or volumes, some supplied SMP/E jobs might fail due to volume allocation errors. See GIMUNZIP on page 77 for more details.

```
//EXTRACT  JOB <job parameters>
//*   - Change:

//*       @PREFIX@

//*       ----+----1----+----2----+

//*                   - To your desired data set name prefix

//*                   - Maximum length is 25 characters

//*                   - This value is used for the names of the

//*                     data sets extracted from the download-package

//*       @zfs_path@

//*       ----+----1----+----2----+----3----+----4----+----5

//*                   - To the absolute z/OS UNIX path for the download

//*                     package (starting with /)

//*                   - Maximum length is 50 characters

//*                   - Do not include a trailing /

//*
//UNPAX    EXEC PGM=IKJEFT01,REGION=0M,COND=(0,LT)
//SYSEXEC  DD DISP=SHR,DSN=SYS1.SBPXEXEC
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
  oshell cd @zfs_path@/ ; +
    pax -rvf AZWE001.pax.Z
//*
//GIMUNZIP EXEC PGM=GIMUNZIP,REGION=0M,COND=(0,LT)
//*STEPLIB  DD DISP=SHR,DSN=SYS1.MIGLIB
//SYSUT3   DD UNIT=SYSALLDA,SPACE=(CYL,(50,10))
//SYSUT4   DD UNIT=SYSALLDA,SPACE=(CYL,(25,5))
//SMPOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SMPDIR   DD PATHDISP=KEEP,
// PATH='@zfs_path@/'
//SYSIN    DD *
<GIMUNZIP>
<ARCHDEF archid="AZWE001.SMPMCS"
newname="@PREFIX@.ZOWE.AZWE001.SMPMCS"/>
<ARCHDEF archid="AZWE001.F1"
newname="@PREFIX@.ZOWE.AZWE001.F1"/>
<ARCHDEF archid="AZWE001.F2"
newname="@PREFIX@.ZOWE.AZWE001.F2"/>
<ARCHDEF archid="AZWE001.F4"
newname="@PREFIX@.ZOWE.AZWE001.F4"/>
</GIMUNZIP>
//*
```

GIMUNZIP

The GIMUNZIP job may issue allocation error messages for SYSUT1 similar to these:

```
IEF244I ZWE0GUNZ GIMUNZIP - UNABLE TO ALLOCATE 1 UNIT(S)  577
        AT LEAST 1 OFFLINE UNIT(S) NEEDED.
IEF877E ZWE0GUNZ NEEDS 1 UNIT(S)  578
FOR GIMUNZIP SYSUT1
FOR VOLUME SCRTCH-   1
OFFLINE
0AA4-0AA6 0AD0-0AD4
:
*07 IEF238D ZWE0GUNZ - REPLY DEVICE NAME OR 'CANCEL'.
 CNZ2605I At 10.10.22 the system will automatically  581
 reply: CANCEL
 to the following WTOR:
 0007 IEF238D ZWE0GUNZ - REPLY DEVICE NAME OR 'CANCEL'.
 R 0007,CANCEL
 IKJ56883I FILE SYSUT1 NOT ALLOCATED, REQUEST CANCELED
 -                                      --TIMINGS (MINS.)--
 -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    TCB    SRB  CLOCK
 -ZWE0GUNZ                      12   2311 ******    .00    2.4
 -ZWE0GUNZ ENDED.  NAME-                    TOTAL TCB CPU TIME=
 $HASP395 ZWE0GUNZ ENDED - RC=0012
```

The job will end with RC=12. If this happens, add a TEMPDS control statement to the existing SYSIN as shown below:

```
//SYSIN    DD *
<GIMUNZIP>
<TEMPDS volume="&VOLSER"> </TEMPDS>
<ARCHDEF archid="&FMID..SMPMCS"
newname="@PREFIX@.ZOWE.&FMID..SMPMCS"/>
<ARCHDEF archid="&FMID..F1"
newname="@PREFIX@.ZOWE.&FMID..F1"/>
<ARCHDEF archid="&FMID..F2"
newname="@PREFIX@.ZOWE.&FMID..F2"/>
<ARCHDEF archid="&FMID..F4"
newname="@PREFIX@.ZOWE.&FMID..F4"/>
</GIMUNZIP>
//*
```

where, `&VOLSER` is a DISK volume with sufficient free space to hold temporary copies of the RELFILES. As a guide, this may require 1,000 cylinders, or about 650 MB.

Sample installation jobs

The following sample installation jobs are provided in `hlq.ZOWE.AZWE001.F1`, or equivalent, as part of the project to help you install Zowe:

Job Name	Job Type	Description	RELFILE
ZWE1SMPE	SMP/E	Sample job to create an SMP/E environment (optional)	ZOWE.AZWE001.F1
ZWE2RCVE	RECEIVE	Sample SMP/E RECEIVE job	ZOWE.AZWE001.F1
ZWE3ALOC	ALLOCATE	Sample job to allocate target and distribution libraries	ZOWE.AZWE001.F1

Job Name	Job Type	Description	RELFILE
ZWE4ZFS	ALLOMZFS	Sample job to allocate, create mountpoint, and mount zFS data sets	ZOWE.AZWE001.F1
ZWE5MKD	MKDIR	Sample job to invoke the supplied ZWEMKDIR EXEC to allocate file system paths	ZOWE.AZWE001.F1
ZWE6DDEF	DDDEF	Sample job to define SMP/E DDDEFs	ZOWE.AZWE001.F1
ZWE7APLY	APPLY	Sample SMP/E ACCEPT job	ZOWE.AZWE001.F1
ZWE8ACPT	ACCEPT	Sample SMP/E ACCEPT job	ZOWE.AZWE001.F1

Note: When Zowe is downloaded from the web, the RELFILE data set name will be prefixed by your chosen high-level qualifier, as documented in the Extract and expand the compressed SMPMCS and RELFILEs on page 75 section.

You can access the sample installation jobs by performing an SMP/E RECEIVE (refer to Perform SMP/E RECEIVE), then copy the jobs from the RELFILES to a work data set for editing and submission.

You can also copy the sample installation jobs from the product files by submitting the following job. Before you submit the job, add a job statement and change the lowercase parameters to uppercase values to meet the requirements of your site.

```
//STEP1    EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//IN       DD DSN=ZOWE.AZWE001.F1,
//            DISP=SHR,
//*           VOL=SER=filevol,
//            UNIT=SYSALLDA
//OUT      DD DSNAME=jcl-library-name,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(TRK,(5,5,5)),
//*           VOL=SER=dasdvol,
//            UNIT=SYSALLDA
//SYSUT3   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSIN    DD *
    COPY INDD=IN,OUTDD=OUT
/*
```

See the following information to update the statements in the sample above:

- IN:
  - **filevol** is the volume serial of the DASD device where the downloaded files reside.
- OUT:
  - **jcl-library-name** is the name of the output data set where the sample jobs are stored.
  - **dasdvol** is the volume serial of the DASD device where the output data set resides. Uncomment the statement is a volume serial must be provided.

The following supplied jobs might fail due to disk space allocation errors, as mentioned above for GIMUNZIP on page 77. Review the following sections for example error and actions that you can take to resolve the error.

- ZWE2RCVE on page 79
- ZWE1SMPE and ZWE4ZFS on page 79

- ZWEMKDIR, ZWE1SMPE, ZWE2RCVE, ZWE3ALOC, ZWE4ZFS and ZWE5MKD

ZWE2RCVE

```
IEC032I E37-04,IGC0005E,ZWE2RCVE,RECEIVE,SMPTLIB,0AC0,USER10,
ZOWE.SMPE.AZWE001.F4
```

Add space and directory allocations to this SMPCNTL statement in the preceding ZWE1SMPE job:

```
ADD DDDEF(SMPTLIB)  UNIT(SYSALLDA) .
```

This makes it as below:

```
ADD DDDEF(SMPTLIB) CYL SPACE(2,1) DIR(10)  UNIT(SYSALLDA) .
```

ZWE1SMPE and ZWE4ZFS

Example error

```
IDC3506I REQUIRED VOLUMES AND/OR DEVICETYPES HAVE BEEN OMITTED
IDC3003I FUNCTION TERMINATED. CONDITION CODE IS 12

IDC0002I IDCAMS PROCESSING COMPLETE. MAXIMUM CONDITION CODE WAS 12
```

Uncomment the `VOLUMES(...)` control statements and refer to the comments at the start of the JCL job for related necessary changes.

ZWEMKDIR, ZWE1SMPE, ZWE2RCVE, ZWE3ALOC, ZWE4ZFS and ZWE5MKD

Example error

```
IEF257I ZWE3ALOC ALLOCD ALLOCD AZWEZFS - SPACE REQUESTED NOT AVAILABLE
IEF272I ZWE3ALOC ALLOCD ALLOCD - STEP WAS NOT EXECUTED.
```

Uncomment the `VOL=SER=&...` control statements and refer to the comments at the start of the JCL job for related necessary changes.

Create SMP/E environment (Optional)

If you choose to create a new SMP/E environment for this install, you can use one of the following two options.

- Create the SMP/E environment using JCL
- Create the SMP/E environment with z/OSMF workflow

Create SMP/E Environment using JCL

A sample job ZWE1SMPE is provided or you may choose to use your own JCL. If you are using an existing CSI, do not run the sample job ZWE1SMPE. If you choose to use the sample job provided, edit and submit ZWE1SMPE. Consult the instructions in the sample job for more information.

Note: If you want to use the default of letting your Automatic Class Selection (ACS) routines decide which volume to use, comment out the following line in the sample job ZWE1SMPE.

```
// SET CSIVOL=#csivol
```

Expected Return Codes and Messages: You will receive a return code of 0 if this job runs correctly.

Create SMP/E Environment with z/OSMF Workflow

z/OSMF workflow simplifies the procedure to create an SMP/E environment for Zowe. Register and execute the Zowe SMP/E workflow to create SMP/E environment in the z/OSMF web interface. Perform the following steps to register and execute the Zowe workflow in the z/OSMF web interface:

1. Log in to the z/OSMF web interface.

2. Select **Workflows** from the navigation tree.
3. Select **Create Workflow** from the **Actions** menu.
4. Enter the complete path to the workflow definition file in the **Workflow Definition filed**.

   The workflow is located in the ZWEWRF01 member of the `hlq.ZOWE.AZWE001.F4` data set.
5. (Optional) Enter the path to the customized variable input file that you prepared in advance.

   The variable input file is located in ZWEYML01 member of the `hlq.ZOWE.AZWE001` data set.

   Create a copy of the variable input file. Modify the file as necessary according to the built-in comments. Set the field to the path where the new file is located. When you execute the workflow, the values from the variable input file override the workflow variables default values.
6. Select the system where you want to execute the workflow.
7. Select **Next**.
8. Specify the unique workflow name.
9. Select or enter an **Owner Use ID** and select **Assign all steps to owner user ID**.
10. Select **Finish**.

    The workflow is registered in z/OSMF and ready to execute.
11. Select the workflow that you registered from the workflow list.
12. Execute the steps in order.

    For general information about how to execute z/OSMF workflow steps, watch the z/OSMF Workflows Tutorial.
13. Perform the following steps to execute each step individually:

    a. Double-click the title of the step.
    b. Select the **Perform** tab.
    c. Review the step contents and update the input values as required.
    d. Select **Next**.
    e. Repeat the previous two steps to complete all items until the option **Finish** is available.
    f. Select **Finish**.

       After you execute each step, the step is marked as **Complete**. The workflow is executed.

After you complete executing all the steps individually, the Zowe SMP/E is created.

Perform SMP/E RECEIVE

Edit and submit sample job ZWE2RCVE to perform the SMP/E RECEIVE for Zowe. Consult the instructions in the sample job for more information.

Expected Return Codes and Messages: You will receive a return code of 0 if this job runs correctly.

Allocate SMP/E target and distributions libraries

Edit and submit sample job ZWE3ALOC to allocate the SMP/E target and distribution libraries for Zowe. Consult the instructions in the sample job for more information.

Expected Return Codes and Messages: You will receive a return code of 0 if this job runs correctly.

Allocate, create and mount ZSF files (Optional)

This job allocates, creates a mountpoint, and mounts zFS data sets.

If you plan to install Zowe into a new z/OS UNIX file system, you can edit and submit the optional ZWE4ZFS job to perform the following tasks. Consult the instructions in the sample job for more information.

- Create the z/OS UNIX file system
- Create a mountpoint
- Mount the z/OS UNIX file system on the mountpoint

The recommended z/OS UNIX file system type is zFS. The recommended mountpoint is */usr/lpp/zowe.*

Before running the sample job to create the z/OS UNIX file system, you must ensure that OMVS is active on the driving system. zFS must be active on the driving system if you are installing Zowe into a file system that is zFS.

If you create a new file system for this product, consider updating the BPXPRMxx PARMLIB member to mount the new file system at IPL time. This action can be helpful if an IPL occurs before the installation is completed.

```
MOUNT FILESYSTEM('#dsn')
 MOUNTPOINT('/usr/lpp/zowe')
 MODE(RDWR)        /* can be MODE(READ) */
 TYPE(ZFS) PARM('AGGRGROW') /* zFS, with extents */
```

See the following information to update the statements in the previous sample:

- **#dsn** is the name of the data set holding the z/OS UNIX file system.
- ***/usr/lpp/zowe*** is the name of the mountpoint where the z/OS UNIX file system will be mounted.

Expected Return Codes and Messages: You will receive a return code of 0 if this job runs correctly.

Allocate z/OS UNIX paths

The target system HFS or zFS data set must be mounted on the driving system when running the sample ZWE5MKD job since the job will create paths in the HFS or zFS.

Before running the sample job to create the paths in the file system, you must ensure that OMVS is active on the driving system and that the target system's HFS or zFS file system is mounted on the driving system. zFS must be active on the driving system if you are installing Zowe into a file system that is zFS.

If you plan to install Zowe into a new HFS or zFS file system, you must create the mountpoint and mount the new file system on the driving system for Zowe.

The recommended mountpoint is */usr/lpp/zowe*.

Edit and submit sample job ZWE5MKD to allocate the HFS or zFS paths for Zowe. Consult the instructions in the sample job for more information.

If you create a new file system for this product, consider updating the BPXPRMxx PARMLIB member to mount the new file system at IPL time. This action can be helpful if an IPL occurs before the installation is completed.

Expected Return Codes and Messages: You will receive a return code of 0 if this job runs correctly.

Create DDDEF entries

Edit and submit sample job ZWE6DDEF to create DDDEF entries for the SMP/E target and distribution libraries for Zowe. Consult the instructions in the sample job for more information.

Expected Return Codes and Messages: You will receive a return code of 0 if this job runs correctly.

Perform SMP/E APPLY

In this step, you run the sample job ZWE7APLY to apply Zowe. This step can take a long time to run, depending on the capacity of your system, and on what other jobs are running.

Follow these steps

1. Ensure that you have the latest HOLDDATA; then edit and submit sample job ZWE7APLY to perform an SMP/E APPLY CHECK for Zowe. Consult the instructions in the sample job for more information.

   The latest HOLDDATA is available through several different portals, including http://service.software.ibm.com/holdata/390holddata.html. The latest HOLDDATA may identify HIPER and FIXCAT APARs for the FMIDs you will be installing. An APPLY CHECK will help you determine whether any HIPER or FIXCAT APARs are applicable to the FMIDs you are installing. If there are any applicable HIPER of FIXCAT APARs, the APPLY CHECK will also identify fixing PTFs that will resolve the APARs, if a fixing PTF is available.

   You should install the FMIDs regardless of the status of unresolved HIPER or FIXCAT APARs. However, do not deploy the software until the unresolved HIPER and FIXCAT APARs have been analyzed to determine their applicability. That is, before deploying the software either ensure fixing PTFs are applied to resolve all HIPER or

FIXCAT APARs, or ensure the problems reported by all HIPER or FIXCAT APARs are not applicable to your environment.

To receive the full benefit of the SMP/E Causer SYSMOD Summary Report, do *not* bypass the PRE, ID, REQ, and IFREQ on the APPLY CHECK. The SMP/E root cause analysis identifies the cause only of *errors* and not of *warnings* (SMP/E treats bypassed PRE, ID, REQ, and IFREQ conditions as warnings, instead of errors).

Here are sample APPLY commands:

a. To ensure that all recommended and critical service is installed with the FMIDs, receive the latest HOLDDATA and use the APPLY CHECK command as follows

```
APPLY S(fmid,fmid,...) CHECK
FORFMID(fmid,fmid,...)
SOURCEID(RSU*)
FIXCAT(IBM.PRODUCTINSTALL-REQUIREDSERVICE)
GROUPEXTEND .
```

- Some HIPER APARs might not have fixing PTFs available yet. You should analyze the symptom flags for the unresolved HIPER APARs to determine if the reported problem is applicable to your environment and if you should bypass the specific ERROR HOLDs in order to continue the installation of the FMIDs.
- This method requires more initial research, but can provide resolution for all HPERs that have fixing PTFs available and not in a PE chain. Unresolved PEs or HIPERs might still exist and require the use of BYPASS.

a. To install the FMIDs without regard for unresolved HIPER APARs, you can add the BYPASS(HOLDCLASS(HIPER)) operand to the APPLY CHECK command. This will allow you to install FMIDs, even though one of more unresolved HIPER APARs exist. After the FMIDs are installed, use the SMP/E REPORT ERRSYSMODS command to identify unresolved HIPER APARs and any fixing PTFs.

```
APPLY S(fmid,fmid,...) CHECK
FORFMID(fmid,fmid,...)
SOURCEID(RSU*)
FIXCAT(IBM.PRODUCTINSTALL-REQUIREDSERVICE)
GROUPEXTEND
BYPASS(HOLDCLASS(HIPER)) .
 ..any other parameters documented in the program directory
```

- This method is quicker, but requires subsequent review of the Exception SYSMOD report produced by the REPORT ERRSYSMODS command to investigate any unresolved HIPERs. If you have received the latest HOLDDATA, you can also choose to use the REPORT MISSINGFIX command and specify Fix Category IBM.PRODUCTINSTALL-REQUIREDSERVICE to investigate missing recommended service.
- If you bypass HOLDs during the installation of the FMIDs because fixing PTFs are not yet available, you can be notified when the fixing PTFs are available by using the APAR Status Tracking (AST) function of the ServiceLink or the APAR Tracking function of Resource Link.

2. After you take actions that are indicated by the APPLY CHECK, remove the CHECK operand and run the job again to perform the APPLY.

Note: The GROUPEXTENDED operand indicates the SMP/E applies all requisite SYSMODs. The requisite SYSMODS might be applicable to other functions.

Expected Return Codes and Messages from APPLY CHECK: You will receive a return code of 0 if the job runs correctly.

Expected Return Codes and Messages from APPLY: You will receive a return code of 0 if the job runs correctly.

Perform SMP/E ACCEPT

Edit and submit sample job ZWE8ACPT to perform an SMP/E ACCEPT CHECK for Zowe. Consult the instructions in the sample job for more information.

To receive the full benefit of the SMP/E Causer SYSMOD Summary Report, do not bypass the PRE, ID, REQ, and IFREQ on the ACCEPT CHECK. The SMP/E root cause analysis identifies the cause of errors but not warnings (SMP/E treats bypassed PRE, ID, REQ, and IFREQ conditions as warnings rather than errors).

Before you use SMP/E to load new distribution libraries, it is recommended that you set the ACCJCLIN indicator in the distribution zone. In this way, you can save the entries that are produced from JCLIN in the distribution zone whenever a SYSMOD that contains inline JCLIN is accepted. For more information about the ACCJCLIN indicator, see the description of inline JCLIN in the SMP/E Commands book for details.

After you take actions that are indicated by the ACCEPT CHECK, remove the CHECK operand and run the job again to perform the ACCEPT.

Note: The GROUPEXTEND operand indicates that SMP/E accepts all requisite SYSMODs. The requisite SYSMODS might be applicable to other functions.

Expected Return Codes and Messages from ACCEPT CHECK: You will receive a return code of 0 if this job runs correctly.

If PTFs that contain replacement modules are accepted, SMP/E ACCEPT processing will link-edit or bind the modules into the distribution libraries. During this processing, the Linkage Editor or Binder might issue messages that indicate unresolved external references, which will result in a return code of 4 during the ACCEPT phase. You can ignore these messages, because the distribution libraries are not executable and the unresolved external references do not affect the executable system libraries.

Expected Return Codes and Messages from ACCEPT: You will receive a return code of 0 if this job runs correctly.

Run REPORT CROSSZONE

The SMP/E REPORT CROSSZONE command identifies requisites for products that are installed in separate zones. This command also creates APPLY and ACCEPT commands in the SMPPUNCH data set. You can use the APPLY and ACCEPT commands to install those cross-zone requisites that the SMP/E REPORT CROSSZONE command identifies.

After you install Zowe, it is recommended that you run REPORT CROSSZONE against the new or updated target and distribution zones. REPORT CROSSZONE requires a global zone with ZONEINDEX entries that describe all the target and distribution libraries to be reported on.

For more information about REPORT CROSSZONE, see the SMP/E manuals.

Cleaning up obsolete data sets, paths, and DDDEFs

The web download data sets listed in DASD storage requirements on page 69 are temporary data sets. You can delete these data sets after you complete the SMP/E installation.

Activating Zowe
File system execution

If you mount the file system in which you have installed Zowe in read-only mode during execution, then you do not have to take further actions to activate Zowe.

Zowe customization

You can find the necessary information about customizing and using Zowe on the Zowe doc site.

- For more information about how to customize Zowe, see Zowe Application Framework configuration on page 108.
- For more information about how to use Zowe, see Getting started tutorial on page 127.

Configuring the z/OS system for Zowe

Configure the z/OS security manager to prepare for launching the Zowe started tasks.

If Zowe has already been launched on a z/OS system from a previous release of Version 1.8 or later, then you are applying a newer Zowe build. You can skip this security configuration step unless told otherwise in the release documentation.

A SAMPLIB JCL member ZWESECUR is provided to assist with the security configuration. You can submit the ZWESECUR JCL member as-is or customize it depending on site preferences. The JCL allows you to vary which security manager you use by setting the *PRODUCT* variable to be one of RACF, ACF2, or TSS.

```
//         SET PRODUCT=RACF          * RACF, ACF2, or TSS
```

If ZWESECUR encounters an error or a step that has already been performed, it will continue to the end, so it can be run repeatedly in a scenario such as a pipeline automating the configuration of a z/OS environment for Zowe installation.

It is expected that security administrator at a site will want to review, edit where necessary, and either execute ZWESECUR as a single job or else execute individual TSO commands one by one to complete the security configuration of a z/OS system in preparation for installing and running Zowe.

If you want to undo all of the z/OS security configuration steps performed by the JCL member ZWESECUR, Zowe provides a reverse member ZWENOSEC that contains the inverse steps that ZWESECUR performs. This is useful in the following situations:

- You are configuring z/OS systems as part of a build pipeline that you want to undo and redo configuration and installation of Zowe using automation.
- You have configured a z/OS system for Zowe that you no longer want to use and you prefer to delete the Zowe user IDs and undo the security configuration settings rather than leave them enabled.

If you run ZWENOSEC on a z/OS system, then you will no longer be able to run Zowe until you rerun ZWESECUR to reinitialize the z/OS security configuration.

User IDs and groups for the Zowe started tasks

Zowe requires a user ID ZWESVUSR to execute its main z/OS runtime started task ZWESVSTC. This userid must have a valid OMVS segment.

Zowe requires a user ID ZWESIUSR to execute the cross memory server started task ZWESISTC. This userid must have a valid OMVS segment.

Zowe requires a group ZWEADMIN that both ZWESVUSR and ZWESIUSR should belong to. This group must have a valid OMVS segment.

The JCL member ZWESECUR contains the TSO commands to create the user IDs.

- To create the ZWEADMIN group, issue the following command:

```
ADDGROUP ZWEADMIN. OMVS(AUTOGID) -
DATA('STARTED TASK GROUP WITH OMVS SEGEMENT')
```

- To create the ZWESVUSR user ID for the main Zowe started task, issue the following command:

```
ADDUSER  ZWESVUSR. -
  NOPASSWORD -
  DFLTGRP(ZWEADMIN) -
  OMVS(HOME(/tmp) PROGRAM(/bin/sh) AUTOUID) -
  NAME('ZOWE SERVER') -
  DATA('ZOWE MAIN SERVER')
```

- To create the ZWESIUSR group for the Zowe cross memory server started task, issue the following command:

```
ADDUSER ZWESIUSR. -
  NOPASSWORD -
  DFLTGRP(ZWEADMIN) -
  OMVS(HOME(/tmp) PROGRAM(/bin/sh) AUTOUID) -
```

```
  NAME('ZOWE XMEM SERVER') -
  DATA('ZOWE XMEM CROSS MEMORY SERVER')
```

Configure ZWESVSTC to run under ZWESVUSR user ID

When the Zowe started task `ZWESVSTC` is started, it must be associated with the user ID `ZWESVUSR` and group `ZWEADMIN`. A different user ID and group can be used if required to conform with existing naming standards.

- If you use RACF, issue the following commands:

```
RDEFINE STARTED ZWESVSTC.* UACC(NONE) STDATA(USER(ZWESVUSR)
 GROUP(ZWEADMIN) PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
SETROPTS REFRESH RACLIST(STARTED)
```

- If you use CA ACF2, issue the following commands:

```
SET CONTROL(GSO)
INSERT STC.ZWESVSTC LOGONID(ZWESVUSR) GROUP(ZWEADMIN) STCID(ZWESVSTC)
F ACF2,REFRESH(STC)
```

- If you use CA Top Secret, issue the following commands:

```
TSS ADDTO(STC) PROCNAME(ZWESVSTC) ACID(ZWESVUSR)
```

Grant users permission to access z/OSMF

TSO user IDs using Zowe must have permission to access the z/OSMF services that are used by Zowe. They should be added to the group with appropriate z/OSMF privileges, `IZUUSER` or `IZUADMIN` by default.

- If you use RACF, issue the following command:

```
CONNECT (userid) GROUP(IZUUSER)
```

- If you use CA ACF2, issue the following commands:

```
ACFNRULE TYPE(TGR) KEY(IZUUSER) ADD(UID(<uid string of user>) ALLOW)
F ACF2,REBUILD(TGR)
```

- If you use CA Top Secret, issue the following commands:

```
TSS ADD(userid)  PROFILE(IZUUSER)
TSS ADD(userid)  GROUP(IZUUSRGP)
```

Configure the cross memory server for SAF

Zowe has a cross memory server that runs as an APF-authorized program with key 4 storage. Client processes accessing the cross memory server's services must have READ access to a security profile `ZWES.IS` in the `FACILITY` class. This authorization step is used to guard against access by non-priviledged clients.

Activate the FACILITY class, define a `ZWES.IS` profile, and grant READ access to the user ID `ZWESVUSR`. This is the user ID that the Zowe started task `ZWESVSTC` runs under.

To do this, issue the following commands that are also included in the `ZWESECUR` JCL member. The commands assume that you run the `ZWESVSTC` under the `ZWESVUSR` user.

- If you use RACF, issue the following commands:
  - To see the current class settings, use:

    ```
    SETROPTS LIST
    ```

  - To define and activate the FACILITY class, use:

    ```
    SETROPTS GENERIC(FACILITY)
    SETROPTS CLASSACT(FACILITY)
    ```

  - To RACLIST the FACILITY class, use:

    ```
    SETROPTS RACLIST(FACILITY)
    ```

  - To define the `ZWES.IS` profile in the FACILITY class and grant Zowe's started task userid READ access, issue the following commands:

    ```
    RDEFINE FACILITY ZWES.IS UACC(NONE)
    ```

    ```
    PERMIT ZWES.IS CLASS(FACILITY) ID(<zwesvstc_user>) ACCESS(READ)
    ```

    where `<zwesvstc_user>` is the user ID `ZWESVUSR` under which the ZWESVSTC started task runs.

    ```
    SETROPTS RACLIST(FACILITY) REFRESH
    ```

  - To check whether the permission has been successfully granted, issue the following command:

    ```
    RLIST FACILITY ZWES.IS AUTHUSER
    ```

    This shows the user IDs who have access to the `ZWES.IS` class, which should include Zowe's started task userid with READ access.
- If you use CA ACF2, issue the following commands:

  ```
  SET RESOURCE(FAC)
  ```

  ```
  RECKEY ZWES ADD(IS ROLE(IZUSVR) SERVICE(READ) ALLOW)
  ```

  ```
  F ACF2,REBUILD(FAC)
  ```

- If you use CA Top Secret, issue the following commands, where `owner-acid` can be IZUSVR or a different ACID:

  ```
  TSS ADD(`owner-acid`) IBMFAC(ZWES.)
  ```

  ```
  TSS PERMIT(IZUSVR) IBMFAC(ZWES.IS) ACCESS(READ)
  ```

Notes:

- The cross memory server treats "no decision" style SAF return codes as failures. If there is no covering profile for the `ZWES.IS` resource in the FACILITY class, the request will be denied.
- Cross memory server clients other than Zowe might have additional SAF security requirements. For more information, see the documentation for the specific client.

Configure an ICSF cryptographic services environment

To generate symmetric keys, the `ZWESVUSR` user who runs `ZWESVSTC` requires READ access to `CSFRNGL` in the `CSFSERV` class.

Define or check the following configurations depending on whether ICSF is already installed:

- The ICSF or CSF job that runs on your z/OS system.
- The configuration of ICSF options in `SYS1.PARMLIB(CSFPRM00)`, `SYS1.SAMPLIB`, `SYS1.PROCLIB`.
- Create CKDS, PKDS, TKDS VSAM data sets.
- Define and activate the CSFSERV class:
  - If you use RACF, issue the following commands:

```
RDEFINE CSFSERV profile-name UACC(NONE)
```

```
PERMIT profile-name CLASS(CSFSERV) ID(tcpip-stackname) ACCESS(READ)
```

```
PERMIT profile-name CLASS(CSFSERV) ID(userid-list)   ... [for
```

```
userids IKED, NSSD, and Policy Agent]
```

```
SETROPTS CLASSACT(CSFSERV)
```

```
SETROPTS RACLIST(CSFSERV) REFRESH
```

- If you use CA ACF2, issue the following commands (note that `profile-prefix` and `profile-suffix` are user-defined):

```
SET CONTROL(GSO)
```

```
INSERT CLASMAP.CSFSERV RESOURCE(CSFSERV) RSRCTYPE(CSF)
```

```
F ACF2,REFRESH(CLASMAP)
```

```
SET RESOURCE(CSF)
```

```
RECKEY profile-prefix ADD(profile-suffix uid(UID string for tcpip-
stackname) SERVICE(READ) ALLOW)
```

```
RECKEY profile-prefix ADD(profile-suffix uid(UID string for IZUSVR)
 SERVICE(READ) ALLOW)
```

(repeat for userids IKED, NSSD, and Policy Agent)

```
F ACF2,REBUILD(CSF)
```

- If you use CA Top Secret, issue the following command (note that `profile-prefix` and `profile-suffix` are user defined):

```
TSS ADDTO(owner-acid) RESCLASS(CSFSERV)
```

```
TSS ADD(owner-acid) CSFSERV(profile-prefix.)
```

```
TSS PERMIT(tcpip-stackname) CSFSERV(profile-prefix.profile-suffix)
 ACCESS(READ)
```

```
TSS PERMIT(user-acid) CSFSERV(profile-prefix.profile-suffix)
 ACCESS(READ)
```

(repeat for user-acids IKED, NSSD, and Policy Agent)

Notes:

- Determine whether you want SAF authorization checks against `CSFSERV` and set `CSF.CSFSERV.AUTH.CSFRNG.DISABLE` accordingly.
- Refer to the z/OS 2.3.0 z/OS Cryptographic Services ICSF System Programmer's Guide: Installation, initialization, and customization.
- CCA and/or PKCS #11 coprocessor for random number generation.
- Enable `FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION` and `RDEFINE CSFINPV2` if required.

Configure security environment switching

Typically, the user `ZWESVUSR` that the `ZWESVSTC` started task runs under needs to be able to change the security environment of its process to allow API requests to be issued on behalf of the logged on TSO user ID, rather than the

server's user ID. This capability provides the functionality that allows users to log on to the Zowe desktop and use apps such as the File Editor to list data sets or USS files that the logged on user is authorized to view and edit, rather than the user ID running the Zowe server. This technique is known as **impersonation**.

To enable impersonation, you must grant the user ID `ZWESVUSR` associated with the `ZWESVSTC` started task UPDATE access to the `BPX.SERVER` and `BPX.DAEMON` profiles in the `FACILITY` class.

You can issue the following commands first to check whether you already have the impersonation profiles defined as part of another server configuration, such as the FTPD daemon. Review the output to confirm that the two impersonation profiles exist and the user `ZWESVUSR` who runs the `ZWESVSTC` started task has UPDATE access to both profiles.

- If you use RACF, issue the following commands:

```
RLIST FACILITY BPX.SERVER AUTHUSER
```

```
RLIST FACILITY BPX.DAEMON AUTHUSER
```

- If you use CA Top Secret, issue the following commands:

```
TSS WHOHAS IBMFAC(BPX.SERVER)
```

```
TSS WHOHAS IBMFAC(BPX.DAEMON)
```

- If you use CA ACF2, issue the following commands:

```
SET RESOURCE(FAC)
```

```
LIST BPX
```

If the user `ZWESVUSR` who runs the ZWESVSTC started task does not have UPDATE access to both profiles follow the instructions below.

- If you use RACF, complete the following steps:

  1. Activate and RACLIST the FACILITY class. This may have already been done on the z/OS environment if another z/OS server has been previously configured to take advantage of the ability to change its security environment, such as the FTPD daemon that is included with z/OS Communications Server TCP/IP services.

     ```
     SETROPTS GENERIC(FACILITY)
     ```

```
SETROPTS CLASSACT(FACILITY) RACLIST(FACILITY)
```

2. Define the impersonation profiles. This may have already been done on behalf of another server such as the FTPD daemon.

```
RDEFINE FACILITY BPX.SERVER UACC(NONE)
```

```
RDEFINE FACILITY BPX.DAEMON UACC(NONE)
```

3. Having activated and RACLIST the FACILITY class, the user ID ZWESVUSR who runs the ZWESVSTC started task must be given update access to the BPX.SERVER and BPX.DAEMON profiles in the FACILITY class.

```
PERMIT BPX.SERVER CLASS(FACILITY) ID(<zwesvstc_user>) ACCESS(UPDATE)
```

```
PERMIT BPX.DAEMON CLASS(FACILITY) ID(<zwesvstc_user>) ACCESS(UPDATE)
```

where <zwesvstc_user> is ZWESVUSR unless a different user ID is being used for the z/OS environment.

/* Activate these changes */

```
SETROPTS RACLIST(FACILITY) REFRESH
```

4. Issue the following commands to check whether permission has been successfully granted:

```
RLIST FACILITY BPX.SERVER AUTHUSER
```

```
RLIST FACILITY BPX.DAEMON AUTHUSER
```

- If you use CA Top Secret, complete the following steps:

1. Define the BPX Resource and access for <zwesvstc_user>.

```
TSS ADD(`owner-acid`) IBMFAC(BPX.)
```

```
TSS PERMIT(<zwesvstc_user>) IBMFAC(BPX.SERVER) ACCESS(UPDATE)
```

```
TSS PERMIT(<zwesvstc_user>) IBMFAC(BPX.DAEMON) ACCESS(UPDATE)
```

where <zwesvstc_user> is ZWESVUSR unless a different user ID is being used for the z/OS environment.

2. Issue the following commands and review the output to check whether permission has been successfully granted:

```
TSS WHOHAS IBMFAC(BPX.SERVER)
```

```
TSS WHOHAS IBMFAC(BPX.DAEMON)
```

- If you use CA ACF2, complete the following steps:

  1. Define the BPX Resource and access for <zwesvstc_user>.

     ```
     SET RESOURCE(FAC)
     ```

     ```
     RECKEY BPX ADD(SERVER ROLE(<zwesvstc_user>) SERVICE(UPDATE) ALLOW)
     ```

     ```
     RECKEY BPX ADD(DAEMON ROLE(<zwesvstc_user>) SERVICE(UPDATE) ALLOW)
     ```

     where <zwesvstc_user> is `ZWESVUSR` unless a different user ID is being used for the z/OS environment.

     ```
     F ACF2,REBUILD(FAC)
     ```

  2. Issue the following commands and review the output to check whether permission has been successfully granted:

     ```
     SET RESOURCE(FAC)
     ```

     ```
     LIST BPX
     ```

Configure address space job naming

The user ID `ZWESVUSR` that is associated with the Zowe started task `ZWESVSTC` must have `READ` permission for the `BPX.JOBNAME` profile in the `FACILITY` class. This is to allow setting of the names for the different z/OS UNIX address spaces for the Zowe runtime components. See Address space names on page 95.

To display who is authorized to the profile, issue the following command:

```
RLIST FACILITY BPX.JOBNAME AUTHUSER
```

Additionally, you need to activate facility class, permit `BPX.JOBNAME`, and refresh facility class:

```
SETROPTS CLASSACT(FACILITY) RACLIST(FACILITY)
PERMIT BPX.JOBNAME CLASS(FACILITY) ID(ZWESVUSR) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

For more information, see Setting up the UNIX-related FACILITY and SURROGAT class profiles in the "z/OS UNIX System Services" documentation.

Configuring Zowe certificates

A keystore directory is used by Zowe to hold the certificate used for encrypting communication between Zowe clients and the Zowe z/OS servers. It also holds the truststore used to hold public keys of any servers that Zowe trusts. When a Zowe is launched the instance directory configuration file `instance.env` specifies the location of the keystore directory, see Configure instance directory

If you have already created a keystore directory from a previous release of Version 1.8 or later, then you may reuse the existing keystore directory with newer version of Zowe.

You can use the existing certificate signed by an external certificate authority (CA) for HTTPS ports in the API Mediation Layer and the Zowe Application Framework, or else you can let the Zowe configuration script to generate a self-signed certificate by the local API Mediation CA.

If you let the Zowe configuration to generate a self-signed certificate, the certificates should be imported into your browser to avoid untrusted network traffic challenges. See Import the local CA certificate to your browser. If you do not import the certificates into your browser when you access a Zowe web page, you may be challenged that the web page cannot be trusted and, depending on the browser you are using, have to add an exception to proceed to the web page. Some browser versions may not accept the Zowe certificate because it is self-signed and the signing authority is

not recognized as a trusted source. Manually importing the certificate into your browser makes it a trusted source and the challenges will no longer occur.

If you have an existing server certificate that is signed by an external CA, then you use this for the Zowe certificate. This could be a CA managed by the IT department of your company which has already ensured that any certificates signed by that CA are trusted by browsers in your company because they have included their company's CA in their company's browsers' truststore. This will avoid the need to manually import the local CA into each client machine's browsers.

If you want to avoid the need to have each browser trust the CA that has signed the Zowe certificate, you can use a public certificate authority such as Symantec, Comodo, or GoDaddy to create a certificate. These certificates are trusted by all browsers and most REST API clients. However, this option involves a manual process of requesting a certificate and may incur a cost payable to the publicly trusted CA.

It's recommended that you start with the local API Mediation Layer CA for an initial evaluation.

You can use the `bin/zowe-setup-certificates.sh` script in the Zowe runtime directory to configure the certificates with the set of defined environment variables. The environment variables act as parameters for the certificate configuration are held in the file `bin/zowe-setup-certificates.env`.

Generate certificate with the default values

The script reads the default variable values that are provided in the `bin/zowe-setup-certificates.env` file and generates the certificate signed by the local API Mediation CA and keystores in the `/global/zowe/keystore` location. To set up certificates with the default environment variables, ensure that you run the following script in the Zowe installation directory:

```
bin/zowe-setup-certificates.sh
```

The keystore and certificates are generated in the default `/global/Zowe/keystore` directory. This can be overridden with the `-p` argument to the script.

Generate certificate with the custom values

We recommend that you review all the parameters in the `zowe-setup-certificates.env` file and customize the values for variables to meet your requirements. For example, set your preferred location to generate certificates and keystores.

Follow the procedure to customize the values for variables in the `zowe-setup-certificates.env` file:

1. Copy the `bin/zowe-setup-certificates.env` file from the read-only location to a new `<your_directory>/zowe-setup-certificates.env` location.
2. Customize the values for the variables based on the descriptions that are provided in the `zowe-setup-certificates.env` file.
3. Execute the following command with the customized environment file:

   ```
   bin/zowe-setup-certificates.sh –p <your_directory>/zowe-setup-certificates.env
   ```

   where `<your_directory>` specifies the location of your customized environment file.

The keystore and certificates are generated based on the customized values in the `bin/zowe-setup-certificates.env` file.

The `zowe-setup-certificates.sh` command also generates `zowe-certificates.env` file in the `KEYSTORE_DIRECTORY` directory. This file is used in the Zowe instance configuration step.

The following example shows how you can configure `zowe-setup-certificates.env` file to use the existing certificates:

1. Update the value of `EXTERNAL_CERTIFICATE`. The value needs to point to a keystore in PKCS12 format that contains the certificate with its private key. The file needs to be transferred as a binary to the z/OS system.

2. Update the value of `KEYSTORE_PASSWORD`. The value is a password to the PKCS12 keystore specified in the `EXTERNAL_CERTIFICATE` variable.
3. Update the value of `EXTERNAL_CERTIFICATE_ALIAS` to the alias of the server certificate in the keystore.

   Note: If you do not know the certificate alias, run the following command where `externalCertificate.p12` is a value of `EXTERNAL_CERTIFICATE` in the `zowe-setup-certificates.env` file.

   ```
   keytool -list -keystore externalCertificate.p12 -storepass password -
   storetype pkcs12 -v
   ```

   Expected output:

   ```
   Keystore type: PKCS12
   Keystore provider: SUN
   Your keystore contains 1 entry
   Alias name: apiml
   Creation date: Oct 9, 2019
   Entry type: PrivateKeyEntry
   Certificate chain length: 3
   ...
   ```

   In this case, the alias can be found in `Alias name: apiml`. Therefore, set `EXTERNAL_CERTIFICATE_ALIAS=apiml`.
4. Update the value of `EXTERNAL_CERTIFICATE_AUTHORITIES` to the path of the public certificate of the certificate authority that has signed the certificate. You can add additional certificate authorities separated by spaces (specify the complete value **in quotes**). This can be used for certificate authorities that have signed the certificates of the services that you want to access via the API Mediation Layer.
5. (Optional) If you have trouble getting the certificates and you want only to evaluate Zowe, you can switch off the certificate validation by setting `VERIFY_CERTIFICATES=false`. The HTTPS will still be used but the API Mediation Layer will not validate any certificate.

   Important! Switching off certificate evaluation is a non-secure setup.

Following is the part of `zowe-setup-certificates.env` file snippet that uses existing certificates:

```
# Should APIML verify certificates of services - true/false
VERIFY_CERTIFICATES=true
# optional - Path to a PKCS12 keystore with a server certificate for APIML
EXTERNAL_CERTIFICATE=/path/to/keystore.p12
# optional - Alias of the certificate in the keystore
EXTERNAL_CERTIFICATE_ALIAS=servercert
# optional - Public certificates of trusted CAs
EXTERNAL_CERTIFICATE_AUTHORITIES="/path/to/cacert_1.cer /path/to/
cacert_2.cer"
# Select a password that is used to secure EXTERNAL_CERTIFICATE keystore
 and
# that will be also used to secure newly generated keystores for API
 Mediation
KEYSTORE_PASSWORD=mypass
```

You may encounter the following message:

```
apiml_cm.sh --action trust-zosmf has failed.
WARNING: z/OSMF is not trusted by the API Mediation Layer. Follow
 instructions in Zowe documentation about manual steps to trust z/OSMF
```

This error does not interfere with the installation progress and can be remediated after the installation completes. For more information, see Trust a z/OSMF certificate.

Notes:

- On many z/OS systems, the certificate for z/OSMF is not signed by a trusted CA and is a self-signed certificate by the z/OS system programmer who configured z/OSMF. If that is the case, then Zowe itself will not trust the z/OSMF certificate and any function dependent on z/OSMF will not operate correctly. To ensure that Zowe trusts a z/OSMF self-signed certificate, you must use the value `VERIFY_CERTIFICATES=false` in the `zowe-setup-certificates.env` file. This is also required if the certificate is from a recognized CA but for a different host name, which can occur when a trusted certificate is copied from one source and reused within a z/OS installation for different servers other than that it was originally created for.
- In order to import the public key of the z/OSMF certificate into the Zowe certificate truststore, the user ID that is used to run the `zowe-setup-certificates.sh` script must have authority to read the z/OSMF keyring. See Trust a z/OSMF certificate.

Creating and configuring the Zowe instance directory

The Zowe instance directory contains configuration data required to launch a Zowe runtime. This includes port numbers, location of dependent runtimes such as Java, Node, z/OSMF, as well as log files. When Zowe is started, configuration data will be read from files in the instance directory and logs will be written to files in the instance directory.

Prerequisites

Before creating an instance directory, ensure that you have created a keystore directory that contains the Zowe certificate. For more information about how to create a keystore directory, see Configuring Zowe certificates on page 91. Also, ensure that you have already configured the z/OS environment. For information about how to configure the z/OS environment, see Configuring the z/OS system for Zowe on page 83.

Creating an instance directory

To create an instance directory, navigate to the Zowe runtime directory `<ZOWE_ROOT_DIR>` and execute the following commands:

```
<ROOT_DIR>/bin/zowe-configure-instance.sh -c <PATH_TO_INSTANCE_DIR>
```

Multiple instance directories can be created and used to launch independent Zowe runtimes from the same Zowe runtime directory.

The Zowe instance directory contains a file `/bin/instance.env` that stores configuration data. The data is read each time Zowe is started.

The purpose of the instance directory is to hold information in the z/OS File System (zFS) that is created (such as log files) or modified (such as preferences) or configured (such as port numbers) away from the zFS runtime directory for Zowe. This allows the runtime directory to be read only and to be replaced when a new Zowe release is installed, with customizations being preserved in the instance directory.

If you have an instance directory that is created from a previous release of Zowe 1.8 or later and are installing a newer release of Zowe, then you should run `zowe-configure-instance.sh -c <PATH_TO_INSTANCE_DIR>` pointing to the existing instance directory to have it updated with any new values. The release documentation for each new release will specify when this is required, and the file `manifest.json` within each instance directory contains information for which Zowe release it was created from.

Reviewing the `instance.env` file

To operate Zowe, a number of zFS folders need to be located for prerequisites on the platform. Default values are selected when you run `zowe-configure-instance.sh`. You might want to modify the values.

Component groups

`LAUNCH_COMPONENT_GROUPS` : This is a comma-separated list of which z/OS microservice groups are started when Zowe launches.

- `GATEWAY` will start the API mediation layer, which includes the API catalog, the API gateway, and the API discovery service. These three address spaces are Apache Tomcat servers and use the version of Java on z/OS as determined by the `JAVA_HOME` value.
- `DESKTOP` will start the Zowe desktop, which is the browser GUI for hosting Zowe applications such as the 3270 Terminal and the File Explorer apps. The Zowe desktop is a Node.js application and uses the version that is specified by the `HOME_HOME` value.

Component prerequisites

- `JAVA_HOME`: The path where 64-bit Java 8 or later is installed. Only needs to be specified if not already set as a shell variable. Defaults to `/usr/lpp/java/J8.0_64`.
- `NODE_HOME`: The path to the Node.js runtime. Only needs to be specified if not already set as a shell variable.
- `ROOT_DIR`: The directory where the Zowe runtime is located. Defaults to the location of where `zowe-configure-instance` was executed.
- `ZOSMF_PORT`: The port used by z/OSMF REST services. Defaults to value determined through running `netstat`.
- `ZOSMF_HOST`: The host name of the z/OSMF REST API services.
- `ZOWE_EXPLORER_HOST`: The hostname of where the Explorer servers are launched from. Defaults to running `hostname -c`. Ensure that this host name is externally accessible from clients who want to use Zowe as well as internally accessible from z/OS itself.
- `ZOWE_IP_ADDRESS`: The IP address of your z/OS system which must be externally accessible to clients who want to use Zowe. This is important to verify for IBM Z Development & Test Environment and cloud systems, where the default that is determined through running `ping` and `dig` on z/OS return a different IP address from the external address.
- `APIML_ENABLE_SSO`: Define whether single sign-on should be enabled. Use a value of `true` or `false`. Defaults to `false`.

Keystore configuration

- `KEYSTORE_DIRECTORY`: This is a path to a zFS directory containing the certificate that Zowe uses to identify itself and encrypt https:// traffic to its clients accessing REST APIs or web pages. This also contains a truststore used to hold the public keys of any z/OS services that Zowe is communicating to, such as z/OSMF. The keystore directory must be created the first time Zowe is installed onto a z/OS system and it can be shared between different Zowe runtimes. For more information about how to create a keystore directory, see Configuring Zowe certificates on page 91.

Address space names

Individual address spaces for different Zowe instances and their subcomponents can be distinguished from each other in RMF records or SDSF views by specifying how they are named. Address space names are 8 characters long and made up of a prefix `ZOWE_PREFIX`, instance `ZOWE_INSTANCE` followed by an identifier for each subcomponent.

- `ZOWE_PREFIX`: This defines a prefix for Zowe address space STC names. Defaults to `ZWE`.
- `ZOWE_INSTANCE`: This is appended to the `ZOWE_PREFIX` to build up the address space name. Defaults to `1`
- A subcomponent will be one of the following values:
  - **AC** - API ML Catalog
  - **AD** - API ML Discovery Service
  - **AG** - API ML Gateway
  - **DS** - Node.js instance for the ZSS Server
  - **DT** - Zowe Desktop Application Server
  - **EF** - Explorer API Data Sets
  - **EJ** - Explorer API Jobs
  - **SZ** - ZSS Server
  - **UD** - Explorer UI Data Sets
  - **UJ** - Explorer UI Jobs
  - **UU** - Explorer UI USS

The STC name of the main started task is `ZOWE_PREFIX+ZOWE_INSTANCE+SV`.

Example:

```
ZOWE_PREFIX=ZWE
ZOWE_INSTANCE=X
```

the first instance of Zowe API ML Gateway identifier will be as follows:

```
ZWEXAG
```

Note: If the address space names are not assigned correctly for each subcomponents, check that the step Configure address space job naming on page 91 has been performed correctly for the z/OS user ID `ZWESVUSR`.

Ports

When Zowe starts, a number of its microservices need to be given port numbers that they can use to allow access to their services. The two most important port numbers are the `GATEWAY_PORT` which is for access to the API gateway through which REST APIs can be viewed and accessed, and `ZOWE_ZLUX_SERVER_HTTPS_PORT` which is used to deliver content to client web browsers logging in to the Zowe desktop. All of the other ports are not typically used by clients and used for intra-service communication by Zowe.

- `CATALOG_PORT`: The port the API catalog service will use.
- `DISCOVERY_PORT`: The port the discovery service will use.
- `GATEWAY_PORT`: The port the API gateway service will use. This port is used by REST API clients to access z/OS services through the API mediation layer, so should be accessible to these clients. This is also the port used to log on to the API catalog web page through a browser.
- `JOBS_API_PORT`: The port the jobs API service will use.
- `FILES_API_PORT`: The port the files API service will use.
- `JES_EXPLORER_UI_PORT`: The port the jes-explorer UI service will use.
- `MVS_EXPLORER_UI_PORT`: The port the mvs-explorer UI service will use.
- `USS_EXPLORER_UI_PORT`: The port the uss-explorer UI service will use.
- `ZOWE_ZLUX_SERVER_HTTPS_PORT`: The port used by the Zowe desktop. It should be accessible to client machines with browsers wanting to log on to the Zowe desktop.
- `ZOWE_ZSS_SERVER_PORT`: This port is used by the ZSS server.

Note: If all of the default port values are acceptable, the ports do not need to be changed. To allocate ports, ensure that the ports are not in use for the Zowe runtime servers.

To determine which ports are not available, follow these steps:

1. Display a list of ports that are in use with the following command:

   ```
   TSO NETSTAT
   ```

2. Display a list of reserved ports with the following command:

   ```
   TSO NETSTAT PORTLIST
   ```

Terminal ports

Note: Unlike the ports needed by the Zowe runtime for its Zowe Application Framework and z/OS Services which must be unused, the terminal ports are expected to be in use.

- `ZOWE_ZLUX_SSH_PORT`: The Zowe desktop contains an application *VT Terminal* which opens a terminal session to z/OS UNIX System Services (USS) inside the Zowe desktop web page. This port is the number used by the z/OS SSH service and defaults to 22. The USS command `netstat -b | grep SSHD1` can be used to display the SSH port used on a z/OS system.
- `ZOWE_ZLUX_TELNET_PORT`: The Zowe desktop contains an application *3270 Terminal* which opens a 3270 terminal emulator inside the Zowe desktop web page. This port is the number used by the TN3270E server and

defaults to 23. The USS command `netstat -b | grep TN3270` can be used to display the TN3270E server port used on a z/OS system.

- `ZOWE_ZLUX_SECURITY_TYPE`: The *3270 Terminal* application needs to know whether the TN3270E server is using `tls` for security. The default value is blank for `telnet`, meaning that the connection between the Zowe Application Server and the TN3270E server is not encrypted. Configuring TLS is recommended, particularly if the Zowe Application Server connects to the TN3270E server via any physical network.

Installing and configuring the Zowe cross memory server (ZWESISTC)

The Zowe cross memory server provides privileged cross-memory services to the Zowe Desktop and runs as an APF-authorized program. The same cross memory server can be used by multiple Zowe desktops. You must install, configure, and launch the cross memory server if you want to use the Zowe desktop. Otherwise, you can skip this step.

To install and configure the cross memory server, you must create or edit APF-authorized load libraries, program properties table (PPT) entries, and a parmlib. This requires familiarity with z/OS.

The cross memory server runtime artifacts, the JCL for the started tasks, the parmlib, and members containing sample configuration commands are installed in the `SZWESAMP` PDS SAMPLIB. The load modules for the cross memory server and an auxiliary server it uses are installed in the `SZWEAUTH` PDS load library. The location of these for a convenience build depends on the value of the `zowe-install.sh -h` argument, see Step 3: Choose a dataset HLQ for the SAMPLIB and LOADLIB on page 64. For an SMP/E installation, the location is the value of `$datasetPrefixIn` in the member `AZWE001.F1(ZWE3ALOC)`.

The cross memory server is a long running angel process server that runs under the started task `ZWESISTC` with the user ID `ZWESIUSR` and group of `ZWEADMIN`.

The `ZWESISTC` started task runs the load module `ZWESIS01`, serves the Zowe desktop that is running under the `ZWESVSTC` started task, and provides it with secure services that require elevated privileges, such as supervisor state, system key, or APF-authorization.

Under some situations in support of a Zowe extension, the cross memory server will start, control, and stop an auxiliary address space. This run as a `ZWESASTC` started task that runs the load module `ZWESAUX`. Under normal Zowe operation, you will not see any auxiliary address spaces started. However, if you have installed a vendor product running on top of Zowe, this may use the auxiliary service so it should be configured to be launchable.

To install the cross memory server, take the following steps either Copy cross memory data set members manually on page 98 by using `cp` commands or use the supplied Copy cross memory data set members automatically on page 98 `zowe-install-xmem.sh` for an automated installation process.

Step 1: Copy the cross memory PROCLIB and load library

Copy cross memory data set members manually

1. Copy the load modules and add JCL to a PROCLIB:

   For the cross memory server to be started, its load modules need to be moved to an APF-authorized PDSE, and its JCL PROCLIB members moved to a PDS in the JES concatenation path.

   a. **Load modules** The cross memory server has two load modules, `ZWESIS01` and `ZWESAUX`, provided in the PDS `SZWEAUTH` created during the installation of Zowe. To manually copy the files to a user-defined data set, you can issue the following commands:

   ```
   cp -X ZWESIS01 "//'<zwes_loadlib>(ZWESIS01)'"
   ```

   ```
   cp -X ZWESAUX "//'<zwes_loadlib>(ZWESAUX)'"
   ```

   Where `<zwes_loadlib>` is the name of the data set, for example ZWES.SISLOAD. The `<zwes_loadlib>` data set must be a PDSE due to language requirements.

   b. **Prob libraries** The cross memory server PROCLIB JCL is `ZWESISTC` and the auxiliary address space PROCLIB JCL is `ZWESASTC`.

   You must specify the `<zwes_loadlib>` data set where `ZWESIS01` and `ZWESAUX` were copied to, in the STEPLIB DD statement of the two PROCLIB JCL members `ZWESISTC` and `ZWESASTC` respectively so that the appropriate version of the software is loaded correctly.

   Do not add the `<zwes_loadlib>` data set to the system LNKLST or LPALST concatenations.
2. Add a `ZWESIP00` PARMLIB member for the `ZWESISTC` started task:

   When started, the ZWESISTC started task must find a valid ZWESIPxx PARMLIB member. The `SZWESAMP` PDS contains the member `ZWESIP00` containing default configuration values. You can copy this member to your system PARMLIB data set, or allocate the default PDS data set ZWES.SISAMP that is specified in the ZWESISTC started task JCL.

Copy cross memory data set members automatically

Instead of the manual steps Copy cross memory data set members manually on page 98, a convenience script `<ROOT_DIR>/scripts/utils/zowe-install-xmem.sh` is shipped with Zowe to help with copying the cross memory and auxiliary address space PROCLIB members, the PARMLIB member, and the load libraries.

The script `zowe-install-xmem.sh` takes four arguments:

- **First Parameter**=Source PDS Prefix

  Data set prefix of the source PDS where .SZWESAMP(ZWESVSTC) was installed into.

  For an installation from a convenience build, this will be the value of `zowe-install.sh -h` when the build was installed. See Step 3: Choose a dataset HLQ for the SAMPLIB and LOADLIB on page 64

  For an SMP/E installation, this will be the value of `$datasetPrefixIn` in the member `AZWE001.F1(ZWE3ALOC)`.
- **Second Parameter**=Target DSN Load Library

  This is the data set name of the PDSE where members `ZWESIS01` and `ZWESAUX` will be copied into. This must be an APF-authorized PDS.
- **Third Parameter**=Target DSN for PARMLIB

  This is the data set name of where the PARMLIB `ZWESIP00` will be placed.
- **Fourth Parameter**=Target DSN for PROCLIB

  Target PROCLIB PDS where ZWESVSTC will be placed. If parameter is omitted the script scans the JES PROCLIB concatenation path and uses the first data set where the user has write access

Example:

Executing the command `zowe-install-xmem.sh MYUSERID.ZWE SYS1.IBM.ZIS.SZISLOAD SYS1.IBM.PARMLIB USER.PROCLIB` with four parameters specified copies:

- the load modules `MYUSERID.ZWE.SZWEAUTH(ZWESIS01)` and `MYUSERID.ZWE.SZWEAUTH(ZWESAUX)` to the load library `SYS.IBM.ZIS.SZISLOAD`
- the PARMLIB member `MYUSERID.ZWE.SZWESAMP(ZWESIP00)` to `SYS1.IBM.PARMLIB(ZWESIP00)`
- the PROCLIB member `MYUSERID.ZWE.SZWESAMP(ZWESISTC)` to `USER.PROCLIB(ZWESISTC` and `MYUSERID.ZWE.SZWESAMP(ZWESASTC)` to `USER.PROCLIB(ZWESASTC)`

The user ID `ZWESIUSR` that is assigned to the cross memory server started tasks must have a valid OMVS segment and read access to the data sets where the load library and PROCLIB are held. The cross memory server loads the modules to LPA for its PC-cp services.

Step 2: Add PPT entries to the system PARMLIB

The cross memory server and its auxiliary address spaces must run in key 4 and be non-swappable. For the server to start in this environment, add the following PPT entries for the server and address spaces to the SCHEDxx member of the system PARMLIB.

```
PPT PGMNAME(ZWESIS01) KEY(4) NOSWAP
```

```
PPT PGMNAME(ZWESAUX) KEY(4) NOSWAP
```

The PDS member `SZWESAMP(ZWESISCH)` contains the PPT lines for reference.

Then, issue the following command to make the SCHEDxx changes effective:

```
/SET SCH=xx
```

Step 3: Add the load libraries to the APF authorization list

Because the cross memory server provides privileges services, its load libraries require APF-authorization. To check whether a load library is APF-authorized, you can issue the following TSO command:

```
D PROG,APF,DSNAME=ZWES.SISLOAD
```

where the value of DSNAME is the name of the data set that contains the ZWESIS01 and ZWESAUX load modules.

To dynamically add a load library to the APF list if the load library is not SMS-managed, issue the following TSO command:

```
SETPROG APF,ADD,DSNAME=ZWES.SISLOAD,VOLUME=volser
```

If the load library is SMS-managed, issue the following TSO command:

```
SETPROG APF,ADD,DSNAME=ZWES.SISLOAD,SMS
```

where the value of DSNAME is the name of the data set that contains the ZWESIS01 and ZWESAUX load modules.

If you want to authorize the loadlib permanently, then add the following statement to `SYS1.PARMLIB(PROGxx)` or equivalent

The PDS member `SZWESAMP(ZWESIMPRG)` contains the SETPROG statement for reference.

Step 4: Configure SAF

The cross memory server performs a sequence of SAF checks to protect its services from unauthorized callers. To do this, it uses the FACILITY class and a `ZWES.IS` entry. Valid callers must have READ access to the `ZWES.IS` profile. Those callers include the STC user `ZWESVUSR` under which the ZWESVSTC started task runs. It is recommended that you also grant READ access to the STC user under which the ZWESASTC started task runs which is `ZWESIUSR`.

The commands required to configure SAF for the cross memory server are included in a JCL member `ZWESECUR` that is delivered with Zowe, see Configuring z/OS system

Step 5: Configure an IVSF cryptographic services environment

To generate symmetric keys, the user `ZWESVUSR` who runs `ZWESVSTC` requires READ access to CSFRNGL in the CSFSERV class.

For commands required to configure ICSF cryptographic services environment for symmetric key generation, see Configuring z/OS system.

Step 6: Configure security environment switching

When responding to API requests, the Zowe desktop node API server running under USS must be able to change the security environment of its process to associate itself with the security context of the logged in user. This is called impersonation.

For commands required to configure impersonation, see Configure security environment switching on page 88.

Starting and stopping the cross memory server on z/OS

The cross memory server is run as a started task from the JCL in the PROCLIB member `ZWESISTC`. It supports reusable address spaces and can be started through SDSF with the operator start command with the REUSASID=YES keyword:

```
/S ZWESISTC,REUSASID=YES
```

The ZWESISTC task starts and stops the ZWESSTC task as needed. Do not start the ZWESASTC task manually.

To end the Zowe cross memory server process, issue the operator stop command through SDSF:

```
/P ZWESISTC
```

Note:

The starting and stopping of the `ZWESVSTC` for the main Zowe servers is independent of the `ZWESISTC` cross memory server that is an angel process. If you are running more than one `ZWESVSTC` instance on the same LPAR, then these will be sharing the same `ZWESISTC` cross memory server. Stopping `ZWESISTC` will affect the behavior of all Zowe servers on the same LPAR that use the same cross-memory server name, for example ZWESIS_STD. The Zowe Cross Memory Server is designed to be a long-lived address space. There is no requirement to recycle regularly. When the cross-memory server is started with a new version of the ZWESIS01 load module, it abandons its current load module instance in LPA and loads the updated version.

To diagnose problems that may occur with the Zowe `ZWESVSTC` being able to attach to the `ZWESISTC` cross memory server, a log file `zssServer-yyyy-mm-dd-hh-mm.log` is created in the instance directory `/logs` folder each time a Zowe `ZWESVSTC` instance is started. More details on diagnosing errors can be found in Cannot log in to the Zowe Desktop on page 304

Installing the Zowe started task (ZWESVSTC)

Zowe has a number of runtimes on z/OS: the z/OS Service microservice server, the Zowe Application Server, and the Zowe API Mediation Layer microservices. A single PROCLIB `ZWESVSTC` is used to start all of these microservices. This member is installed by Zowe into the data set SAMPLIB `SZWESAMP` during the installation or either a convenience build or SMP/E. This topic describes how to configure the z/OS runtime in order to launch the Zowe started task.

Step 1: Copy the PROCLIB member ZWESVSTC

When the Zowe runtime is launched, it is run under a z/OS started task (STC) with the PROCLIB member named `ZWESVSTC`. A sample PROCLIB is created during installation into the PDS SZWESAMP. To launch Zowe as a started task, you must copy the member `ZWESVSTC` to a PDS that is in the proclib concatenation path.

If your site has your own technique for PROCLIB creation, you may follow this and copy the `ZWESVSTC` as-is. If you want to create a pipeline or automate the PROCLIB copying, you can use a convenience script `zowe-install-proc.sh` that is provided in the `<ROOT_DIR>/scripts/utils` folder.

The script `zowe-install-proc.sh` has two arguments:

- **First Parameter**=Source PDS Prefix

  Dataset prefix of the source PDS where `.SZWESAMP(ZWESVSTC)` was installed into.

  For an installation from a convenience build, this will be the value of the `-h` argument when `zowe-install.sh` was executed.

  For an SMP/E installation, this will be the value of `$datasetPrefixIn` in the member `AZWE001.F1(ZWE3ALOC)`.
- **Second Parameter**=Target PROCLIB PDS

  Target PROCLIB PDS where ZWESVSTC will be placed. If parameter is omitted, the script scans the JES PROCLIB concatenation path and uses the first data set where the user has write access

  Example

  Executing the command `zowe-install-proc.sh MYUSERID.ZWE USER.PROCLIB` copies the PDS member `MYUSERID.ZWE.SZWESAMP(ZWESVSTC)` to `USER.PROCLIB(ZSWESAMP)`

Step 2: Configure ZWESVSTC to run under the correct user ID

The `ZWESVSTC` must be configured as a started task (STC) under the ZWESVUSR user ID with the administrator user ID of ZWEADMIN. The commands to create the user ID and group is supplied in the PDS member `ZWESECUR`, see Configuring the z/OS system for Zowe on page 83. To associate the `ZWESVSTC` started task with the user ID and group see Configuring the z/OS system for Zowe on page 83. This step will be done once per z/OS environment by a system programmer who has sufficient security privileges.

Step 3: Launch the ZWESVSTC started task

You can launch the Zowe started task in two ways.

Option 1: Starting Zowe from a USS shell

To launch the `ZWESVSTC` started task, run the `zowe-start.sh` script from a USS shell. This reads the configuration values from the `zowe-instance.env` file in the Zowe instance directory.

```
cd <ZOWE_INSTANCE_DIR>/bin
./zowe-start.sh
```

where,

<ZOWE_INSTANCE_DIR> is the directory where you set the instance directory to. This script starts `ZWESVSTC` for you so you do not have to log on to TSO and use SDSF.

Option 2: Starting Zowe with a `/S` TSO command

You can use SDSF to start Zowe.

If you issue the SDSF command `/S ZWESVSTC`, it will fail because the script needs to know the instance directory containing the configuration details.

If you have a default instance directory you want you always start Zowe with, you can tailor the JCL member `ZWESVSTC` at this line

```
//ZWESVSTC   PROC INSTANCE='{{instance_directory}}'
```

to replace the `instance_directory` with the location of the Zowe instanceDir that contains the configurable Zowe instance directory.

If the JCL value `instance-directory` is not specified in the JCL, in order to start the Zowe server from SDSF, you will need to add the `INSTANCE` parameter on the START command when you start Zowe in SDSF:

```
/S ZWESVSTC,INSTANCE='$ZOWE_INSTANCE_DIR',JOBNAME='ZWEXSV'
```

The `JOBNAME='ZWEXSV'` is optional and the started task will operate correctly without it, however having it specified ensures that the address spaces will be prefixed with `ZWEXSV` which makes them easier to find in SDSF or locate in RMF records.

Stopping the ZWESVSTC PROC

To stop the Zowe server, the ZWESVSTC PROC needs to be ended. Run the `zowe-stop.sh` script at the Unix Systems Services command prompt that is in the zowe instance directory used to start the Zowe started task:

```
cd $ZOWE_INSTANCE_DIR/bin
./zowe-stop.sh
```

where <*ZOWE_INSTANCE_DIR*> is the directory where you set the instance directory to.

When you stop ZWESVSTC, you might get the following error message:

```
IEE842I ZWESVSTC DUPLICATE NAME FOUND- REENTER COMMAND WITH 'A='
```

This error results when there is more than one started task named ZWESVSTC. To resolve the issue, stop the required ZWESVSTC instance by issuing the following commands:

```
/C ${ZOWE_PREFIX}${ZOWE_INSTANCE}SV,A=asid
```

Where *ZOWE_PREFIX* and *ZOWE_INSTANCE* are specified in your configuration (and default to ZWE and 1) and you can obtain the *asid* from the value of `A=asid` when you issue the following commands:

```
/D A,${ZOWE_PREFIX}${ZOWE_INSTANCE}SV
```

Verifying Zowe installation on z/OS

After Zowe™ is running, follow the instructions in the following sections to verify that the components are installed correctly and are functional.

- Verifying Zowe Application Framework installation on page 102
- Verifying z/OS Services installation
- Verifying API Mediation installation on page 103

Verifying Zowe Application Framework installation

If the Zowe Application Framework is installed correctly, you can open the Zowe Desktop from a supported browser.

From a supported browser, open the Zowe Desktop at `https://myhost:httpsPort/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html`

where:

- *myHost* is the host on which you installed the Zowe Application Server.
- *httpPort* is the port number that is assigned to *node.http.port* in `zluxserver.json`.
- *httpsPort* is the port number that is assigned to *node.https.port* in `zluxserver.json`.

  For example, if the Zowe Application Server runs on host *myhost* and the port number that is assigned to *node.https.port* is 12345, you specify `https://myhost:12345/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html`.

Verifying z/OS Services installation

After the ZWESVSTC procedure is started, you can verify the installation of z/OS Services from an internet browser by entering the following case-sensitive URL:

```
https://hostName:<_gatewayPort_>/api/v1/jobs?prefix=*
```

where, *gatewayPort* is the port number that is assigned to `ZOWE_ZLUX_SERVER_HTTPS_PORT` in the instance.env file used to launch Zowe, see Ports on page 96.

Verifying API Mediation installation

Use your preferred REST API client to review the value of the status variable of the API Catalog service that is routed through the API Gateway using the following URL:

```
https://hostName:basePort/api/v1/apicatalog/application/health
```

The `hostName` is set during installation, and `basePort` is set as the `gatewayPort` parameter.

Example:

The following example illustrates how to use the **curl** utility to invoke API Mediation Layer endpoint and the **grep** utility to parse out the response status variable value

```
$ curl -v -k --silent https://hostName:basePort/api/v1/apicatalog/
application/health 2>&1 | grep -Po '(?<=\"status\":\")[^\"]+'
UP
```

The response `UP` confirms that API Mediation Layer is installed and is running properly.

Zowe Auxiliary Address space

The cross memory server runs as a started task `ZWESISTC` that uses the load module `ZWESIS01`.

In some use cases, the Zowe cross memory server has to spawn child address spaces, which are known as auxiliary (AUX) address spaces. The auxiliary address spaces run as the started task `ZWESASTC` using the load module `ZWESAUX` and are started, controlled, and stopped by the cross memory server.

An example of when an auxiliary address space is used is for a system service that requires supervisor state but cannot run in cross-memory mode. The service can be run in an AUX address space which is invoked by the Cross Memory Server acting as a proxy for unauthorized users of the service.

Do not install the Zowe auxiliary address space unless a Zowe extension product's installation guide explicitly asks for it to be done. This will occur if the extension product requires services of Zowe that cannot be performed by the cross memory server and an auxiliary address space needs to be started.

A default installation of Zowe does not require auxiliary address spaces to be configured.

You do not start or stop the ZWESASTC manually.

Uninstalling Zowe from z/OS

You can uninstall Zowe™ from z/OS if you no longer need to use it.

Follow these steps:

1. Stop the Zowe started task which stops all of its microservices by using the following command:

   ```
   /C ${ZOWE_PREFIX}${ZOWE_INSTANCE}SV
   ```

   Where ZOWE_PREFIX and ZOWE_INSTANCE are specified in your configuration (and default to ZWE and 1), see Address space names on page 95

   Aftter Zowe has been stopped its subcomponents will end which include the API Mediation Layer and the Zowe desktop.
2. Delete the `ZWESVSTC` member from your system `PROCLIB` data set.

   To do this, you can issue the following TSO DELETE command from the TSO READY prompt or from ISPF option 6:

   ```
   delete 'your.zowe.proclib(zwesvstc)'
   ```

   Alternatively, you can issue the TSO DELETE command at any ISPF command line by prefixing the command with TSO:

   ```
   tso delete 'your.zowe.proclib(zwesvstc)'
   ```

   To query which PROCLIB data set that ZWESVSTC is put in, you can view the SDSF JOB log of ZWESVSTC and look for the following message:

   ```
   IEFC001I PROCEDURE ZWESVSTC WAS EXPANDED USING SYSTEM LIBRARY
    your.zowe.proclib
   ```

   If no ZWESVSTC JOB log is available, issue the `/$D PROCLIB` command at the SDSF COMMAND INPUT line and BROWSE each of the `DSNAME=some.jes.proclib` output lines in turn with ISPF option 1, until you find the first data set that contains member ZWESVSTC. Then, issue the DELETE command as shown above.

   After you have removed `ZWESVSTC` from the `PROCLIB` data set it will no longer be possible to start Zowe instances.
3. Remove the USS folders containing the Zowe artifacts

   Remove the USS folders containing the Zowe runtime, the Zowe keystore-directory, and the Zowe instance directories.
4. Reverse the z/OS security and environment updates from `ZWESECUR` job

   As part of installing Zowe the z/OS environment will have been altered to allow Zowe to operate, see Configuring the z/OS System for Zowe. You may leave the environment configured which allows you to install and operate a Zowe instance at a point in the future, or you may undo the configuration steps to your z/OS environment. A JCL member `ZWENOSEC` is provided with Zowe that contains the commands needed to reset a z/OS environment and undo the steps that were performed in `ZWESECUR` when the environment was configured for Zowe operation.

Installing Zowe CLI

Installing Zowe CLI

Install Zowe™ CLI on your computer. You can learn about new CLI features in the Release notes on page 14 or read about overall CLI functionality in the Zowe overview on page 8.

Tip: If you are familiar with command-line tools and want to get started using Zowe CLI quickly, see Zowe CLI quick start on page 29

Methods to install Zowe CLI

Use one of the following methods to install Zowe CLI.

- Installing Zowe CLI from a local package on page 105
- Installing Zowe CLI from an online registry on page 106

If you encounter problems when you attempt to install Zowe CLI, see Troubleshooting Zowe CLI on page 308.

Installing Zowe CLI from a local package

If you do not have internet access at your site, use the following method to install Zowe CLI from a local package.

Follow these steps:

1. Ensure that the following prerequisite software is installed on your computer:
   - **Node.js V8.0 or later**

     Tip: You might need to restart the command prompt after installing Node.js. Issue the command `node --version` to verify that Node.js is installed.
   - **Node Package Manager V5.0 or later**

     npm is included with the Node.js installation. Issue the command `npm --version` to verify that npm is installed.
2. Obtain the installation files. From the Zowe Download website, click **Zowe Command Line Interface** to download the Zowe CLI installation package named `zowe-cli-package-*v*.*r*.*m*.zip` to your computer.

   Note:

   - *v* indicates the version
   - *r* indicates the release number
   - *m* indicates the modification number
3. Open a command line window such as Windows Command Prompt. Browse to the directory where you downloaded the Zowe CLI installation package (.zip file). Issue the following command to unzip the files:

   ```
   unzip zowe-cli-package-v.r.m.zip
   ```

   Example:

   ```
   unzip zowe-cli-package-1.0.1.zip
   ```

   By default, the unzip command extracts the contents of the zip file to the directory where you downloaded the .zip file. You can extract the contents of the zip file to your preferred location.

   Optional: Double-click the Zowe CLI installation package to extract its contents into the directory where you downloaded the package. (Windows and Mac computers contain built-in software that lets you extract .zip files into a preferred location.)
4. Issue the following command against the extracted directory to install Zowe CLI on your computer:

   ```
   npm install -g zowe-cli.tgz
   ```

   Note: On Linux, you might need to prepend `sudo` to your `npm` commands so that you can issue the install and uninstall commands. For more information, see Troubleshooting Zowe CLI on page 308.

   Zowe CLI is installed on your computer. See Installing Zowe CLI plug-ins on page 154 for information about the commands for installing plug-ins from the package.
5. (Optional) Create a `zosmf` profile so that you can issue commands that communicate with z/OSMF. For information about how to create a profile, see Creating Zowe CLI profiles on page 123.

   Tip: Profiles are a Zowe CLI feature that let you store configuration information for use on multiple commands. You can create a profile that contains your username, password, and connection details for a particular mainframe system, then reuse that profile to avoid typing it again on every command.

After you install and configure Zowe CLI, you can issue the `zowe --help` command to view a list of available commands. For information about how to connect the CLI to the mainframe (using command-line options, user profiles, or environment variables), see Defining Zowe CLI connection details on page 122. You can also test your connection to z/OSMF with or without a profile.

Installing Zowe CLI from an online registry

If your computer is connected to the Internet, you can use the following method to install Zowe CLI from an npm registry.

Follow these steps:

1. Ensure that the following required software is installed on your computer:
   - **Node.js V8.0 or later**

     Tip: You might need to restart the command prompt after installing Node.js. Issue the command `node --version` to verify that Node.js is installed.
   - **Node Package Manager V5.0 or later**

     npm is included with the Node.js installation. Issue the command `npm --version` to verify that npm is installed.
2. Issue the following command to set the registry to the Zowe CLI scoped package. In addition to setting the scoped registry, your default registry must be set to an npm registry that includes all of the dependencies for Zowe CLI, such as the global npm registry:

   ```
   npm config set @brightside:registry https://api.bintray.com/npm/ca/brightside
   ```
3. Issue the following command to install Zowe CLI from the registry:

   ```
   npm install -g @brightside/core@lts-incremental
   ```
4. (Optional) To install all available plug-ins to Zowe CLI, issue the following command:

   ```
   zowe plugins install @brightside/cics@lts-incremental @brightside/db2@lts-incremental
   ```

   Note: The IBM Db2 plug-in requires additional configuration. For more information about how to install multiple plug-ins, update to a specific version of a plug-in, and install from specific registries, see Installing Zowe CLI plug-ins on page 154.

After you install and configure Zowe CLI, you can issue the `zowe --help` command to view a list of available commands. For information about how to connect the CLI to the mainframe (using command-line options, user profiles, or environment variables), see Defining Zowe CLI connection details on page 122. You can also test your connection to z/OSMF with or without a profile.

Updating Zowe CLI

Zowe™ CLI is updated continuously. You can update Zowe CLI to a more recent version using online registry method or the local package method. However, you can only update Zowe CLI using the method that you used to install Zowe CLI.

- (Optional) Identify the currently installed version of Zowe CLI on page 107
- (Optional) Identify the currently installed versions of Zowe CLI plug-ins on page 107
- Update Zowe CLI from the online registry on page 107
- Update or revert Zowe CLI to a specific version on page 107
- Update Zowe CLI from a local package on page 107

(Optional) Identify the currently installed version of Zowe CLI

Issue the following command:

```
zowe -V
```

(Optional) Identify the currently installed versions of Zowe CLI plug-ins

Issue the following command:

```
zowe plugins list
```

Update Zowe CLI from the online registry

You can update Zowe CLI to the latest version from the online registry on Windows, Mac, and Linux computers.

Note: The following steps assume that you set the `npm` registries for the `@brightside` scopes as described in Installing Zowe CLI from an online registry on page 106.

Follow these steps:

1. Issue the following command to update Zowe CLI to the most recent `@lts-incremental` version:

   ```
   npm install -g @brightside/core@lts-incremental
   ```

2. Reinstall the plug-ins and update existing plug-ins using the following command:

   ```
   zowe plugins install @brightside/cics@lts-incremental @brightside/db2@lts-incremental
   ```

3. Recreate any user profiles that you created before you updated to the latest version of Zowe CLI.

Update or revert Zowe CLI to a specific version

Optionally, you can update Zowe CLI (or revert) to a known version. The following example illustrates the syntax to update Zowe CLI to version 3.3.1:

```
npm install -g @brightside/core@3.3.1
```

Update Zowe CLI from a local package

To update Zowe CLI from an offline (`.tgz`), local package, uninstall your current package then reinstall from a new package using the Install CA Brightside from a Local Package instructions. For more information, see Uninstalling Zowe CLI on page 107 and Installing Zowe CLI from a local package on page 105.

Important! Recreate any user profiles that you created before the update.

Uninstalling Zowe CLI

You can uninstall Zowe™ CLI from the desktop if you no longer need to use it.

Important\! The uninstall process does not delete the profiles and credentials that you created when using the product from your computer. To delete the profiles from your computer, delete them before you uninstall Zowe CLI.

The following steps describe how to list the profiles that you created, delete the profiles, and uninstall Zowe CLI.

Follow these steps:

1. Open a command line window.

   Note: If you do not want to delete the Zowe CLI profiles from your computer, go to Step 5.

2. List all profiles that you created for a Command Group by issuing the following command:

```
zowe profiles list <profileType>
```

Example:

```
$ zowe profiles list zosmf
The following profiles were found for the module zosmf:
'SMITH-123' (DEFAULT)
smith-123@SMITH-123-W7 C:\Users\SMITH-123
$
```

3. Delete all of the profiles that are listed for the command group by issuing the following command:

Tip: For this command, use the results of the `list` command.

Note: When you issue the `delete` command, it deletes the specified profile and its credentials from the credential vault in your computer's operating system.

```
zowe profiles delete <profileType> <profileName> --force
```

Example:

```
zowe profiles delete zosmf SMITH-123 --force
```

4. Repeat Steps 2 and 3 for all Zowe CLI command groups and profiles.
5. Uninstall Zowe CLI by issuing one of the following commands:
   - If you installed Zowe CLI from the package, issue the following command

     ```
     npm uninstall --global @brightside/core
     ```

   - If you installed Zowe CLI from the online registry, issue the following command:

     ```
     npm uninstall --global brightside
     ```

     The uninstall process removes all Zowe CLI installation directories and files from your computer.
6. Delete the `C:\Users\<user_name>\.brightside` directory on your computer. The directory contains the Zowe CLI log files and other miscellaneous files that were generated when you used the product.

   Tip: Deleting the directory does not harm your computer.
7. If you installed Zowe CLI from the online registry, issue the following command to clear your scoped npm registry:

```
npm config set @brightside:registry
```

Advanced Zowe configuration

Zowe Application Framework configuration

After you install Zowe™, you can optionally configure the Zowe Application Framework as a Mediation Layer client, configure connections for the terminal application plug-ins, or modify the Zowe Application Server and Zowe System Services (ZSS) configuration, as needed.

Configuring the framework as a Mediation Layer client

For simpler Zowe administration and better security, you can install an instance of the Zowe Application Framework as an API Mediation Layer client.

This configuration is simpler to administer because the framework servers are accessible externally through a single port. It is more secure because you can implement stricter browser security policies for accessing cross-origin content.

You must use SSL certificates to configure the Zowe Application Server to communicate with the SSL-enabled Mediation Layer. Those certificates were created during the Zowe installation process, and are located in the `$ROOT_DIR/components/app-server/share/zlux-app-server/defaults/serverConfig` directory.

Enabling the Application Server to register with the Mediation Layer

When you install Zowe v1.8.0 or later, the Application Server automatically registers with the Mediation Layer.

For earlier releases, you must register the Application Server with the Mediation Layer manually. Refer to previous release documentation for more information.

Accessing the Application Server

To access the Application Server through the Mediation Layer, use the Mediation Layer gateway server hostname and port. For example, when accessed directly, this is Zowe Desktop URL: `https://<appservername_port>/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html`

The port number for the Zowe Desktop is the value of the `ZOWE_ZLUX_SERVER_HTTPS_PORT` variable in the `zowe-instance.env` file in the instance directory, see Creating and configuring the Zowe instance directory on page 94.

When accessed through the API Mediation Layer, this is the Zowe Desktop URL: `https://<gwsname_port>/ui/v1/zlux/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html`

The port number for the API Mediation Layer is the value of the `GATEWAY_PORT` variable in the `zowe-instance.env` file in the instance directory.

Setting up terminal application plug-ins

Follow these optional steps to configure the default connection to open for the terminal application plug-ins.

Configuration file

The Zowe App Server and ZSS rely on many required or optional parameters to run, which includes setting up networking, deployment directories, plugin locations, and more.

For convenience, the Zowe Application Server and ZSS read from a JSON file with a common structure. ZSS reads this file directly as a startup argument, while the Zowe Application Server (as defined in the `zlux-server-framework` repository) accepts several parameters. The parameters are intended to be read from a JSON file through an implementer of the server, such as the example in the `zlux-app-server` repository (the `lib/zluxServer.js` file). The file accepts a JSON file that specifies most, if not all, of the parameters needed. Other parameters can be provided through flags, if needed.

For an instance, the configuration file is located at and can be edited at `$INSTANCE_DIR/workspace/app-server/serverConfig/server.json`. The defaults from which that file is generated are located at `$ROOT_DIR/components/app-server/share/zlux-app-server/defaults/serverConfig/server.json`

Note: All examples are based on the *zlux-app-server* repository defaults.

Network configuration

Note: The following attributes are to be defined in the server's JSON configuration file.

The App Server can be accessed over HTTP and/or HTTPS, provided it has been configured for either.

HTTP

To configure the server for HTTP, complete these steps:

1. Define an attribute *http* within the top-level *node* attribute.

2. Define *port* within *http*. Where *port* is an integer parameter for the TCP port on which the server will listen. Specify 80 or a value between 1024-65535.

HTTPS

For HTTPS, specify the following parameters:

1. Define an attribute *https* within the top-level *node* attribute.
2. Define the following within *https*:

- *port*: An integer parameter for the TCP port on which the server will listen. Specify 443 or a value between 1024-65535.
- *certificates*: An array of strings, which are paths to PEM format HTTPS certificate files.
- *keys*: An array of strings, which are paths to PEM format HTTPS key files.
- *pfx*: A string, which is a path to a PFX file which must contain certificates, keys, and optionally Certificate Authorities.
- *certificateAuthorities* (Optional): An array of strings, which are paths to certificate authorities files.
- *certificateRevocationLists* (Optional): An array of strings, which are paths to certificate revocation list (CRL) files.

Note: When using HTTPS, you must specify *pfx*, or both *certificates* and *keys*.

Network example

In the example configuration, both HTTP and HTTPS are specified:

```
  "node": {
    "https": {
      "ipAddresses": ["0.0.0.0"],
      "port": 8544,
      //pfx (string), keys, certificates, certificateAuthorities, and
 certificateRevocationLists are all valid here.
      "keys": ["../defaults/serverConfig/server.key"],
      "certificates": ["../defaults/serverConfig/server.cert"]
    },
    "http": {
      "ipAddresses": ["0.0.0.0"],
      "port": 8543
    }
  }
```

Configuration Directories

When running, the App Server will access the server's settings and read or modify the contents of its resource storage. All of this data is stored within a heirarchy of folders which correspond to scopes:

- Product: The contents of this folder are not meant to be modified, but used as defaults for a product.
- Site: The contents of this folder are intended to be shared across multiple App Server instances, perhaps on a network drive.
- Instance: This folder represents the broadest scope of data within the given App Server instance.
- Group: Multiple users can be associated into one group, so that settings are shared among them.
- User: When authenticated, users have their own settings and storage for the Apps that they use.

These directories dictate where the Configuration Dataservice will store content.

Directories example

```
// All paths relative to zlux-app-server/lib
// In real installations, these values will be configured during the
 install.
  "productDir":"../defaults",
  "siteDir":"/home/myuser/.zowe/workspace/app-server/site",
```

```
  "instanceDir":"/home/myuser/.zowe/workspace/app-server",
  "groupsDir":"/home/myuser/.zowe/workspace/app-server/groups",
  "usersDir":"/home/myuser/.zowe/workspace/app-server/users",
```

Old defaults

Prior to Zowe release 1.8.0, the location of the configuration directories were initialized to be within the `zlux-app-server` folder unless otherwise customized. 1.8.0 has backwards compatibility for the existence of these directories, but they can and should be migrated to take advantage of future enhancements.

Folder	New Location	Old Location	Note
productDir	zlux-app-server/defaults	zlux-app-server/deploy/product	Official installs place zlux-app-server within <ROOT_DIR>/components/app-server/share
siteDir	<INSTANCE_DIR>/workspace/app-server/site	zlux-app-server/deploy/site	INSTANCE_DIR is ~/.zowe if not otherwise defined. Site is placed within instance due to lack of SITE_DIR as of 1.8
instanceDir	<INSTANCE_DIR>/workspace/app-server	zlux-app-server/deploy/instance	
groupsDir	<INSTANCE_DIR>/workspace/app-server/groups	zlux-app-server/deploy/instance/groups	
usersDir	<INSTANCE_DIR>/workspace/app-server/users	zlux-app-server/deploy/instance/users	
pluginsDir	<INSTANCE_DIR>/workspace/app-server/plugins	zlux-app-server/deploy/instance/ZLUX/plugins	Defaults located at zlux-app-server/defaults/plugins, previously at zlux-app-server/plugins

Application plug-in configuration

This topic describes application plug-ins that are defined in advance.

In the configuration file, you can specify a directory that contains JSON files, which tell the server what application plug-in to include and where to find it on disk. The backend of these application plug-ins use the server's plug-in structure, so much of the server-side references to application plug-ins use the term *plug-in*.

To include application plug-ins, define the location of the plug-ins directory in the configuration file, through the top-level attribute **pluginsDir**.

Note: In this example, the directory for these JSON files is the Application Server defaults. However, in an instance of Zowe it is best to provide a folder unique to that instance - usually `$INSTANCE_DIR/workspace/app-server/plugins`.

Plug-ins directory example

```
// All paths relative to zlux-app-server/lib
// In real installations, these values will be configured during the install
 process.
//...
  "pluginsDir":"../defaults/plugins",
```

Logging configuration

For more information, see Logging utility on page 275.

ZSS configuration

Running ZSS requires a JSON configuration file that is similar or the same as the one used for the Zowe Application Server. The attributes that are needed for ZSS, at minimum, are:*productDir*, *siteDir*, *instanceDir*, *groupsDir*, *usersDir*, *pluginsDir* and *agent.http.port*. All of these attributes have the same meaning as described above for the server, but if the Zowe Application Server and ZSS are not run from the same location, then these directories can be different.

Attributes that control ZSS are in the agent object. For example, *agent.http.port* is the TCP port that ZSS will listen on to be contacted by the App Server. Define this in the configuration file as a value between 1024-65535. Similarly, if specified, *agent.http.ipAddresses* will be used to determine which IP addresses the server should bind to. Only the first value of the array is used. It can either be a hostname or an ipv4 address.

Example of the agent body:

```
  "agent": {
    "host": "localhost",
    "http": {
      "ipAddresses": ["127.0.0.1"],
      "port": 8542
    }
  }
```

Connecting App Server to ZSS

When running the App Server, simply specify a few flags to declare which ZSS instance the App Server will proxy ZSS requests to:

- *-h*: Declares the host where ZSS can be found. Use as "-h \<hostname\>"
- *-P*: Declares the port at which ZSS is listening. Use as "-P \<port\>"

Configuring ZSS for HTTPS

To secure ZSS, you can use Application Transparent Transport Layer Security (AT-TLS) to enable Hyper Text Transfer Protocol Secure (HTTPS) on communication with ZSS.

Before you begin, you must have a basic knowledge of RACF and AT-TLS, and you must have Policy Agent configured. For more information on AT-TLS and Policy Agent, see the z/OS Knowledge Center.

To configure ZSS for HTTPS, you create a certificate authority (CA) certificate and a personal certificate, and add the personal certificate to a key ring. Then you define an AT-TLS rule. Then you copy the certificate to the Zowe App Server and specify values in the Zowe App Server configuration file.

By default, the Zowe App Server is the only client that communicates with the ZSS server. In these steps, you configure HTTPS between them by creating a CA certificate and using it to sign a personal certificate. If you want to configure other clients to communicate with ZSS, best practice is to sign their certificates using a recognized certificate authority, such as Symantec. For more information, see documentation for that client.

Note: Bracketed values below (including the brackets) are variables. Replace them with values relevant to your organization.

Creating certificates and a key ring

Use the IBM Resource Access Control Facility (RACF) to create a CA certificate and a personal certificate, and sign the personal certificate with the CA certificate. Then create a key ring with the personal certificate attached.

1. Enter the following command to generate a RACF (CA) certificate:

```
RACDCERT CERTAUTH GENCERT +
  SUBJECTSDN(CN('[common_name]') +
  OU('[organizational_unit]') +
```

```
  O('[organization_name]') +
  L('[locality]') SP('[state_or_province]') C('[country]')) +
  KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN CERTSIGN) +
  WITHLABEL('[ca_label]') +
  NOTAFTER(DATE([xxxx/xx/xx])) +
  SIZE(2048)
```

Note: `[common_name]` must be the ZSS server host name.

1. Enter the follow command to generate a RACF personal certificate signed by the CA certificate:

```
RACDCERT ID('[cert_owner]') GENCERT +
  SUBJECTSDN(CN('[common_name]') +
  OU('[organizational_unit]') +
  O('[organization_name]') +
  L('[locality]') SP('[state_or_province]') C('[country]')) +
  KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN CERTSIGN) +
  WITHLABEL('[personal_label]') +
  NOTAFTER(DATE([xxxx/xx/xx])) +
  SIZE(2048) +
  SIGNWITH(CERTAUTH LABEL('[ca_label]'))
```

1. Enter the following command to create a RACF key ring and connect the personal certificate to the key ring:

```
RACDCERT ID([cert_owner]) ADDRING([ring_name])
RACDCERT CONNECT(ID([cert_owner]) LABEL('[cert_label]') RING([ring_name]))
```

1. Enter the following command to refresh the DIGTRING and DIGTCERT classes to activate your changes:

```
SETROPTS RACLIST(DIGTRING,DIGTCERT) REFRESH
```

1. Enter the following command to verify your changes:

```
RACDCERT LISTRING([ring_name]) ID([cert_owner])
```

1. Enter the following command to export the RACF CA certificate to a dataset:

```
RACDCERT EXPORT(LABEL('[ca_label]')) CERTAUTH DSN('[output_dataset_name]')
 FORMAT(CERTB64)
```

Defining the AT-TLS rule

To define the AT-TLS rule, use the sample below to specify values in your AT-TLS Policy Agent Configuration file:

```
TTLSRule                          ATTLS1~ZSS
{
  LocalAddr                       All
  RemoteAddr                      All
  LocalPortRange                  [zss_port]
  Jobname                         *
  Userid                          *
  Direction                       Inbound
  Priority                        255
  TTLSGroupActionRef              gAct1~ZSS
  TTLSEnvironmentActionRef        eAct1~ZSS
  TTLSConnectionActionRef         cAct1~ZSS
}
TTLSGroupAction                   gAct1~ZSS
{
  TTLSEnabled                     On
  Trace                           1
}
```

```
TTLSEnvironmentAction              eAct1~ZSS
{
  HandshakeRole                    Server
  EnvironmentUserInstance          0
  TTLSKeyringParmsRef              key~ZSS
  Trace                            1
}
TTLSConnectionAction               cAct1~ZSS
{
  HandshakeRole                    Server
  TTLSCipherParmsRef               cipherZSS
  TTLSConnectionAdvancedParmsRef   cAdv1~ZSS
  Trace                            1
}
TTLSConnectionAdvancedParms        cAdv1~ZSS
{
  SSLv3                            Off
  TLSv1                            Off
  TLSv1.1                          Off
  TLSv1.2                          On
  CertificateLabel                 [personal_label]
}
TTLSKeyringParms                   key~ZSS
{
  Keyring                          [ring_name]
}
TTLSCipherParms                    cipher~ZSS
{
  V3CipherSuites                   TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
  V3CipherSuites                   TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
  V3CipherSuites                   TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
  V3CipherSuites                   TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
  V3CipherSuites                   TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
  V3CipherSuites                   TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
  V3CipherSuites                   TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
  V3CipherSuites                   TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
}
```

Configuring the Zowe App Server for HTTPS communication with ZSS

Copy the CA certificate to the ZSS server. Then in the Zowe App Server configuration file, specify the location of the certificate, and add a parameter to specify that ZSS uses AT-TLS.

1. Enter the following command to copy the CA certificate to the correct location in UNIX System Services (USS):

```
cp "//'[output_dataset_name]'" 'zlux-app-server/deploy/instance/ZLUX/
serverConfig/[ca_cert]'
```

1. In the `[INSTANCE_DIR]/workspace/app-server/serverConfig` directory, open the `server.json` file.
2. In the **node.https.certificateAuthorities** object, add the CA certificate file path, for example:

```
"certificateAuthorities": ["[INSTANCE_DIR]/workspace/app-server/
serverConfig/[ca_cert]"]
```

1. In the **agent.http** object add the key-value pair `"attls": true`, for example:

```
"agent": {
  "host": "localhost",
  "http": {
    "ipAddresses": ["127.0.0.1"],
    "port": 8542,
```

```
    "attls": true
  }
}
```

Installing additional ZSS instances

After you install Zowe, you can install and configure additional instances of ZSS on the same z/OS server. You might want to do this to test different ZSS versions.

The following steps assume you have installed a Zowe runtime instance (which includes ZSS), and that you are installing a second runtime instance to install an additional ZSS.

1. To stop the installed Zowe runtime, in SDSF enter the following command:

   ```
   /C ${ZOWE_PREFIX}${ZOWE_INSTANCE}SV
   ```

   Where ZOWE_PREFIX and ZOWE_INSTANCE are specified in your configuration (and default to ZWE and 1)
2. Install a new Zowe runtime by following steps in Installing Zowe on z/OS.

   Note: In the `zowe-install.yaml` configuration file, specify ports that are not used by the first Zowe runtime.
3. To restart the first Zowe runtime, in SDSF enter the following command:

   ```
   /S ZWESVSTC,SRVRPATH='$ZOWE_ROOT_DIR'
   ```

   Where `'$ZOWE_ROOT_DIR'` is the first Zowe runtime root directory. By default the command starts the most recently installed runtime unless you specify the root directory of the runtime that you want to start.
4. To specify a name for the new ZSS instance, follow these steps:

   a. Copy the PROCLIB member JCL named ZWESISTC that was installed with the new runtime.
   b. Rename the copy to uniquely identify it as the JCL that starts the new ZSS, for example ZWESIS02.
   c. Edit the JCL, and in the `NAME` parameter specify a unique name for the cross-memory server, for example:

      ```
      //ZWESIS02  PROC NAME='ZWESIS_MYSRV',MEM=00,RGN=0M
      ```

      Where `ZWESIS_MYSRV` is the unique name of the new ZSS.
5. To start the new ZSS, in SDSF enter the following command:

   ```
   /S ZWESIS02
   ```
6. Make sure that the TSO user ID that runs the first ZSS started task also runs the new ZSS started task. The default ID is IZUSVR.
7. In the new ZSS `server.json` configuration file, add a `"privilegedServerName"` parameter and specify the new ZSS name, for example:

   ```
   "productDir":"../defaults",
   // All paths relative to zlux-app-server/bin
   // In real installations, these values will be configured during the
    install.
   "productDir":"../defaults",
   "siteDir":"../deploy/site",
   "instanceDir":"../deploy/instance",
   "groupsDir":"../deploy/instance/groups",
   "usersDir":"../deploy/instance/users",
   "pluginsDir":"../defaults/plugins",
   "privilegedServerName":"ZWESIS_MYSRV",
   "dataserviceAuthentication": { ... }
   ```

   Note: The instance location of `server.json` is `$INSTANCE_DIR/workspace/app-server/serverConfig/server.json`, and the defaults are stored in `$ROOT_DIR/components/app-server/share/zlux-app-server/defaults/serverConfig/server.json`

8. To start the new Zowe runtime, in SDSF enter the following command:

```
/S ZWESVSTC,INSTANCE='$ZOWE_INSTANCE_DIR'
```

9. To verify that the new cross-memory server is being used, check for the following messages in the `ZWESVSTC` server job log:

```
ZIS status - Ok (name='ZWESIS_MYSRV ', cmsRC=0, description='Ok',
clientVersion=2)
```

Controlling access to applications

You can control which applications are accessible (visible) to all Zowe desktop users, and which are accessible only to individual users. For example, you can make an application that is under development only visible to the team working on it.

You control access by editing JSON files that list the apps. One file lists the apps all users can see, and you can create a file for each user. When a user logs into the desktop, Zowe determines the apps that user can see by concatenating their list with the all users list.

You can also control access to the JSON files. The files are accessible directly on the file system, and since they are within the configuration dataservice directories, they are also accessible via REST API. We recommend that only Zowe administrators be allowed to access the file system locations, and you control that by setting the directories and their contents to have file permissions on z/OS that only allow the Zowe admin group read & write access. You control who can read and edit the JSON files through the REST API by controlling who can Creating authorization profiles on page 118 URLs that serve the JSON files.

Controlling application access for all users

1. Open the Zowe Application Server configuration JSON file. By default, the file is in the following location:

```
$ROOT_DIR/components/app-server/share/zlux-app-server/defaults/
serverConfig/server.json
```

2. To enable RBAC, in the `dataserviceAuthentication` object add the object: `"rbac": true`
3. Navigate to the following location:

```
$ROOT_DIR/components/app-server/share/zlux-app-server/defaults/ZLUX/
pluginStorage/org.zowe.zlux.bootstrap/plugins
```

4. Copy the `allowedPlugins.json` file and paste it in the following location:

```
.zowe/workspace/app-server/ZLUX/pluginStorage/org.zowe.zlux.bootstrap
```

5. Open the copied `allowedPlugins.json` file and perform either of the following steps:
   - To an application unavailable, delete it from the list of objects.
   - To make an application available, copy an existing plugin object and specify the application's values in the new object. Identifier and version attributes are required.
6. Stopping the ZWESVSTC PROC on page 102.

Controlling application access for individual users

1. Open the Zowe Application Server configuration JSON file. By default, the file is in the following location:

```
$ROOT_DIR/components/app-server/share/zlux-app-server/defaults/
serverConfig/server.json
```

2. To enable RBAC, in the `dataserviceAuthentication` object add the object: `"rbac": true`

3. In the user's ID directory path, in the `\pluginStorage` directory, create `\org.zowe.zlux.bootstrap\plugins` directories. For example:

```
.zowe\workspace\app-server\users\TS6320\ZLUX\pluginStorage
\org.zowe.zlux.bootstrap\plugins
```

4. In the `/plugins` directory, create an `allowedPlugins.json` file. You can use the default `allowedPlugins.json` file as a template by copying it from the following location:

```
$ROOT_DIR/components/app-server/share/zlux-app-server/defaults/ZLUX/
pluginStorage/org.zowe.zlux.bootstrap/plugins
```

5. Open the `allowedPlugins.json` file and specify applications that user can access. For example:

```
{
  "allowedPlugins": [
    {
      "identifier": "org.zowe.appA",
      "versions": [
        "*"
      ]
    },
    {
      "identifier": "org.zowe.appB",
      "versions": [
        "*"
      ]
    },
}
```

Notes:

- Identifier and version attributes are required.
- When a user logs in to the desktop, Zowe determines which apps they can see by concatenating the list of apps available to all users with the apps available to the individual user.

6. Stopping the ZWESVSTC PROC on page 102.

Controlling access to dataservices

To apply role-based access control (RBAC) to dataservice endpoints, you must enable RBAC for Zowe, and then use a z/OS security product such as RACF to map roles and authorities to the endpoints. After you apply RBAC, Zowe checks authorities before allowing access to the endpoints.

You can apply access control to Zowe endpoints and to your application endpoints. Zowe provides endpoints for a set of configuration dataservices and a set of core dataservices. Applications can use Configuration Dataservice on page 261 to store and their own configuration and other data. Administrators can use core endpoints to get status information from the Application Framework and ZSS servers. Any dataservice added as part of an application plugin is a service dataservice.

Defining the RACF ZOWE class

If you use RACF security, take the following steps define the ZOWE class to the CDT class:

1. Make sure that the CDT class is active and RACLISTed.
2. In TSO, issue the following command:

```
RDEFINE CDT ZOWE UACC(NONE)
CDTINFO(
    DEFAULTUACC(NONE)
    FIRST(ALPHA) OTHER(ALPHA,NATIONAL,NUMERIC,SPECIAL)
    MAXLENGTH(246)
    POSIT(607)
```

```
    RACLIST(DISALLOWED))
```

If you receive the following message, ignore it:

```
"Warning: The POSIT value is not within the recommended ranges for
 installation use. The valid ranges are 19-56 and 128-527."
```

3. In TSO, issue the following command to refresh the CDT class:

```
SETROPTS RACLIST(CDT) REFRESH
```

4. In TSO, issue the following command to activate the ZOWE class:

```
SETROPTS CLASSACT(ZOWE)
```

For more information RACF security administration, see the IBM Knowledge Center at https://www.ibm.com/support/knowledgecenter/.

Enabling RBAC

By default, RBAC is disabled and all authenticated Zowe users can access all dataservices. To enable RBAC, follow these steps:

1. Open the Zowe Application Server configuration JSON file. In the a server instance, the configuration file is `$INSTANCE_DIR/workspace/app-server/serverConfig/server.json`.
2. In the `dataserviceAuthentication` object, add `"rbac": true`.

Creating authorization profiles

For users to access endpoints after you enable RBAC, in the ZOWE class you must create System Authorization Facility (SAF) profiles for each endpoint and give users READ access to those profiles.

Endpoints are identified by URIs in the following format:

`/<product>/plugins/<plugin_id>/services/<service>/<version>/<path>`

For example:

`/ZLUX/plugins/org.zowe.foo/services/baz/_current/users/fred`

Where the path is `/users/fred`.

SAF profiles have the following format:

`<product>.<instance_id>.<service>.<pluginid_with_underscores>.<service>.<HTTP_method>.<u`

For example, to issue a POST request to the dataservice endpoint documented above, users must have READ access to the following profile:

`ZLUX.DEFAULT.SVC.ORG_ZOWE_FOO.BAZ.POST.USERS.FRED`

For configuration dataservice endpoint profiles use the service code `CFG`. For core dataservice endpoints use `COR`. For all other dataservice endpoints use `SVC`.

Creating generic authorization profiles

Some endpoints can generate an unlimited number of URIs. For example, an endpoint that performs a DELETE action on any file would generate a different URI for each file, and users can create an unlimited number of files. To apply RBAC to this type of endpoint you must create a generic profile, for example:

`ZLUX.DEFAULT.COR.ORG_ZOWE_FOO.BAZ.DELETE.**`

You can create generic profile names using wildcards, such as asterisks (*). For information on generic profile naming, see IBM documentation.

Configuring basic authorization

The following are recommended for basic authorization:

- To give administrators access to everything in Zowe, create the following profile and give them UPDATE access to it: `ZLUX.**`
- To give non-administrators basic access to the site and product, create the following profile and give them READ access to it: `ZLUX.*.ORG_ZOWE_*`
- To prevent non-administrators from configuring endpoints at the product and instance levels, create the following profile and do not give them access to it: `ZLUX.DEFAULT.CFG.**`
- To give non-administrators all access to user, create the following profile and give them UPDATE access to it: `ZLUX.DEFAULT.CFG.*.*.USER.**`

Endpoint URL length limitations

SAF profiles cannot contain more than 246 characters. If the path section of an endpoint URL is long enough that the profile name exceeds the limit, the path is trimmed to only include elements that do not exceed the limit. To avoid this issue, we recommend that appliction developers maintain relatively short endpoint URL paths.

For information on endpoint URLs, see Dataservice endpoint URL lengths and RBAC

Enabling tracing

To obtain more information about how a server is working, you can enable tracing within the `server.json` file.

For example:

```
"logLevels": {
  "_zsf.routing": 0,
  "_zsf.install": 0,
  "_zss.traceLevel": 0,
  "_zss.fileTrace": 1
}
```

Specify the following settings inside the **logLevels** object.

All settings are optional.

Zowe Application Server tracing

To determine how the Zowe Application Server (`zlux-app-server`) is working, you can assign a logging level to one or more of the pre-defined logger names in the `server.json` file.

The log prefix for the Zowe Application Server is **_zsf**, which is used by the server framework. (Applications and plug-ins that are attached to the server do not use the **_zsf** prefix.)

The following are the logger names that you can specify:

_zsf.bootstrap Logging that pertains to the startup of the server.

_zsf.auth Logging for network calls that must be checked for authentication and authorization purposes.

_zsf.static Logging of the serving of static files (such as images) from an application's `/web` folder.

_zsf.child Logging of child processes, if any.

_zsf.utils Logging for miscellaneous utilities that the server relies upon.

_zsf.proxy Logging for proxies that are set up in the server.

_zsf.install Logging for the installation of plug-ins.

_zsf.apiml Logging for communication with the api mediation layer.

_zsf.routing Logging for dispatching network requests to plug-in dataservices.

_zsf.network Logging for the HTTPS server status (connection, ports, IP, and so on)

Log levels

The log levels are:

- SEVERE = 0,
- WARNING = 1,
- INFO = 2,
- FINE = 3,
- FINER = 4,
- FINEST = 5

FINE, FINER, and FINEST are log levels for debugging, with increasing verbosity.

Enabling tracing for ZSS

To increase logging for ZSS, you can assign a logging level (an integer value greater than zero) to one or more of the pre-defined logger names in the `server.json` file.

A higher value specifies greater verbosity.

The log prefix for ZSS is **_zss**. The following are the logger names that you can specify:

_zss.traceLevel: Controls general server logging verbosity.

_zss.fileTrace: Logs file serving behavior (if file serving is enabled).

_zss.socketTrace: Logs general TCP Socket behavior.

_zss.httpParseTrace: Logs parsing of HTTP messages.

_zss.httpDispatchTrace: Logs dispatching of HTTP messages to dataservices.

_zss.httpHeadersTrace: Logs parsing and setting of HTTP headers.

_zss.httpSocketTrace: Logs TCP socket behavior for HTTP.

_zss.httpCloseConversationTrace: Logs HTTP behavior for when an HTTP conversation ends.

_zss.httpAuthTrace: Logs behavior for session security.

When you are finished specifying the settings, save the `server.json` file.

Zowe Application Framework logging

The Zowe Application Framework log files contain processing messages and statistics. The log files are generated in the following default locations:

- Zowe Application Server: `$INSTANCE_DIR/logs/appServer-yyyy-mm-dd-hh-mm.log`
- ZSS: `$INSTANCE_DIR/logs/zssServer-yyyy-mm-dd-hh-mm.log`

The logs are timestamped in the format yyyy-mm-dd-hh-mm and older logs are deleted when a new log is created at server startup.

Controlling the logging location

The log information is written to a file and to the screen. (On Windows, logs are written to a file only.)

ZLUX_NODE_LOG_DIR and ZSS_LOG_DIR environment variables

To control where the information is logged, use the environment variable *ZLUX_NODE_LOG_DIR*, for the Zowe Application Server, and *ZSS_LOG_DIR*, for ZSS. While these variables are intended to specify a directory, if you specify a location that is a file name, Zowe will write the logs to the specified file instead (for example: `/dev/null` to disable logging).

When you specify the environment variables *ZLUX_NODE_LOG_DIR* and *ZSS_LOG_DIR* and you specify directories rather than files, Zowe will timestamp the logs and delete the older logs that exceed the *ZLUX_NODE_LOGS_TO_KEEP* threshold.

ZLUX_NODE_LOG_FILE and ZSS_LOG_FILE environment variables

If you set the log file name for the Zowe Application Server by setting the *ZLUX_NODE_LOG_FILE* environment variable, or if you set the log file for ZSS by setting the *ZSS_LOG_FILE* environment variable, there will only be one log file, and it will be overwritten each time the server is launched.

Note: When you set the *ZLUX_NODE_LOG_FILE* or *ZSS_LOG_FILE* environment variables, Zowe will not override the log names, set a timestamp, or delete the logs.

If the directory or file cannot be created, the server will run (but it might not perform logging properly).

Retaining logs

By default, the last five logs are retained. To specify a different number of logs to retain, set *ZLUX_NODE_LOGS_TO_KEEP* (Zowe Application Server logs) or *ZSS_LOGS_TO_KEEP* (ZSS logs) to the number of logs that you want to keep. For example, if you set *ZLUX_NODE_LOGS_TO_KEEP* to 10, when the eleventh log is created, the first log is deleted.

Administering the servers and plugins using an API

You can use a REST API to retrieve and edit Zowe Application Server and ZSS server configuration values, and list, add, update, and delete plugins. If an administrator has configured Zowe to use RBAC, they must authorize you to access the endpoints.

The API returns the following information in a JSON response:

API	Description
/server (GET)	Returns a list of accessible server endpoints for the Zowe Application Server.
/server/config (GET)	Returns the Zowe Application Server configuration from the `zluxserver.json` file.
/server/log (GET)	Returns the contents of the Zowe Application Server log file.
/server/loglevels (GET)	Returns the verbosity levels set in the Zowe Application Server logger.
/server/environment (GET)	Returns Zowe Application Server environment information, such as the operating system version, node server version, and process ID.
/server/reload (GET)	Reloads the Zowe Application Server. Only available in cluster mode.
/server/agent (GET)	Returns a list of accessible server endpoints for the ZSS server.
/server/agent/config (GET)	Returns the ZSS server configuration from the `zluxserver.json` file.
/server/agent/log (GET)	Returns the contents of the ZSS log file.
/server/agent/loglevels (GET)	Returns the verbosity levels of the ZSS logger.
/server/agent/environment (GET)	Returns ZSS environment information.
/server/config/:attrib (POST)	Specify values for server configuration attributes in the `zluxserver.json` file. You can change a subset of configuration values.
/server/logLevels/name/:componentName/level/:level (POST)	Specify the logger that you are using and a verbosity level.
/plugins (GET)	Returns a list of all plugins and their dataservices.

API	Description
/plugins (PUT)	Adds a new plugin or upgrades an existing plugin. Only available in cluster mode.
/plugins/:id (DELETE)	Deletes a plugin. Only available in cluster mode.

Swagger API documentation is provided in the `$ROOT_DIR/components/app-server/share/zlux-app-server/doc/swagger/server-plugins-api.yaml` file. To see it in HTML format, you can paste the contents into the Swagger editor at https://editor.swagger.io/.

Note: The "agent" end points interact with the agent specified in the `server.json` file. By default this is ZSS.

Configuring Zowe CLI

This section explains how to define and verify your connection to the mainframe through Zowe™ CLI. You can also configure CLI settings, such as the level of detail produced in logs and the location of the home directory on your computer.

Note: The configuration for the CLI is stored on your computer in a directory such as C:\Users\user01\.zowe. The configuration includes log files, your profile information, and CLI plug-ins that are installed. When you troubleshoot an issue with the CLI, the log files in the imperative and zowe folders contain valuable information.

- Defining Zowe CLI connection details on page 122
- Testing Zowe CLI connection to z/OSMF
- Setting Zowe CLI log levels on page 127
- Setting the Zowe CLI home directory on page 127

Defining Zowe CLI connection details

Zowe CLI has a *command option order of precedence* that lets you define arguments and options for commands in multiple ways (command-line, environment variables, and profiles). This provides flexibility when you issue commands and write automation scripts. This topic explains order of precedence and different methods for specifying your mainframe connection details.

- Understanding command option order of precedence on page 122
- Creating Zowe CLI profiles on page 123
- Defining Environment Variables on page 124
- Integrating with API Mediation Layer on page 125

Understanding command option order of precedence

Before you issue commands, it is helpful to understand the command option order of precedence. The following is the order in which Zowe CLI *searches for* your command arguments and options when you issue a command:

1. Arguments and options that you specify directly on the command line.
2. Environment variables that you define in the computer's operating system. For more information, see Defining Environment Variables on page 124
3. User profiles that you create.
4. The default value for the argument or option.

The affect of the order is that if you omit an argument/option from the command line, Zowe CLI searches for an environment variable that contains a value that you defined for the argument/option. If Zowe CLI does not find a value for the argument/option in an environment variable, Zowe CLI searches your user profiles for the value that you defined for the option/argument. If Zowe CLI does not find a value for the argument/option in your profiles, Zowe CLI executes the command using the default value for the argument/option.

Note: If a required option or argument value is not located, you receive a syntax error message that states `Missing Positional Argument` or `Missing Option.`

Creating Zowe CLI profiles

Profiles let you store configuration details for use on multiple commands. You can create a profile that contains your username, password, and connection details for a particular mainframe system, then reuse that profile to avoid typing it again on every command. Switch between profiles to quickly target different mainframe subsystems.

Notes:

- Profile values are stored on your computer in plaintext in the `C:\Users\<yourUsername>\.zowe\profiles` folder.
- Profiles are **not** required to use the CLI. You can choose to specify all connection details in options on every command.
- For information about securely connecting to the server when you issue commands, see Certificate security on page 126.

Displaying profiles help

To learn about the options available for creating `zosmf` profiles, issue the following command:

```
zowe profiles create zosmf-profile --help
```

Creating and Using a profile

Create a profile, then use the profile when you issue a command. For example, substitute your connection details and issue the following command to create a profile with the name `myprofile123`:

```
zowe profiles create zosmf-profile myprofile123 --host host123 --port
 port123 --user ibmuser --password pass123
```

Issue the following command to list all data sets under the username ibmuser on the system specified in `myprofile123`:

```
zowe zos-files list data-set "ibmuser.*" --zosmf-profile myprofile123
```

After you create a profile, you can verify that it can communicate with z/OSMF. For more information, see Testing Connection to z/OSMF.

Creating a profile that accesses API Mediation Layer

You can create profiles that access either an exposed API or API Mediation Layer (API ML) in the following ways:

- When you create a profile, specify the host and port of the API that you want to access. When you only provide the host and port configuration, Zowe CLI connects to the exposed endpoints of a specific API.
- When you create a profile, specify the host, port, and the base path of API ML instance that you want to access. Using the base path to API ML, Zowe CLI routes your requests to an appropriate instance of the API based on the system load and the available instances of the API.

For more information, see Integrating with API Mediation Layer on page 125.

Example:

The following example illustrates the command to create a profile that connects to z/OSMF through API ML with the base path `my/api/layer`:

```
zowe profiles create zosmf myprofile -H <myhost> -P <myport> -u <myuser> --
pw <mypass> --base-path <my/api/layer>
```

After you create a profile, verify that it can communicate with z/OSMF. For more information, see Testing Zowe CLI connection to z/OSMF.

Defining Environment Variables

You can define environment variables in your environment to execute commands more efficiently. You can store a value, such as your password, in an environment variable, then issue commands without specifying your password every time. The term environment refers to your operating system, but it can also refer to an automation server, such as Jenkins or a Docker container. In this section we explain how to transform arguments and options from Zowe CLI commands into environment variables and define them with a value.

- **Assigning an environment variable for a value that is commonly used.**

  For example, you might want to specify your mainframe user name as an environment variable on your computer. When you issue a command and omit the `--username` argument, Zowe CLI automatically uses the value that you defined in the environment variable. You can now issue a command or create any profile type without specifying your user name repeatedly.
- **Overriding a value that is used in existing profiles.**

  For example, you might want to override a value that you previously set on multiple profiles to avoid recreating each profile.This reduces the number of profiles that you need to maintain and lets you avoid specifying every option on command line for one-off commands.
- **Specifying environment variables in a Jenkins environment (or other automation server) to store credentials securely.**

  You can set values in Jenkins environment variables for use in scripts that run in your CI/CD pipeline. You can define Jenkins environment variables in the same manner that you can on your computer. You can also define sensitive information in the Jenkins secure credential store. For example, you might need to define your mainframe password in the secure credential store so that it is not available in plain text.

Transforming arguments/options to environment variable format

Transform the option/argument into the correct format for a Zowe CLI environment variable, then define values to the new variable. The following rules apply to this transformation:

- Prefix environment variables with `ZOWE_OPT_`
- Convert lowercase letters in arguments/options to uppercase letters
- Convert hyphens in arguments/options to underscores

Tip: See your operating system documentation for information about how to set and get environment variables. The procedure for setting environment variables varies between Windows, Mac, and various versions of Linux operating systems.

Examples:

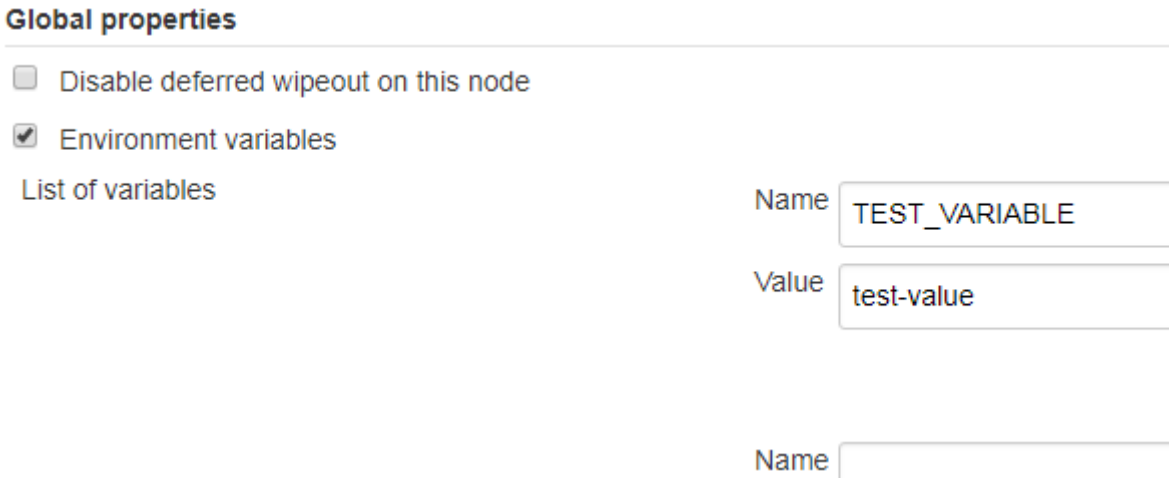
The following table shows command line options that you might want to transform and the resulting environment variable to which you should define the value. Use the appropriate procedure for for your operating system to define the variables.

Command Option	Environment Variable	Use Case
`--user`	`ZOWE_OPT_USER`	Define your mainframe user name to an environment variable to avoid specifying it on all commands or profiles.
`--reject-unauthorized`	`ZOWE_OPT_REJECT_UNAUTHORIZED`	Define a value of `true` to the `--reject-unathorized` flag when you always require the flag and do not want to specify it on all commands or profiles.

Setting environment variables in an automation server

You can use environment variables in an automation server, such as Jenkins, to write more efficient scripts and make use of secure credential storage.

You can either set environment variables using the `SET` command within your scripts, or navigate to **Manage Jenkins \> Configure System \> Global Properties** and define an environment variable in the Jenkins GUI. For example:



Using secure credential storage

Automation tools such as Jenkins automation server usually provide a mechanism for securely storing configuration (for example, credentials). In Jenkins, you can use `withCredentials` to expose credentials as an environment variable (ENV) or Groovy variable.

Note: For more information about using this feature in Jenkins, see Credentials Binding Plugin in the Jenkins documentation.

Integrating with API Mediation Layer

The API Mediation Layer provides a single point of access to a defined set of microservices. The API Mediation Layer provides cloud-like features such as high-availability, scalability, dynamic API discovery, consistent security, a single sign-on experience, and API documentation.

When Zowe CLI executes commands that connect to a service through the API Mediation Layer, the layer routes the command execution requests to an appropriate instance of the API. The routing path is based on the system load and available instances of the API.

Use the `--base-path` option on commands to let all of your Zowe CLI core command groups (excludes plug-in groups) access REST APIs through an API Mediation Layer. To access API Mediation Layers, you specify the base path, or URL, to the API gateway as you execute your commands. Optionally, you can define the base path URL as an environment variable or in a profile that you create.

Examples:

The following example illustrates the base path for a REST request that is not connecting through an API Mediation Layer to one system where an instance of z/OSMF is running:

```
https://mymainframehost:port/zosmf/restjobs/jobs
```

The following example illustrates the base path (named `api/v1/zosmf1)` for a REST request to an API mediation layer:

```
https://myapilayerhost:port/api/v1/zosmf1/zosmf/restjobs/jobs
```

The following example illustrates the command to verify that you can connect to z/OSMF through an API Mediation Layer that contains the base path `my/api/layer`:

```
zowe zosmf check status -H <myhost> -P <myport> -u <myuser> --pw <mypass> --
base-path <my/api/layer>
```

More Information:

- Zowe overview on page 8
- Creating a profile that accesses API Mediation Layer on page 123

Testing Zowe CLI connection to z/OSMF

You can issue a command at any time to receive diagnostic information from the server and confirm that Zowe CLI can communicate with z/OSMF or other mainframe APIs.

Important! By default, the server certificate is verified against a list of Certificate Authorities (CAs) trusted by Mozilla. This handshake ensures that the CLI can trust the server. You can append the flag `--ru false` to any of the following commands to bypass the certificate verification against CAs. If you use the `--ru false` flag, ensure that you understand the potential security risks of bypassing the certificate requirement at your site. For the most secure environment, system administrators configure a server keyring with a server certificate signed by a Certificate Authority (CA). For more information, see Certificate security on page 126.

Without a Profile

Verify that your CLI instance can communicate with z/OSMF.

```
zowe zosmf check status --host <host> --port <port> --user <username> --pass
 <password>
```

Default profile

After you Creating Zowe CLI profiles on page 123, verify that you can use your *default profile* to communicate with z/OSMF:

```
zowe zosmf check status
```

Specific profile

After you Creating Zowe CLI profiles on page 123, verify that you can use *a specific profile* to communicate with z/OSMF:

```
zowe zosmf check status --zosmf-profile <profile_name>
```

The commands return a success or failure message and display information about your z/OSMF server, such as the z/OSMF version number. Report any failure to your systems administrator and use the information for diagnostic purposes.

Certificate security

Certificates authorize communication between a server and client, such as z/OSMF and Zowe CLI. The client CLI must "trust" the server to successfully issue commands. Use one of the following methods to let the CLI communicate with the server:

- Configure certificates signed by a Certificate Authority (CA) on page 126
- Extend trusted certificates on client on page 127
- Bypass certificate requirement with CLI flag on page 127

Configure certificates signed by a Certificate Authority (CA)

System Administrators can configure the server with a certificate signed by a Certificate Authority (CA) trusted by Mozilla. When a CA trusted by Mozilla exists in the certificate chain, the CLI automatically recognizes the server and authorizes the connection.

Related information:

- Using certificates with z/OS client/server applications in the IBM Knowledge Center.
- Configuring the z/OSMF key ring and certificate in the IBM Knowledge Center.
- Certificate management in Zowe API Mediation Layer

- Mozilla Included CA Certificate List

Extend trusted certificates on client

If your organization uses self-signed certificates in the certificate chain (rather than a CA trusted by Mozilla), you can download the certificate to your computer add it to the local list of trusted certificates. Provide the certificate locally using the `NODE_EXTRA_CERTS` environment variable. Organizations might want to configure all client computers to trust the self-signed certificate.

This blog post outlines the process for using environment variables to trust the self-signed certificate.

Bypass certificate requirement with CLI flag

If you do not have server certificates configured at your site, or you want to trust a known self-signed certificate, you can append the `--reject-unauthorized false` flag to your CLI commands. Setting the `--reject-unauthorized` flag to `false` rejects self-signed certificates and essentially bypasses the certificate requirement.

Important! Understand the security implications of accepting self-signed certificates at your site before you use this command.

Example:

```
zowe zosmf check status --host <host> --port <port> --user <username> --pass
 <password> --reject-unauthorized false
```

Setting Zowe CLI log levels

You can set the log level to adjust the level of detail that is written to log files:

Important\! Setting the log level to TRACE or ALL might result in "sensitive" data being logged. For example, command line arguments will be logged when TRACE is set.

Environment Variable	Description	Values	Default
`ZOWE\_APP\_LOG \_LEVEL`	Zowe CLI logging level	Log4JS log levels (OFF, TRACE, DEBUG, INFO, WARN, ERROR, FATAL)	DEBUG
`ZOWE\_IMPERATIVE \_LOG\_LEVEL`	Imperative CLI Framework logging level	Log4JS log levels (OFF, TRACE, DEBUG, INFO, WARN, ERROR, FATAL)	DEBUG

Setting the Zowe CLI home directory

You can set the location on your computer where Zowe CLI creates the *.zowe* directory, which contains log files, profiles, and plug-ins for the product:

Environment Variable	Description	Values	Default
`ZOWE\_CLI\_HOME`	Zowe CLI home directory location	Any valid path on your computer	Your computer default home directory

Using Zowe

Getting started tutorial

Contents

- Learning objectives on page 128
- Estimated time on page 128
- Prerequisites and assumptions on page 128

- Logging in to the Zowe Desktop on page 129
- Querying JES jobs and viewing related status in JES Explorer on page 130
- Using the 3270 Terminal in the Zowe Desktop to view the job on page 132
- Editing a data set in MVS Explorer on page 142
- Using the Zowe CLI to edit a data set on page 143
- Viewing the data set changes in MVS Explorer on page 145
- Next steps on page 145
  - Go deeper with Zowe on page 145
  - Try the Extending Zowe scenarios on page 145
  - Give feedback on page 145

Learning objectives

This tutorial walks you through the Zowe™ interfaces, including the Zowe Desktop and Zowe CLI, with several simple tasks to help you get familiar with Zowe.

- If you are new to Zowe, start with this tutorial to explore the base Zowe features and functions.
- If you are already familiar with Zowe interfaces and capabilities, you might want to visit the **Extending** section which guides you to extend Zowe by creating your own APIs or applications.
  - Onboarding Overview on page 164
  - Overview on page 237
  - Developing for Zowe CLI on page 224

By the end of the session, you'll know how to:

- Log in to the Zowe Desktop
- Query jobs with filters and view the related status by using the JES Explorer
- View jobs by using the 3270 Terminal in the Zowe Desktop
- View and edit data sets by using the MVS Explorer
- Edit a data set and upload it to the mainframe by using Zowe Command-Line Interface (CLI)

As an introductory scenario, no previous knowledge of Zowe is needed.

Estimated time

This tutorial guides you through the steps in roughly 20 minutes. If you explore other concepts related to this tutorial, it can take longer to complete.

Prerequisites and assumptions

Before you begin, it is assumed that you have already successfully installed Zowe. You are ready to launch Zowe Desktop and Zowe CLI.

For information about how to install Zowe, see Introduction on page 36.

Important!

- In this tutorial, the following parameters are used as an example. Replace them with your own settings when you follow the tutorial in your environment.
  - URL to access the Zowe Desktop: `https://s0w1:8544/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html`
  - Mainframe credentials:
    - Username: `ibmuser`
    - Password: `sys1`
- It is assumed that you perform the tasks in a Windows environment and that you have Visual Studio Code (VS Code) installed.

Logging in to the Zowe Desktop

Access and navigate the Zowe Desktop to view the Zowe applications. In this tutorial, you will use the Firefox browser to log in to the Zowe Desktop.

The URL to access the Zowe Desktop is `https://myhost:httpsPort/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html` in your own environment, where:

- *myHost* is the host on which you are running the Zowe Application Server.
- *httpsPort* is the value that was assigned to *node.https.port* in `zluxserver.json`. For example, if you run the Zowe Application Server on host *myhost* and the value that is assigned to *node.https.port* in `zluxserver.json` is 12345, you would specify `https://myhost:12345/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html`.

Follow these steps:

1. In the address field, enter the URL to access the Zowe Desktop. In this tutorial, the following URL is used as an example:

   `https://s0w1:8544/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html`

2. On the login page of the Zowe Desktop, enter your mainframe credentials. In this tutorial, the following ID is used as an example:
   - Username: ibmuser
   - Password: sys1
3. Press Enter.

Upon authentication of your user name and password, the Zowe Desktop opens. Several applications are pinned to the taskbar. Click the Start menu to see a list of applications that are installed by default. You can pin other applications to the taskbar by right-clicking the application icon and selecting **Pin to taskbar**.

Next, you will use the JES Explorer application to query the jobs with filters and view the related status.

Querying JES jobs and viewing related status in JES Explorer

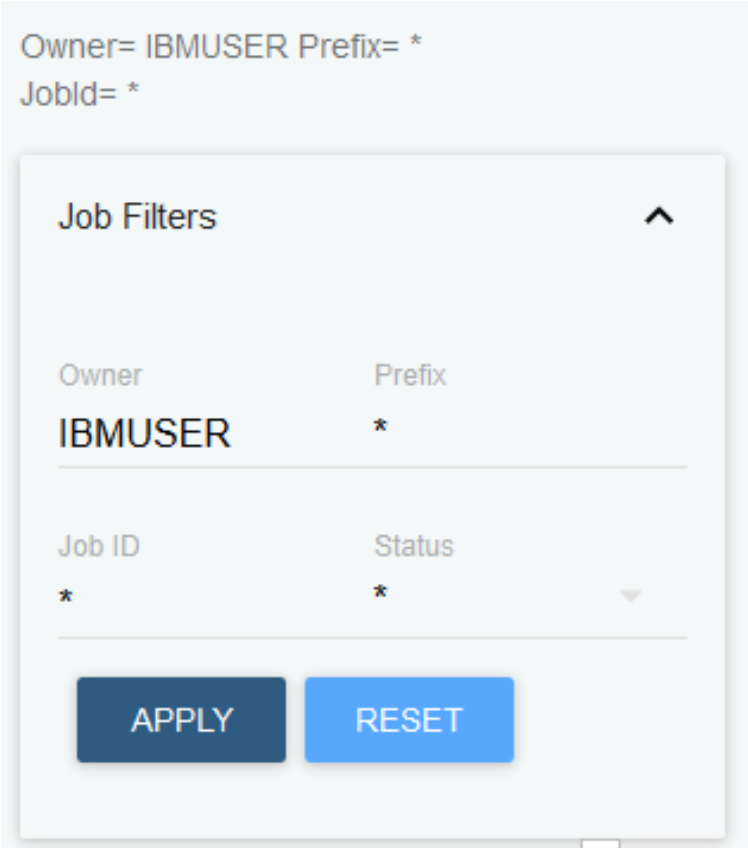
Use the Job Entry Subsystem (JES) Explorer to query JES jobs with filters and view the related status.

Follow these steps:

1. Click the Start menu in the Zowe Desktop.

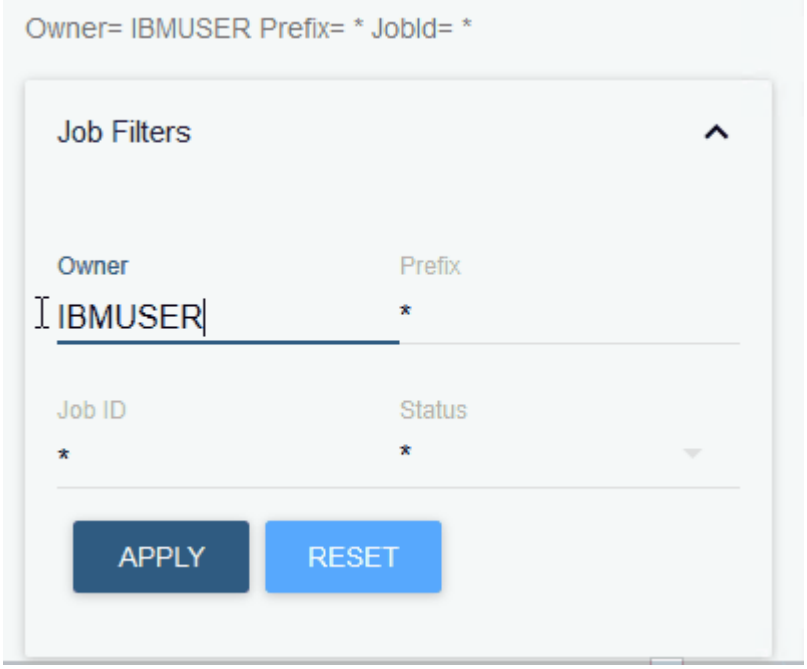
2. Scroll down to find the JES Explorer icon and click to open it. The JES Explorer is displayed. If prompted to provide credentials for authentication, enter your mainframe credentials.

3. Click the **Job Filters** column to expand the filter criteria. You can filter jobs on various criteria by Owner, Prefix, Job ID, and Status. By default, the jobs are filtered by Owner. In this tutorial, the example owner is `IBMUSER`.



4. To query the jobs starting with `SDSF` and in an active status, clear the field of **Owner**, then enter `SDSF*` in the **Prefix** field and select **ACTIVE** from the **Status** drop-down list, and click **APPLY**.

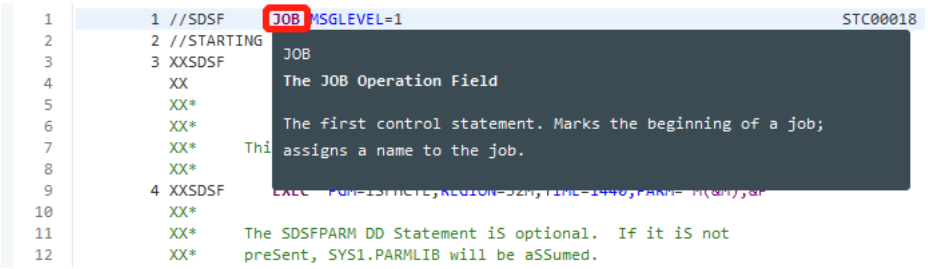
   Note: Wildcard is supported. Valid wildcard characters are asterisk (*), percent sign (%), and question mark (?).



5. From the job filter results, click the job named **SDSF**. The data sets for this job are listed.

6. Click **JESJCL** to open the JESJCL data set. The contents of this data set are displayed. You can also select other data sets to view their contents.

   Tip: You can hover over the text in purple color to display a hover help window.



You used the JES Explorer to query the JES jobs with filters and viewed the related steps, files, and status.

Close the JES Explorer window. Next, you'll use the TN3270 application plug-in in the Zowe Desktop to view the same job that you viewed in this task.

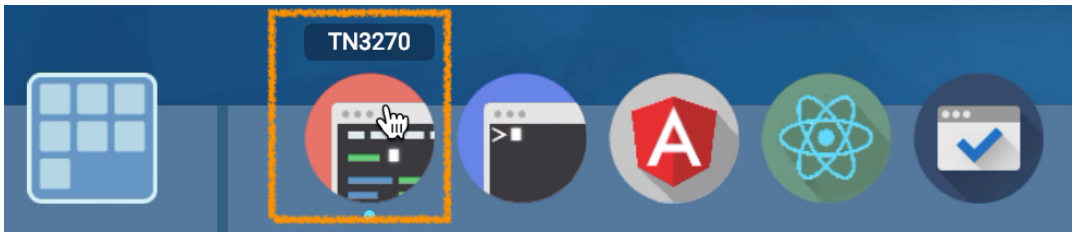
Using the 3270 Terminal in the Zowe Desktop to view the job

Use the 3270 Terminal application plug-in to view the same job that you filtered out in the previous task.

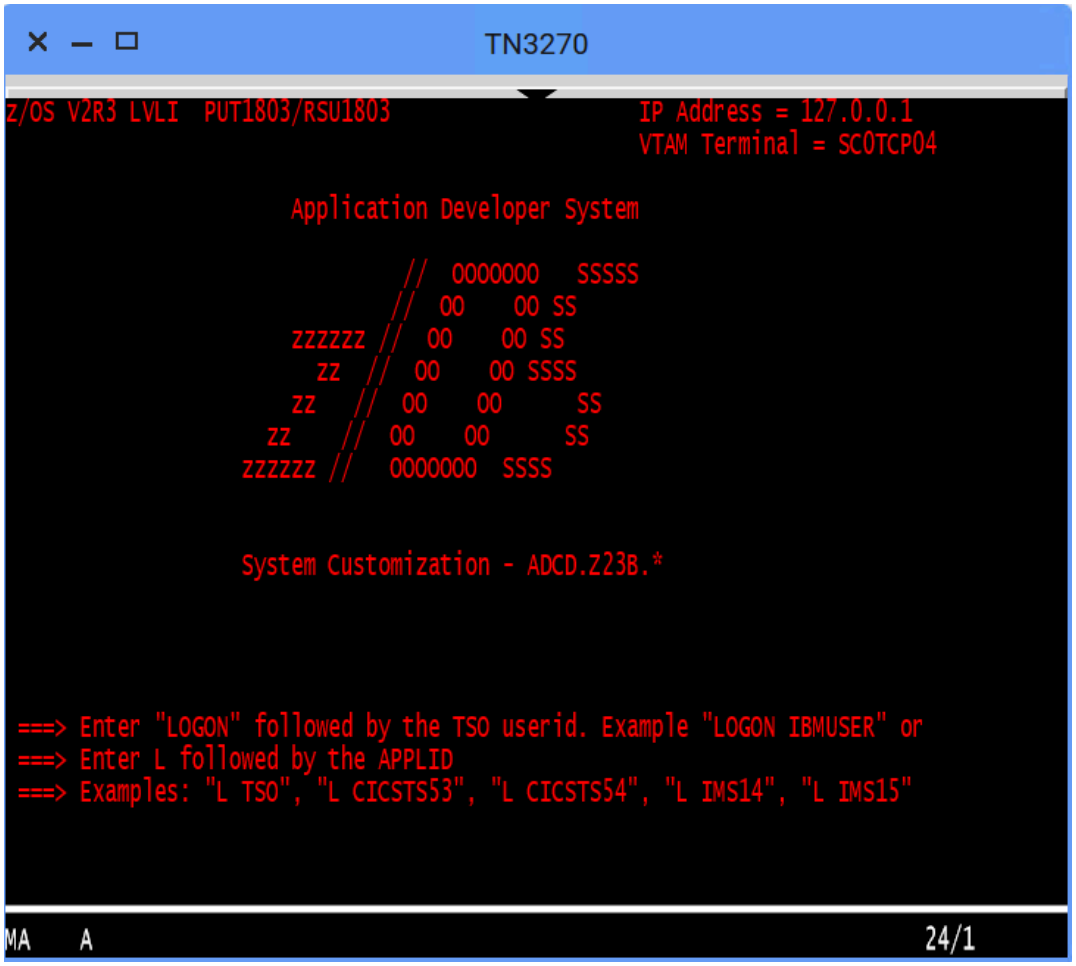
Zowe not only provides new, modern applications to interact with z/OS®, but it also integrates the traditional 3270 terminal interface that you may also be familiar with. The 3270 Terminal application plug-in provides a basic, emulated 3270 terminal connection to the mainframe via the Zowe Application Server.

Follow these steps:

1. From the taskbar at the bottom of the Zowe Desktop, click the 3270 Terminal icon to open the 3270 Terminal application plug-in.

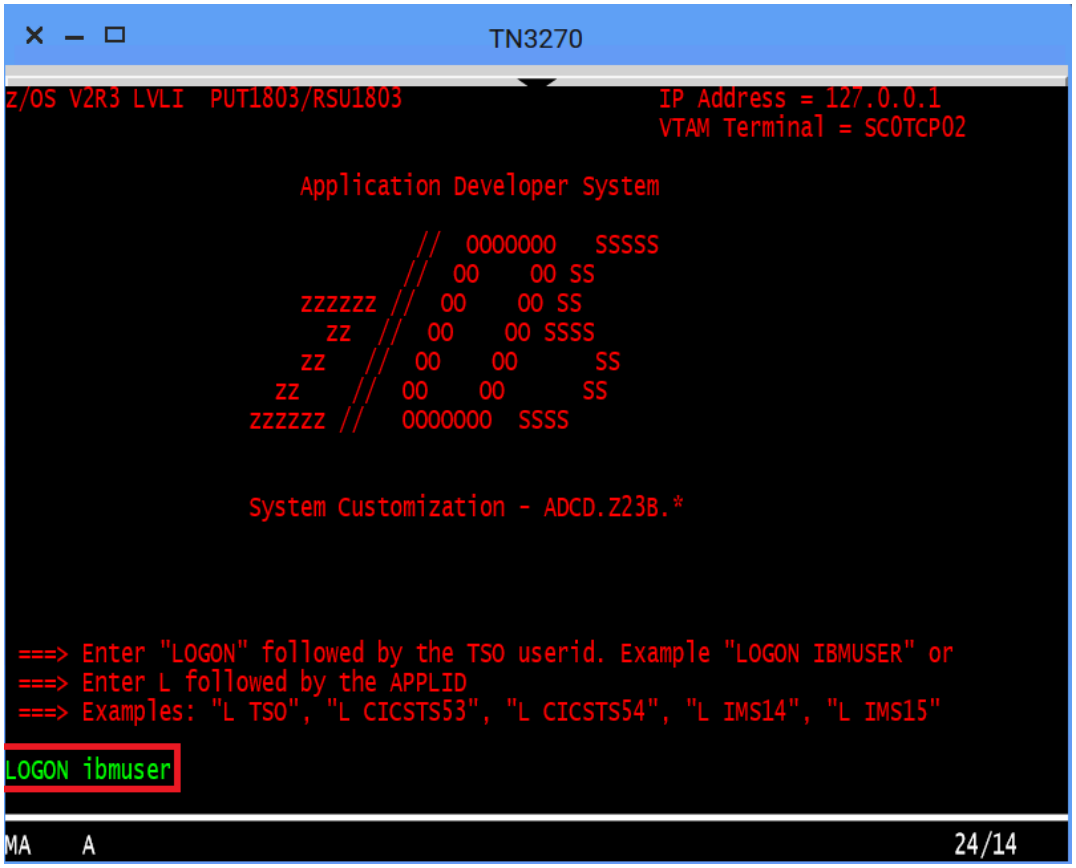


The 3270 Terminal panel is displayed, which offers selections to access various mainframe services.

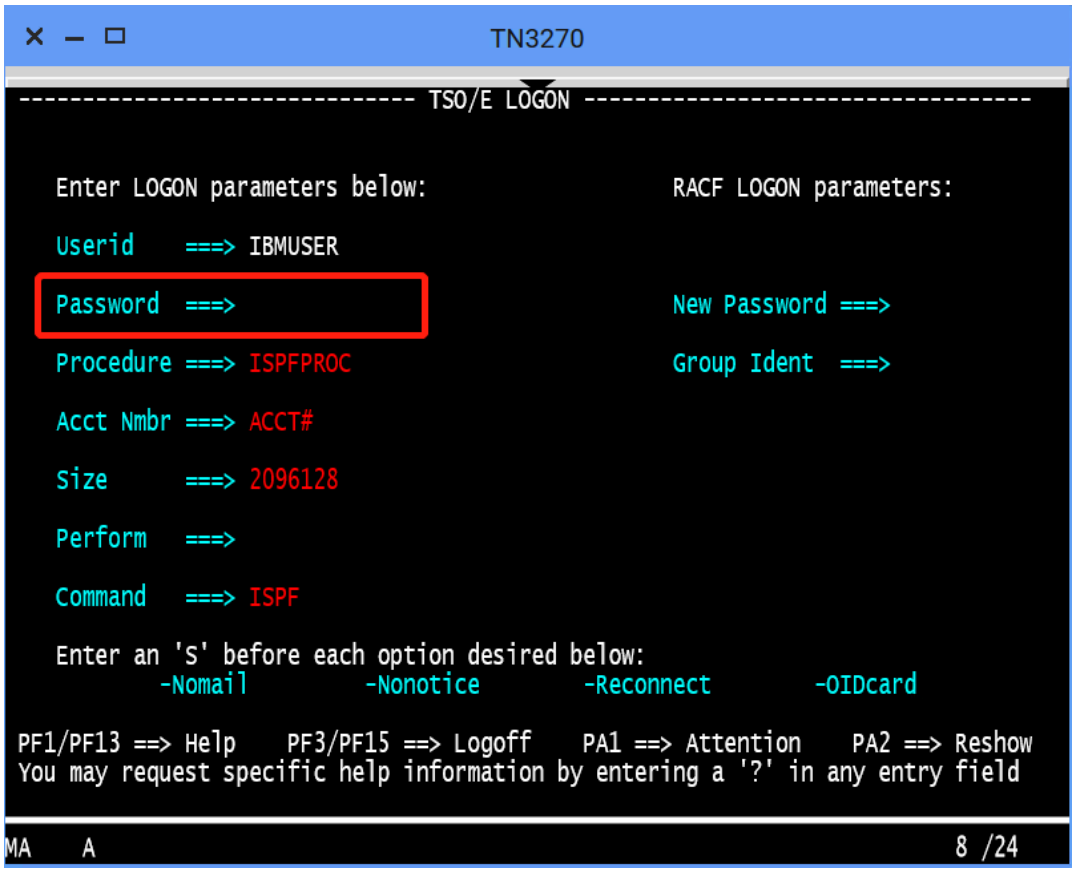


2. Enter the following command and press Enter to log on to TSO:

   ```
   LOGON ibmuser
   ```

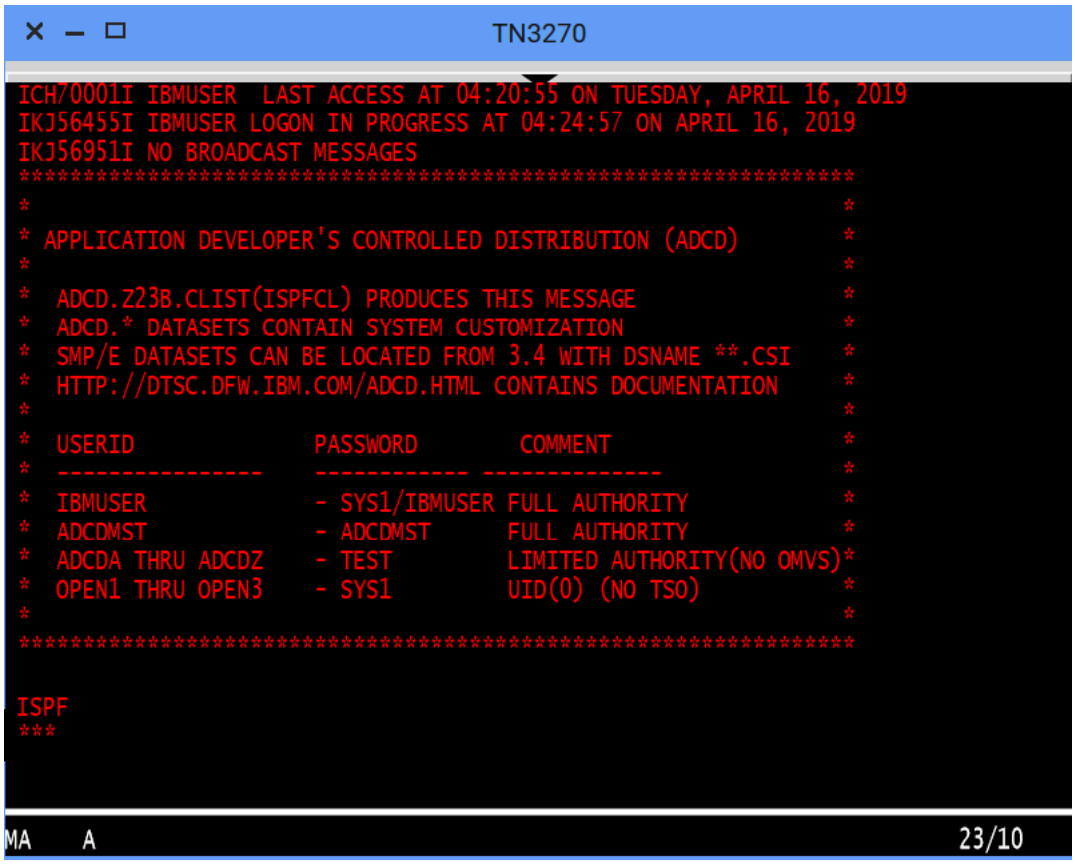


3. On the TSO/E LOGON panel, enter the password `sys1` in the **Password** field and press Enter.



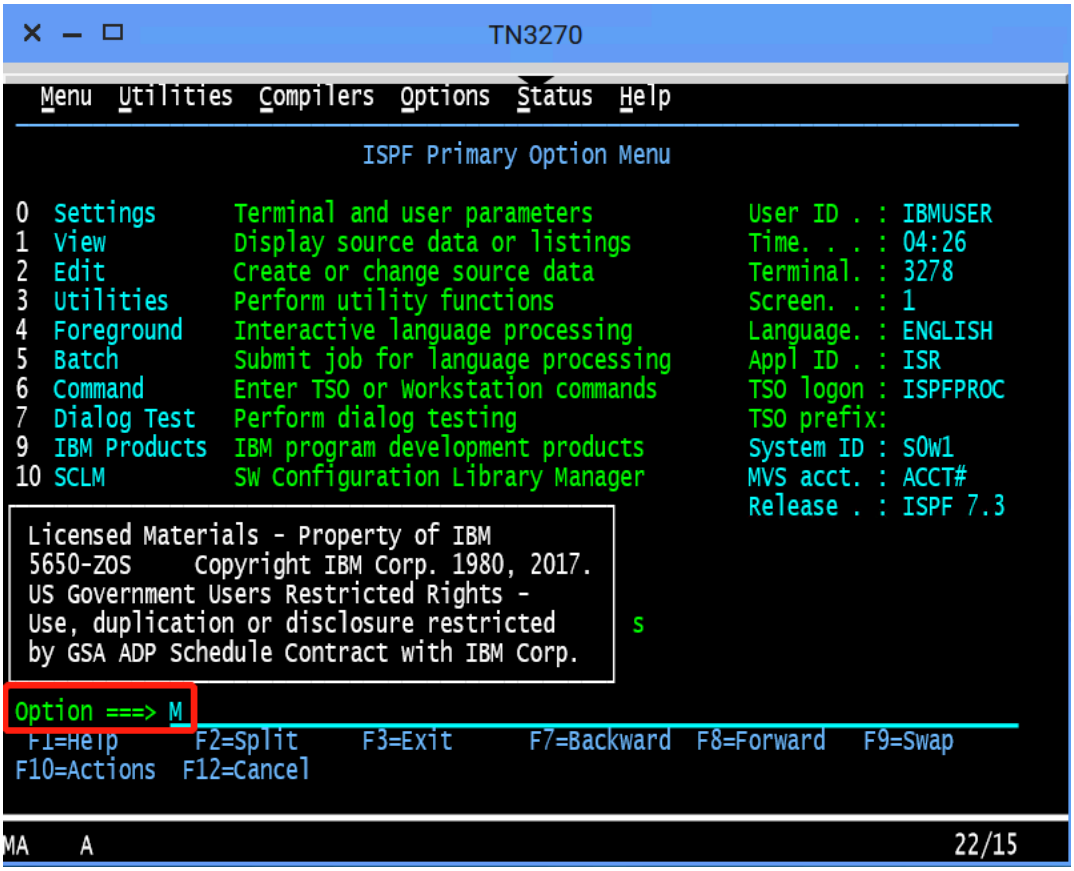
You successfully log on to TSO.

4. When you see the following screen, press Enter. The **ISPF Primary Option Menu** is displayed.



5. Access SDSF to view output from a job. To do this,

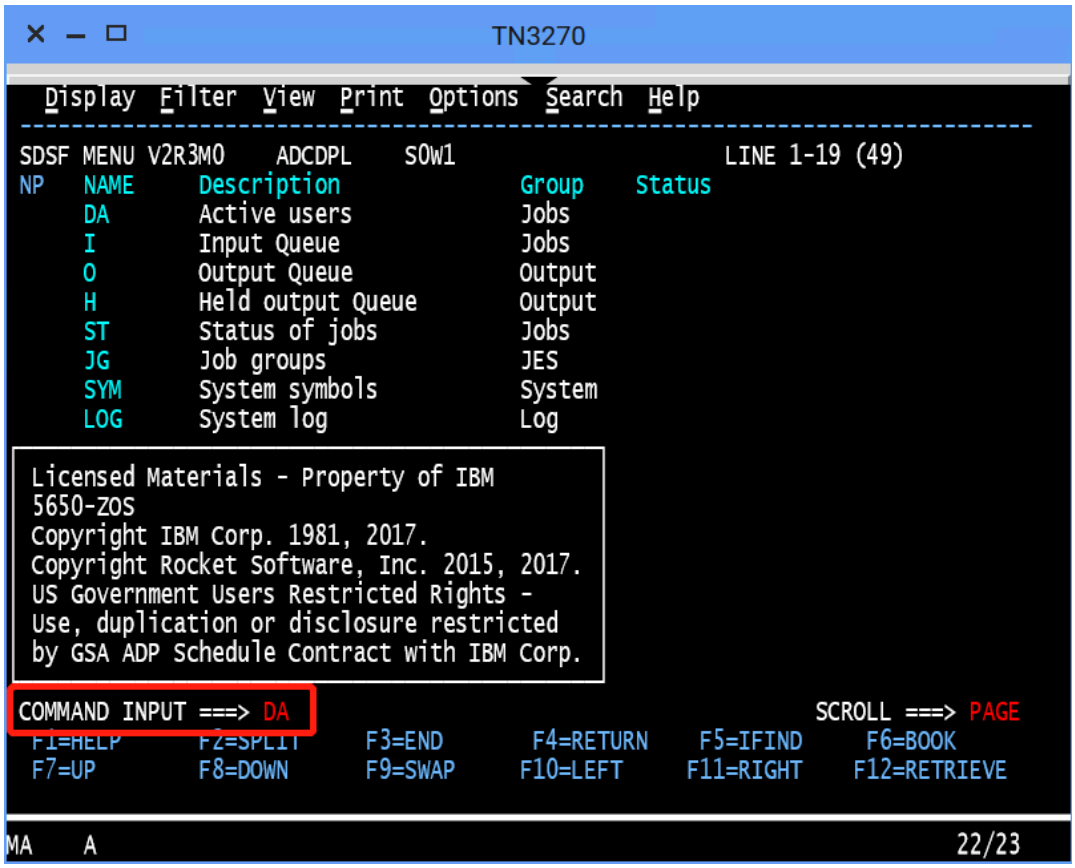
   a. Type `M` at the **Option** prompt and press Enter.



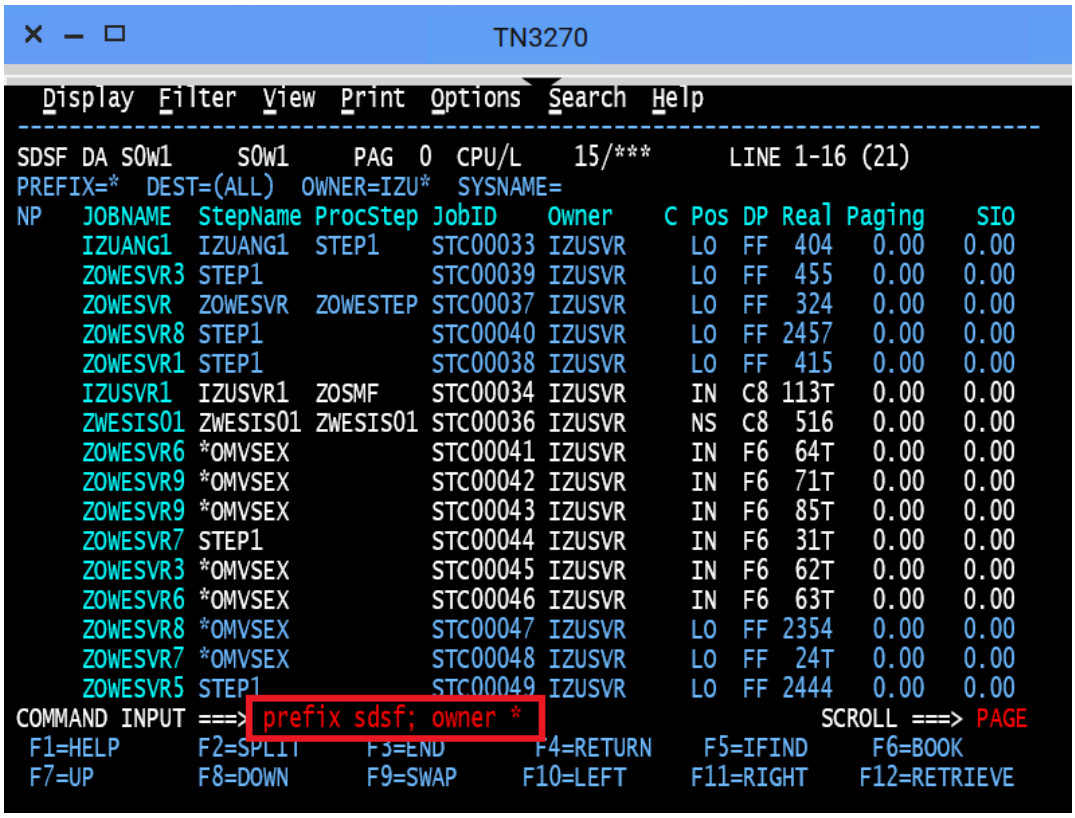
   b. Type `5` and press Enter.

TN3270
Menu Help
IBM Products Panel
1 SMP/E System Modification Program/Extended
2 ISMF Integrated Storage Management Facility
3 RACF Resource Access Control Facility
4 HCD Hardware Configuration Dialogs
5 SDSF Spool Search and Display Facility
6 IPCS Interactive Problem Control System
7 DITTO DITTO/ESA for MVS Version 1
8 RMF Resource Measurement Facility
10 OMVS MVS OpenEdition
11 RRS Resource Recovery Services
12 MQ V9.0.5 WMQ Series Operations and Control
13 MQ V8.0 WMQ Series Operations and Control
14 FMN File Manager Operations and Control
15 WLM Workload Manager
16 RTC Rational Team Concert
Option ===> 5
F1=Help F2=Split F3=Exit F7=Backward F8=Forward F9=Swap
F10=Actions F12=Cancel
MA A 22/15

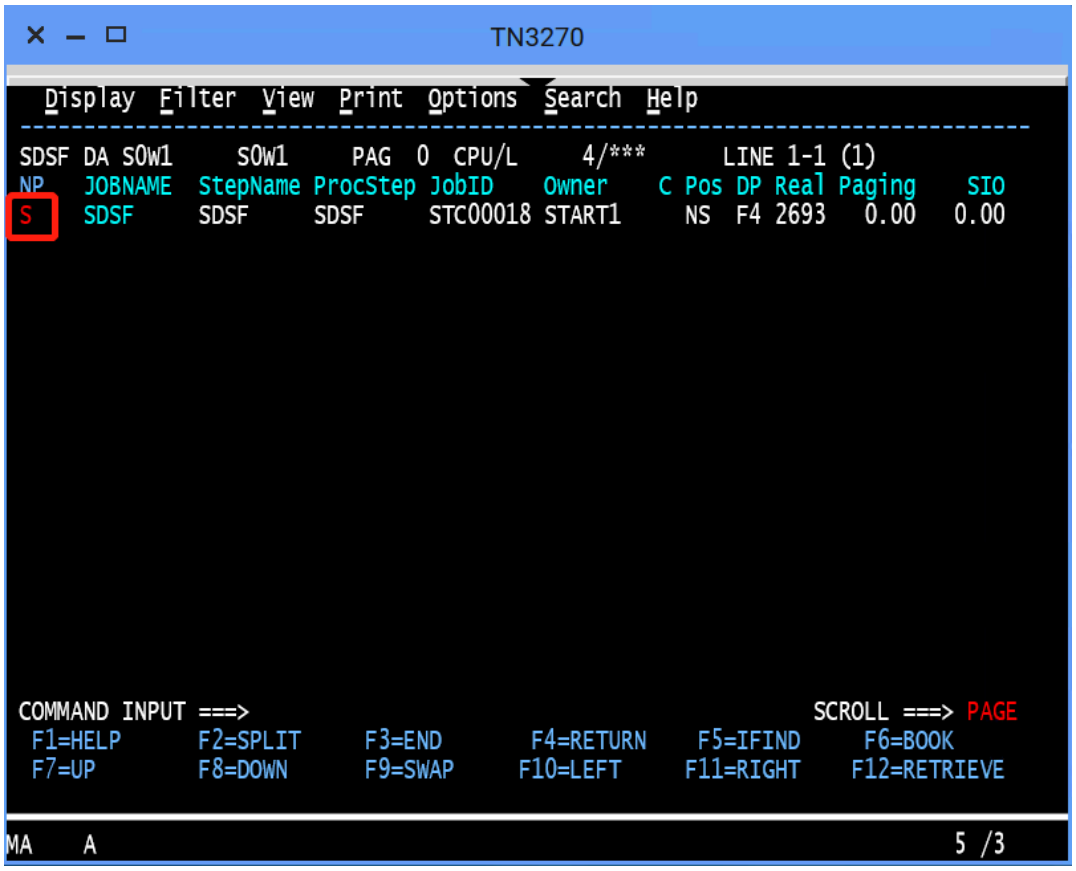
6. To view the jobs in an active status, type DA at the command input prompt and press Enter. The jobs that are running are displayed.



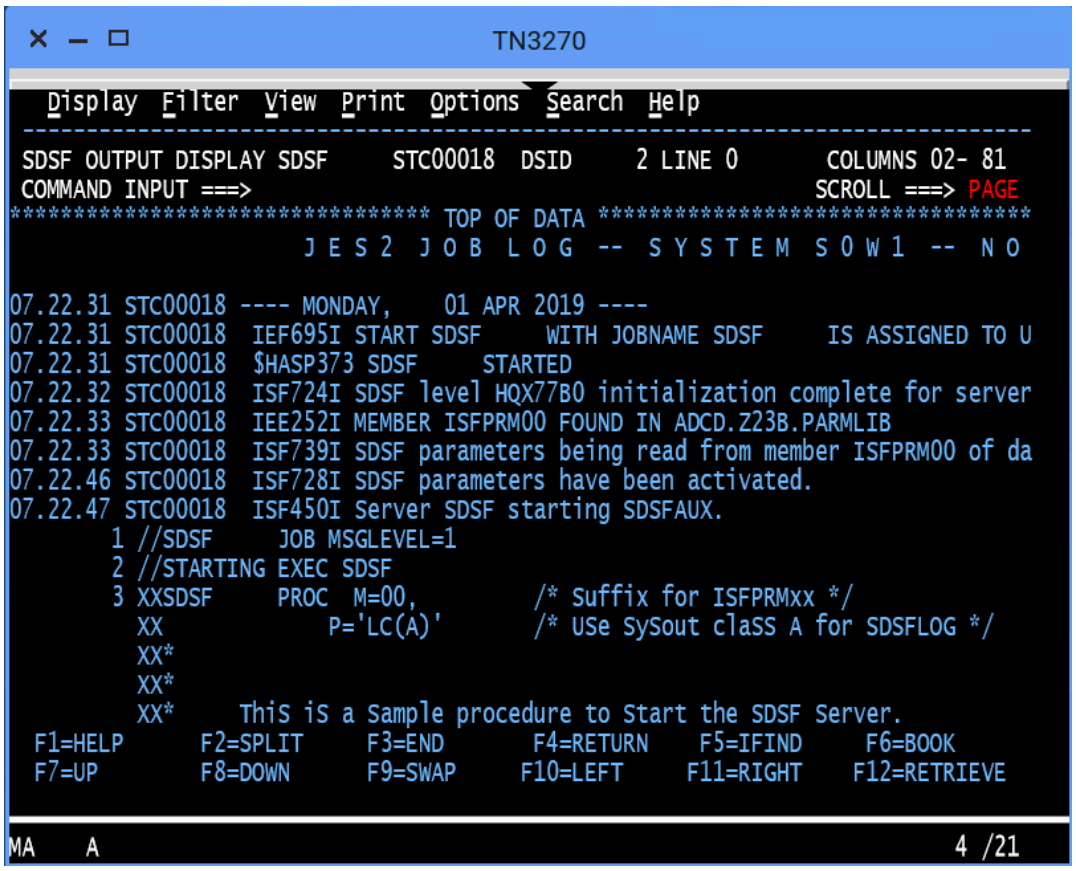
7. To query the jobs that start with SDSF, type prefix `sdsf; owner *` at the command input prompt and press Enter.



8. To view the contents of the job, type S next to the job name SDSF and press Enter.



The contents of the job are displayed.



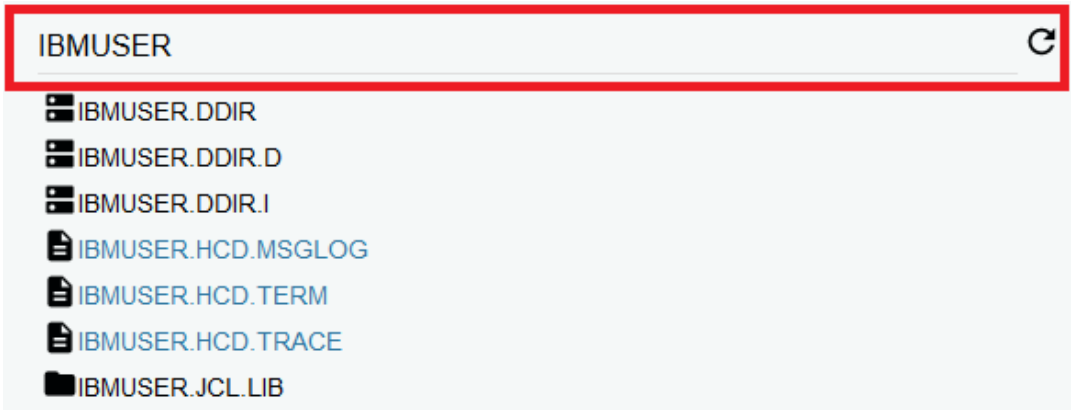
Close the 3270 Terminal window. In the next step, you will use the MVS Explorer to make changes to a data set.

Editing a data set in MVS Explorer

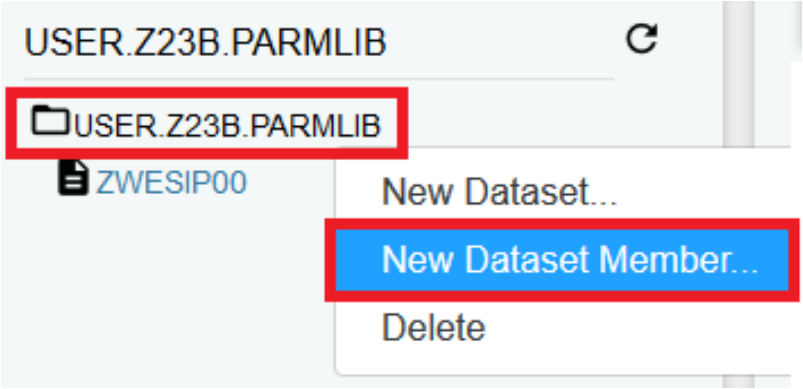
Use the MVS Explorer to create and edit a data set member and save the changes. The MVS Explorer view lets you browse the MVS file system by creating filters against data set names.

Follow these steps:

1. Click the Start menu on Zowe Desktop.
2. Scroll down to find the MVS Explorer icon and pin this application to the desktop for later use.
3. Click the **MVS Explorer** icon on the taskbar. The MVS Explorer opens. The **Filter** field is pre-filled with the user name. In this tutorial, the filter string is `IBMUSER`. All the data sets matching this filter are displayed. You can expand a data set name and see the members in it.

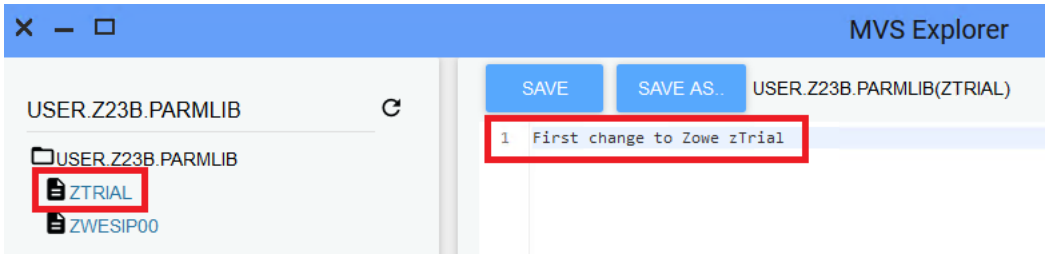


4. Enter `USER.Z23B.PARMLIB` in the **Filter** field to locate this data set and then click to expand it. Ensure that there is no extra space before the data set member name that you enter.
5. Right-click the `USER.Z23B.PARMLIB` data set and select **New Dataset Member**.



6. Enter **ZTRIAL** as the data set member name and click **OK** to create the data set member.

7. Click the data set member you just created and edit it by adding a new sentence, for example, `First change to Zowe zTrial`.



8. Click **SAVE** to save your edits.

   The following message **Save success for USER.Z23B.PARMLIB(ZTRIAL)** pops up quickly at the bottom of the MVS Explorer window, which indicates that your edits are successfully saved.

Leave the MVS Explorer window open; we will look at the contents of the data set in a later step. If asked to leave the page, choose **Stay on Page**. Next, you will use Zowe CLI to view and add another change to the same data set.

Using the Zowe CLI to edit a data set

Use Zowe CLI to download the same data set that you edited by using MVS Explorer in the previous step, edit it, and upload the changes to the mainframe.

Zowe CLI is a command-line interface that lets you interact with z/OS from various other platforms, such as cloud or distributed systems, to submit jobs, issue TSO and z/OS console commands, integrate z/OS actions into scripts, produce responses as JSON documents, and more. With this extensible and scriptable interface, you can tie in mainframes to distributed DevOps pipelines and build automation.

Follow these steps:

1. Start the Command Prompt or a terminal in your local desktop. In this tutorial, it's assumed that you use Windows Command Prompt.

2. (Optional) Issue the following command to view the top-level help descriptions.

```
zowe --help
```

Tip: The command `zowe` initiates the product on a command line. All Zowe CLI commands begin with `zowe`.

3. To list the data sets of USER, enter the following command:

```
zowe zos-files list data-set "USER.*"
```

The following data sets are listed.

```
C:\Users\Administrator>zowe zos-files list data-set "USER.*"
USER.ADCD.SMPELIST
USER.Z23B.CIDTABL
USER.Z23B.CLIST
USER.Z23B.HELP
USER.Z23B.ISPLLIB
USER.Z23B.ISPMLIB
USER.Z23B.ISPPLIB
USER.Z23B.ISPSLIB
USER.Z23B.ISPTLIB
USER.Z23B.LINKLIB
USER.Z23B.LOADLIB
USER.Z23B.LPALIB
USER.Z23B.MSGENU
USER.Z23B.PARMLIB
USER.Z23B.PROCLIB
USER.Z23B.STCJOBS
USER.Z23B.SYSEXEC
USER.Z23B.TCPPARMS
USER.Z23B.VTAMLIB
USER.Z23B.VTAMLST
```

4. To download all the data set members of USER.Z23B.PARMLIB, enter the following command:

```
zowe zos-files download all-members "USER.Z23B.PARMLIB"
```

The message "Data set downloaded successfully" indicates that the data set members are downloaded. A file heirarchy is added to your current directory.

```
C:\Users\Administrator>zowe zos-files download all-members "USER.Z23B.PARMLIB"
Data set downloaded successfully.
Destination: user/z23b/parmlib
```

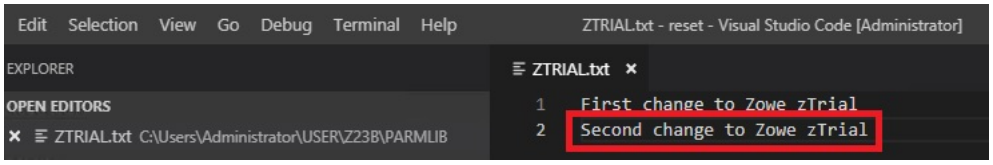
5. To open the data set member named ZTRIAL in Visual Studio Code, issue the following command against the same directory where you issued the download command:

```
code USER/Z23B/PARMLIB/ZTRIAL.txt
```

Alternatively, navigate to the PARMLIB directory and issue `code ZTRIAL.txt`.

The file opens in a text editor (in this example, VS Code). You will see the changes you made in the previous step by using the MVS Explorer.

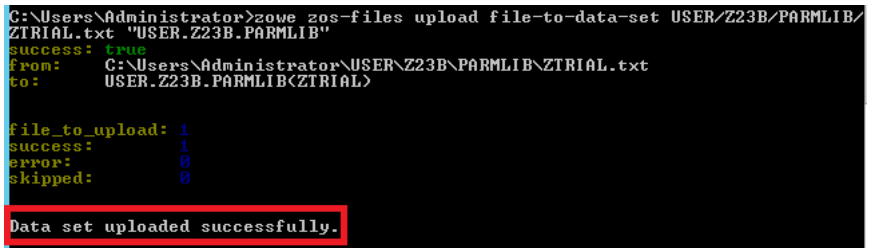
6. Add the text `Second change to Zowe zTrial` to the file and then use `Ctrl+S` to save your edits.



7. Open the Command Prompt again and upload your changes to the mainframe by entering the following command:

```
zowe zos-files upload file-to-data-set USER/Z23B/PARMLIB/ZTRIAL.txt
 "USER.Z23B.PARMLIB"
```

The following message indicates that you successfully uploaded your changes:



Congratulations! You used Zowe CLI to edit a data set member and upload the changes to mainframe.

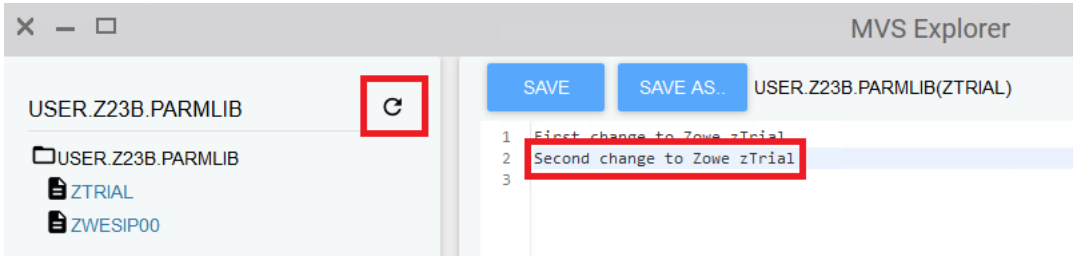
Close the Command Prompt window. In the next step, you will open the MVS Explorer again to view the updates that you made to the data set in this procedure.

Viewing the data set changes in MVS Explorer

Use the MVS Explorer to view the data set changes from the previous step.

Procedure

1. Return to the Zowe Desktop and open the MVS Explorer application.
2. Locate the data set member **USER.Z23B.PARMLIB** > **ZTRIAL** and click the refresh icon. You will see the changes you just made by using Zowe CLI.



Congratulations! You explored several applications on the Zowe Desktop and learned how to work with them.

Next steps

Here are some next steps.

Go deeper with Zowe

In roughly 20 minutes, you used the MVS™ Explorer and Zowe CLI to edit the same data set member, and used the JES Explorer and the 3270 Terminal to query the same JES job with filters, all without leaving Zowe. Now that you're familiar with Zowe components, you can continue to learn more about the project. Zowe also offers many more plug-ins for both Zowe Desktop and Zowe CLI.

For more information, see the Using the Zowe Desktop on page 146.

For a complete list of available CLI commands, explore the Zowe CLI Command Reference Guide.

Try the Extending Zowe scenarios

You can add your own application plug-ins to Zowe. See how easy it is to extend Zowe to create your own APIs and applications by reading the Onboarding Overview on page 164 section.

Give feedback

Did you find this tutorial useful? You can tell us what you think about this tutorial via an online survey.

If you encounter any problems or have an idea for improving this tutorial, you can create a GitHub issue here.

Using the Zowe Desktop

You can use the Zowe™ Application Framework to create application plug-ins for the Zowe Desktop. For more information, see Overview on page 237.

Navigating the Zowe Desktop

From the Zowe Desktop, you can access Zowe applications.

Accessing the Zowe Desktop

From a supported browser, open the Zowe Desktop at `https://myhost:httpsPort/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html`

where:

- *myHost* is the host on which you are running the Zowe Application Server.
- *httpsPort* is the value that was assigned to *node.https.port* in `zluxserver.json`. For example, if you run the Zowe Application Server on host *myhost* and the value that is assigned to *node.https.port* in `zluxserver.json` is 12345, you would specify `https://myhost:12345/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html`.

Logging in and out of the Zowe Desktop

1. To log in, enter your mainframe credentials in the **Username** and **Password** fields.
2. Press Enter. Upon authentication of your user name and password, the desktop opens.

To log out, click the the avatar in the lower right corner and click **Sign Out**.

Pinning applications to the task bar

1. Click the Start menu.
2. Locate the application you want to pin.
3. Right-click the on the application icon and select **Pin to taskbar**.

Changing the desktop language

Use the Languages setting in the personalization panel to change the desktop language. After you change the language and restart Zowe, desktop menus and text display in the specified language. Applications that support the specified desktop language also display in that language.

1. Click the personalization icon in the lower right corner.
2. Click **Languages**.
3. In the **Languages** dialog, click a language, and then click **Apply**.
4. When you are prompted, restart Zowe.

Zowe Desktop application plug-ins

Application plug-ins are applications that you can use to access the mainframe and to perform various tasks. Developers can create application plug-ins using a sample application as a guide. The following application plug-ins are installed by default:

Hello World Sample

The Hello World sample application plug-in for developers demonstrates how to create a dataservice and how to create an application plug-in using Angular.

IFrame Sample

The IFrame sample application plug-in for developers demonstrates how to embed pre-made webpages within the desktop as an application and how an application can request an action of another application (see the source code for more information).

z/OS Subsystems

The z/OS Subsystems plug-in helps you find information about the important services on the mainframe, such as CICS, Db2, and IMS.

TN3270

This TN3270 plug-in provides a 3270 connection to the mainframe on which the Zowe Application Server runs.

VT Terminal

The VT Terminal plug-in provides a connection to UNIX System Services and UNIX.

API Catalog

The API Catalog plug-in lets you view API services that have been discovered by the API Mediation Layer. For more information about the API Mediation Layer, Discovery Service, and API Catalog, see Zowe overview on page 8.

Editor

With the Zowe Editor you can create and edit files on the system that Zowe serves.

Workflows

From the Workflows application plug-in you can create, manage, and use z/OSMF workflows to manage your system.

JES Explorer

Use this application to query JES jobs with filters, and view the related steps, files, and status. You can also purge jobs from this view.

MVS Explorer

Use this application to browse the MVS™ file system by using a high-level qualifier filter. With the MVS Explorer, you can complete the following tasks:

- List the members of partitioned data sets.
- Create new data sets using attributes or the attributes of an existing data set ("Allocate Like").
- Submit data sets that contain JCL to Job Entry Subsystem (JES).
- Edit sequential data sets and partitioned data set members with basic syntax highlighting and content assist for JCL and REXX.
- Conduct basic validation of record length when editing JCL.
- Delete data sets and members.
- Open data sets in full screen editor mode, which gives you a fully qualified link to that file. The link is then reusable for example in help tickets.

USS Explorer

Use this application to browse the USS files by using a path. With the USS Explorer, you can complete the following tasks:

- List files and folders.
- Create new files and folders.
- Edit files with basic syntax highlighting and content assist for JCL and REXX.
- Delete files and folders.

Using the Workflows application plug-in

The Workflows application plug-in is available from the Zowe Deskstop Start menu. To launch Workflows, click the Start menu in the lower-left corner of the desktop and click the Workflows application plug-in icon. The **Users/Tasks Workflows** window opens.

Logging on to the system

If you are prompted to log on to the system, complete these steps:

1. Enter your user ID and password.

2. Click **Sign in**.

Updating the data display

To refresh the data on any tab, click in the upper right corner of the window.

Configuration

From the **Configuration** tab, you can view, add, and remove servers.

Adding a z/OSMF server

Complete these steps to add a new z/OSMF server:

1. Click the **Configuration** tab.
2. Click the plus sign (+) on the left side of the window.
3. In the **Host** field, type the name of the host.
4. In the **Port** field, type the port number.
5. Click **OK**.

Testing a server connection

To test the connection, click **Test**. When the server is online the **Online** indicator next to the server **Host** and **Port** is green.

Setting a server as the default z/OSMF server

Complete these steps to set a default z/OSMF server:

1. Click **Set as default**.
2. Enter your user ID and password.
3. Click **Sign in**.

Note: You must specify a default server.

Removing a server

To remove a server, click **x** next to the server that you want to remove.

Reload a server configuration

To reload a server configuration, click **Reload**.

Save a server configuration

To save a server configuration, click **Save**.

Workflows

To display all workflows on the system, click the **Workflows** tab.

You can sort the workflows based on the following information:

Workflow

The name of the workflow.

Description

The description of the workflow.

Version

The version number.

Owner

The user ID of the workflow owner.

System

The system identifier.

Status

The status of the workflow (**In progress** or **Completed**).

Progress

Indicates how much of the workflow has been completed based on the number of tasks completed.

Searching workflows

To locate a specific workflow, type a search string in the search box in the upper right corner of the window.

Defining a workflow

To define a workflow, complete these steps:

1. From the **Workflows** tab, click **Actions** > **New workflow**. (By default, the **Advanced Mode** check box is selected.)
2. In the **Name** field, specify a descriptive name for the workflow.
3. In the **Workflow definition file** field, specify the primary XML file for this workflow.
4. In the **System** field, specify a system.
5. In the **Owner** field, specify the user ID of the person that is responsible for assigning the tasks in the workflow. (To set the owner to the current user, select the **Set owner to current user** check box.)
6. Click **OK**.

Viewing tasks

To view the tasks associated with a workflow, click the **My Tasks** tab. Workflows that have assigned tasks are shown on the left side of the window. The task work area is on the right side of the window.

You can choose to view workflows that have **Pending** or **Completed** tasks or you can choose to view all workflows (**Pending** and **Completed**) and their tasks, regardless of the task status.

For each workflow, you can click the arrow to expand or collapse the task list. Assigned tasks display below each workflow. Hovering over each task displays more information about the task, such as the status and the owner.

Each task has a indicator of **PERFORM** (a step to be performed) or **CHECK** (Check the step that was performed). Clicking **CHECK** or **PERFORM** opens a work area on the right side of the window. When a task is complete, a green clipboard icon with a checkmark is displayed.

Note: If you are viewing tasks on a **Pending** or **Completed** tab, only those workflows that have tasks with a corresponding status, are displayed.

Task work area

When you click **CHECK** or **PERFORM**, a work area on the right side of the window opens to display the steps to complete the task. Expand or collapse the work area by clicking .

Tip: Hovering over the task description in the title bar of the work area window on the right side displays more information about the corresponding workflow and the step description.

Performing a task

1. To perform a task that has steps that are assigned to you, click **PERFORM**.
2. Use the work area to perform the steps associated with the selected task. Depending on the task, you might use an embedded tool (such as another application) or you might complete a series of steps to complete the task.
3. If there are multiple steps to perform, click **Next** to advance to the next step for the task.
4. Click **Finish**.

Note: When a task is complete, a green clipboard icon with a checkmark is displayed next to the task.

Checking a task

1. To check a task, click **CHECK**.

2. In the task work area, view the JESMSGLG, JESJCL, JESYSMSG, or SYSTSPRT output that is associated with the selected task.

Managing tasks

To manage a task in the PERFORM status, click to the right of the task status. Choose from the following options:

Properties

Display the title and description of the task.

Perform

Perform the first step.

Skip

Skip this step.

Override Complete

Override the completion of the step. The selected step will be bypassed and will not be performed for this workflow. You must ensure that the step is performed manually.

Assignment

Opens the Manage Assignees window where authorized users can add or remove the user ID of the person that is assigned to the step.

Return

Remove ownership of the step.

Viewing warnings

To view any warning messages that were encountered, click the **Warnings** tab. A message is listed in this tab each time it is encountered.

To locate a specific message, type a search string in the search box in the upper right corner of the window.

You can sort the warning messages based on the following information.

Message Code

The message code that is associated with the warning.

Description

A description of the warning.

Date

The date of the warning.

Corresponding Workflow

The workflow that is associated with the warning.

Using the Editor

With the Zowe Editor, you can create and edit the many types of files.

Specifying a language server

To specify a language server, complete these steps:

1. From the **Language Server** menu, select **URL***.
2. From the **Language Server Setting**, **Put your config here** area, paste your configuration.
3. Ensure that the **Enable Language Server** check box is selected.
4. Click **Save**.

Specifying a language

From the **Language** menu, select the language you want to use.

Opening a directory

To open a directory on the system, complete these steps:

1. From the **File** menu, select **Open Directory**.(Alternatively, you can click **Open Directory** in the File Explorer.)
2. From the **Open Directory**, **Input Your Directory** field, type the name of the directory you want to open. For example: `/u/zs1234`
3. Click **Open**.

The File Explorer on the left side of the window lists the folders and files in the specified directory. Clicking on a folder expands the tree. Clicking on a file opens a tab that displays the file contents.

Creating a new file

To create a new file, complete these steps:

1. From the **File** menu, select **New File**. The **New File** tab opens.
2. From the **New File**, **File Name** field, type the name of the file.
3. Click **Create**.

Saving a file

To save a file, click **File** > **Save**.

Note: To save all files, click **File** > **Save All** (or Ctrl+S).

API Catalog

As an application developer, use the API Catalog to view what services are running in the API Mediation Layer. Through the API Catalog, you can also view the associated API documentation corresponding to a service, descriptive information about the service, and the current state of the service. The tiles in the API Catalog can be customized by changing values in the mfaas.catalog-ui-tile section defined in the application.yml of a service. A microservice that is onboarded with the API Mediation Layer and configured appropriately, registers automatically with the API Catalog and a tile for that service is added to the Catalog.

Note: For more information about how to configure the API Catalog in the application.yml, see: Onboard a Java REST APIs with Spring Boot on page 188.

View Service Information and API Documentation in the API Catalog

Use the API Catalog to view services, API documentation, descriptive information about the service, the current state of the service, service endpoints, and detailed descriptions of these endpoints.

Note: Verify that your service is running. At least one started and registered instance with the Discovery Service is needed for your service to be visible in the API Catalog.

Follow these steps:

1. Use the search bar to find the service that you are looking for. Services that belong to the same product family are displayed on the same tile.

   Example: `Sample Applications, Endevor, SDK Application`

2. Click the tile to view header information, the registered services under that family ID, and API documentation for that service.

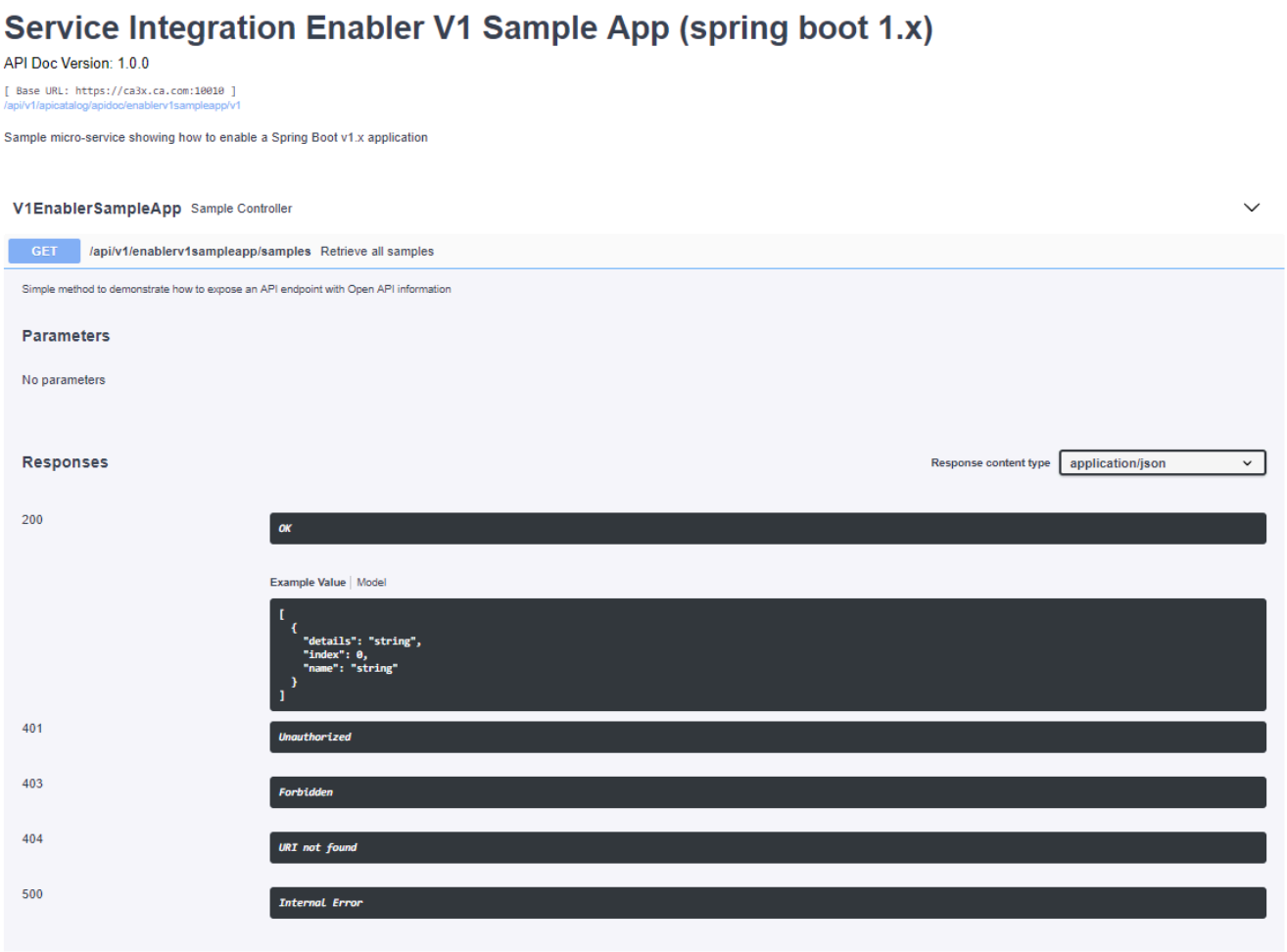
   Notes:

   - The state of the service is indicated in the service tile on the dashboard page. If no instances of the service are currently running, the tile displays a message displays that no services are running.
   - At least one instance of a service must be started and registered with the discovery service for it to be visible in the API Catalog. If the service that you are onboarding is running, and the corresponding API documentation is displayed, this API documentation is cached and remains visible even when the service and all service instances stop.
   - Descriptive information about the service and a link to the home page of the service is displayed.

   Example:

3. Expand the endpoint panel to see a detailed summary with responses and parameters of each endpoint, the endpoint description, and the full structure of the endpoint.

   Example:



   Notes:

   - If a lock icon is visible on the right side of the endpoint panel, the endpoint requires authentication.
   - The structure of the endpoint is displayed relative to the base URL.
   - The URL path of the abbreviated endpoint relative to the base URL is displayed in the following format:

     Example:

     `/api/v1/{yourServiceId}/{endpointName}`

     The path of the full URL that includes the base URL is also displayed in the following format:

     `https://hostName:basePort/api/v1/{yourServiceId}/{endpointName}`

     Both links target the same endpoint location.

Zowe CLI extensions and plug-ins

Extending Zowe CLI

You can install plug-ins to extend the capabilities of Zowe™ CLI. Plug-ins CLI to third-party applications are also available, such as Visual Studio Code Extension for Zowe (powered by Zowe CLI). Plug-ins add functionality to the product in the form of new command groups, actions, objects, and options.

Important! Plug-ins can gain control of your CLI application legitimately during the execution of every command. Install third-party plug-ins at your own risk. We make no warranties regarding the use of third-party plug-ins.

- Installing Zowe CLI plug-ins on page 154
- IBM® CICS® Plug-in for Zowe CLI on page 158
- IBM® Db2® Database Plug-in for Zowe CLI on page 158
- Zowe Explorer Extension for VSCode on page 161

Software requirements for Zowe CLI plug-ins

Before you use Zowe™ CLI plug-ins, complete the following steps:

1. Install Installing Zowe CLI on page 104 on your computer.
2. Complete the following configurations:

Plug-in	Required Configurations
IBM® CICS® Plug-in for Zowe CLI on page 158	• Ensure that IBM CICS Transaction Server v5.2 or later is installed and running in your mainframe environment • IBM CICS Management Client Interface (CMCI) is configured and running in your CICS region.
IBM® Db2® Database Plug-in for Zowe CLI on page 158	• Download and prepare the ODBC driver (required for only package installations) and address the licensing requirements. • **(MacOS)** Download and Install Xcode.

Important! You can perform the required configurations for the plug-ins that you want to use ***before*** or ***after*** you install the plug-ins. However, if you do not perform the required configurations, the plug-ins will not function as designed.

Installing Zowe CLI plug-ins

Use commands in the `plugins` command group to install and manage Zowe™ CLI plug-ins.

Important! Plug-ins can gain control of your CLI application legitimately during the execution of commands. Install third-party plug-ins at your own risk. We make no warranties regarding the use of third-party plug-ins.

You can install the following Zowe plug-ins:

- IBM® CICS® Plug-in for Zowe CLI
- IBM® Db2® Plug-in for Zowe CLI
- Third-party Zowe Conformant Plug-ins

Use either of the following methods to install plug-ins:

- Install from an online NPM registry. Use this method when your computer ***can*** access the Internet.

  For more information, see Installing plug-ins from an online registry on page 155.

- Install from a local package. With this method, you download and install the plug-ins from a bundled set of `.tgz` files. Use this method when your computer ***cannot*** access the Internet.

  For more information, see Installing plug-ins from a local package on page 155.

Installing plug-ins from an online registry

Install Zowe CLI plug-ins using npm commands on Windows, Mac, and Linux. The procedures in this article assume that you previously installed the core CLI.

Follow these steps:

1. Meet the Software requirements for Zowe CLI plug-ins on page 154 that you install.
2. Set the proper target registry by issuing the following command:

   ```
   npm config set @brightside:registry https://api.bintray.com/npm/ca/
   brightside
   ```

3. Issue the following command to install a plug-in:

   ```
   zowe plugins install <my-plugin>
   ```

   Note: Replace `<my-plugin>` with the installation command syntax in the following table:

Plug-in	Installation Command Syntax
IBM CICS Plug-in for Zowe CLI	`@brightside/cics@lts-incremental`
IBM Db2 Plug-in for Zowe CLI	`@brightside/db2@lts-incremental`

4. (Optional) Issue the following command to install two or more plug-ins using one command. Separate the `<my-plugin>` names with one space.

   ```
   zowe plugins install <@brightside/my-plugin1> <@brightside/my-plugin2>
    <@brightside/my-plugin3> ...
   ```

   Note: The IBM Db2 Plug-in for Zowe CLI requires additional licensing and ODBC driver configurations. If you installed the DB2 plug-in, see IBM® Db2® Database Plug-in for Zowe CLI on page 158.

You installed Zowe CLI plug-ins.

Installing plug-ins from a local package

Install plug-ins from a local package on any computer that has limited or no access to the Internet. The procedure assumes that you previously installed the core CLI.

Follow these steps:

1. Meet the Software requirements for Zowe CLI plug-ins on page 154 that you want to install.
2. Obtain the installation files.

   From the Zowe Download website, click **Download Zowe CLI** to download the Zowe CLI installation package named `zowe-cli-package-*v*.*r*.*m*.zip` to your computer.

   Note: `v` indicates the version, `r` indicates the release number, and `m` indicates the modification number

3. Open a command-line window such as Windows Command Prompt. Browse to the directory where you downloaded the Zowe CLI installation package. Issue the following command to unzip the files:

   ```
   unzip zowe-cli-package-v.r.m.zip
   ```

   Example:

   ```
   unzip zowe-cli-package-1.0.1.zip
   ```

   By default, the unzip command extracts the contents of the zip file to the directory where you downloaded the file. Optionally, extract the contents of the .zip file to your preferred location.
4. Open a command-line window and change to the local directory where you extracted the zip file.

   Example:

   ```
   cd C:\Users\userID\my_downloads\<file_name>.zip
   ```
5. Issue the following command to install the plug-in:

   ```
   zowe plugins install <my-plugin>
   ```

   Replace `<my-plugin>` with the .tgz file name listed in the following table:

Plug-in	`.tgz` File Name
IBM CICS Plug-in for Zowe CLI	`cics.tgz`
IBM Db2 Plug-in for Zowe CLI	`db2.tgz`

You installed Zowe CLI plug-ins.

Validating plug-ins

Issue the plug-in validation command to run tests against all plug-ins (or against a plug-in that you specify) to verify that the plug-ins integrate properly with Zowe CLI. The tests confirm that the plug-in does not conflict with existing command groups in the base application. The command response provides you with details or error messages about how the plug-ins integrate with Zowe CLI.

The `validate` command has the following syntax:

```
zowe plugins validate [plugin]
```

- **`[plugin]`**(Optional) Specifies the name of the plug-in that you want to validate. If you do not specify a plug-in name, the command validates all installed plug-ins. The name of the plug-in is not always the same as the name of the NPM package.

  |Plug-in|Syntax| |-|-| |IBM CICS Plug-in for Zowe CLI|`@brightside/cics`| |IBM Db2 Plug-in for Zowe CLI|`@brightside/db2`| |||

Examples: Validate plug-ins

- The following example illustrates the syntax to use to validate the IBM CICS Plug-in for Zowe CLI:

  ```
  zowe plugins validate @brightside/cics
  ```
- The following example illustrates the syntax to use to validate all installed plug-ins:

  ```
  zowe plugins validate
  ```

Updating plug-ins

You can update Zowe CLI plug-ins from an online registry or from a local package.

Update plug-ins from an online registry

Issue the `update` command to install the latest stable version or a specific version of a plug-in that you installed previously. The `update` command has the following syntax:

```
zowe plugins update [plugin...] [--registry <registry>]
```

- Specifies the name of an installed plug-in that you want to update. The name of the plug-in is not always the same as the name of the NPM package. You can use npm semantic versioning to specify a plug-in version to which to update. For more information, see npm semver.
- [--registry \<registry>\]

  (Optional) Specifies a registry URL that is different from the registry URL of the original installation.

Examples: Update plug-ins

The following example illustrates the syntax to use to update an installed plug-in to the latest version:

```
zowe plugins update @brightside/my-plugin@lts-incremental
```

The following example illustrates the syntax to use to update a plug-in to a specific version:

```
zowe plugins update @brightside/my-plugin@"^1.2.3"
```

Update plug-ins from a local package

You can update plug-ins from a local package. You acquire the media from the Zowe Download website and update the plug-ins using the `zowe plugins install` command.

To update plug-ins from a local package, follow the steps described in Installing plug-ins from a local package on page 155.

Uninstall Plug-ins

Issue the `uninstall` command to uninstall plug-ins from Zowe CLI. After the uninstall process completes successfully, the product no longer contains the plug-in configuration.

The uninstall command contains the following syntax:

```
zowe plugins uninstall [plugin]
```

- `[plugin]`

  Specifies the name of the plug-in that you want to uninstall.

The following table describes the uninstallation command syntax for each plug-in:

Plug-in	Uninstallation Command Syntax
IBM CICS Plug-in for Zowe CLI	`@brightside/cics`
IBM Db2 Plug-in for Zowe CLI	`@brightside/db2`

Example:

The following example illustrates the command to uninstall the CICS plug-in:

```
zowe plugins uninstall @brightside/cics
```

IBM® CICS® Plug-in for Zowe CLI

The IBM® CICS® Plug-in for Zowe™ CLI lets you extend Zowe CLI to interact with CICS programs and transactions. The plug-in uses the IBM CICS® Management Client Interface (CMCI) API to achieve the interaction with CICS. For more information, see CICS management client interface on the IBM Knowledge Center.

- Use cases on page 158
- Commands on page 158
- Software requirements on page 158
- Installing on page 158
- Creating a user profile on page 158

Use cases

As an application developer, you can use the plug-in to perform the following tasks:

- Deploy code changes to CICS applications that were developed with COBOL.
- Deploy changes to CICS regions for testing or delivery. See the Commands on page 158 for an example of how you can define programs to CICS to assist with testing and delivery.
- Automate CICS interaction steps in your CI/CD pipeline with Jenkins Automation Server or TravisCI.
- Deploy build artifacts to CICS regions.
- Alter, copy, define, delete, discard, and install CICS resources and resource definitions.

Commands

For detailed documentation on commands, actions, and options available in this plug-in, see our Web Help. It is available for download in three formats: a PDF document, an interactive online version, and a ZIP file containing the HTML for the online version.

- Browse Online
- Download (ZIP)
- Download (PDF)

Software requirements

Before you install the plug-in, meet the software requirements in Software requirements for Zowe CLI plug-ins on page 154.

Installing

Use one of the following methods to install or update the plug-in:

- Installing plug-ins from an online registry on page 155
- Installing plug-ins from a local package on page 155

Creating a user profile

You can set up a CICS profile to avoid typing your connection details on every command. The profile contains your host, port, username, and password for the CMCI instance of your choice. You can create multiple profiles and switch between them if necessary. Issue the following command to create a cics profile:

```
zowe profiles create cics <profile name> -H <host> -P <port> -u <user> -p
 <password>
```

Note: For more information, issue the command `zowe profiles create cis --help`

IBM® Db2® Database Plug-in for Zowe CLI

The IBM® Db2® Database Plug-in for Zowe™ CLI lets you interact with Db2 for z/OS to perform tasks through Zowe CLI and integrate with modern development tools. The plug-in also lets you interact with Db2 to advance continuous integration and to validate product quality and stability.

Zowe CLI Plug-in for IBM Db2 Database lets you execute SQL statements against a Db2 region, export a Db2 table, and call a stored procedure. The plug-in also exposes its API so that the plug-in can be used directly in other products.

[]

Use cases

As an application developer, you can use Zowe CLI Plug-in for IBM DB2 Database to perform the following tasks:

- Execute SQL and interact with databases.
- Execute a file with SQL statements.
- Export tables to a local file on your computer in SQL format.
- Call a stored procedure and pass parameters.

Commands

For detailed documentation on commands, actions, and options available in this plug-in, see our Web Help. It is available for download in three formats: a PDF document, an interactive online version, and a ZIP file containing the HTML for the online version.

- Browse Online
- Download (ZIP)
- Download (PDF)

Software requirements

Before you install the plug-in, meet the software requirements in Software requirements for Zowe CLI plug-ins on page 154.

Installing

Use one of the following methods to install the the Zowe CLI Plug-in for IBM Db2 Database:

- Installing from an online registry on page 159
- Installing from a local package on page 159

Installing from an online registry

If you installed Zowe CLI from **online registry**, complete the following steps:

1. Open a command line window and issue the following command:

```
zowe plugins install @brightside/db2@lts-incremental
```

2. Addressing the license requirement on page 160 to begin using the plug-in.

Installing from a local package

Follow these procedures if you downloaded the Zowe installation package:

Downloading the ODBC driver

Download the ODBC driver before you install the Db2 plug-in.

Follow these steps:

1. Download the ODBC CLI Driver. Use the table within the download URL to select the correct CLI Driver for your platform and architecture.
2. Create a new directory named `odbc_cli` on your computer. Remember the path to the new directory. You will need to provide the full path to this directory immediately before you install the Db2 plug-in.
3. Place the ODBC driver in the `odbc_cli` folder. **Do not extract the ODBC driver**.

You downloaded and prepared to use the ODBC driver successfully. Proceed to install the plug-in to Zowe CLI.

Installing the plug-in

Now that the Db2 ODBC CLI driver is downloaded, set the `IBM_DB_INSTALLER_URL` environment variable and install the Db2 plug-in to Zowe CLI.

Follow these steps:

1. Open a command line window and change the directory to the location where you extracted the `zowe-cli-bundle.zip` file. If you do not have the `zowe-cli-bundle.zip` file, see the topic **Install Zowe CLI from local package** in Installing Zowe CLI on page 104 for information about how to obtain and extract it.
2. From a command line window, set the `IBM_DB_INSTALLER_URL` environment variable by issuing the following command:

   - Windows operating systems:

     ```
     set IBM_DB_INSTALLER_URL=<path_to_your_odbc_folder>/odbc_cli
     ```

   - Linux and Mac operating systems:

     ```
     export IBM_DB_INSTALLER_URL=<path_to_your_odbc_folder>/odbc_cli
     ```

   For example, if you downloaded the Windows x64 driver (ntx64_odbc_cli.zip) to C:\odbc_cli, you would issue the following command:

   ```
   set IBM_DB_INSTALLER_URL=C:\odbc_cli
   ```

3. Issue the following command to install the plug-in:

   ```
   zowe plugins install zowe-db2.tgz
   ```

4. Addressing the license requirement on page 160 to begin using the plug-in.

Addressing the license requirement

The following steps are required for both the registry and offline package installation methods:

1. Locate your client copy of the Db2 license. You must have a properly licensed and configured Db2 instance for the Db2 plugin to successfully connect to Db2 on z/OS.

   Note: The license must be of version 11.1 if the Db2 server is not `db2connectactivated`. You can buy a db2connect license from IBM. The connectivity can be enabled either on server using db2connectactivate utility or on client using client side license file. To know more about DB2 license and purchasing cost, please contact IBM Customer Support.
2. Copy your Db2 license file and place it in the following directory.

   - **Windows:**

     ```
     <zowe_home>\plugins\installed\node_modules\@brightside\db2\node_modules
     \ibm_db\installer\clidriver\license
     ```

   - **Linux:**

     ```
     <zowe_home>/plugins/installed/lib/node_modules/@brightside/db2/
     node_modules/ibm_db/installer/clidriver/license
     ```

   Tip: By default, <zowe_home> is set to `~/.zowe` on *NIX systems, and `C:\Users\<Your_User>\.zowe` on Windows systems.

   After the license is copied, you can use the Db2 plugin functionality.

Creating a user profile

Before you start using the IBM Db2 plug-in, create a profile.

Issue the command `-DISPLAY DDF` in the SPUFI or ask your DBA for the following information:

- The Db2 server host name
- The Db2 server port number
- The database name (you can also use the location)
- The user name
- The password
- If your Db2 systems use a secure connection, you can also provide an SSL/TSL certificate file.

To create a db2 profile in Zowe CLI, issue the following command with your connection details for the Db2 instance:

```
zowe profiles create db2 <profile name> -H <host> -P <port> -d <database> -u
 <user> -p <password>
```

Note For more information, issue the command `zowe profiles create db2-profile --help`

Zowe Explorer Extension for VSCode

The Zowe Explorer extension for Visual Studio Code (VSCode) lets you interact with data sets, USS files and jobs that are stored on z/OS mainframe. Install the extension directly to VSCode to enable the extension within the GUI. For some users, working with data sets and USS files from VSC can be more convenient than using 3270 emulators, and complements your Zowe CLI experience.

Note: The primary documentation for the Zowe Explorer is available on the Visual Studio Code Marketplace. This article is a high-level overview of the extension.

- Use-Cases on page 161
- Software requirements on page 161
- Installing on page 161

Use-Cases

As a developer, you can use Zowe Explorer to perform the following tasks.

- View, rename, copy and filter mainframe data sets, USS files and jobs.
- Create download, edit, upload, and delete PDS and PDS members.
- Create Zowe CLI compatible `zosmf` profiles.
- Switch between Zowe CLI `zosmf` profiles to quickly target different mainframe systems.
- Submit jobs.

Software requirements

Before you use the extension, meet the following software requirements on your computer:

- Get access to z/OSMF.
- Install Node.js v8.0 or later.
- Install VSCode.
- Create one Zowe CLI `zosmf` profile so that the extension can communicate with the mainframe.

  Note: You might use an existing Zowe CLI `zosmf` profile that was created with the Zowe CLI v.2.0.0 or later.

Installing

1. Address Software requirements on page 161.
2. Open VSCode. Navigate to the **Extensions** tab on the left side of the UI.
3. Click the green **Install** button to install the extension.
4. Restart VSCode.

The extension is now installed and available for use.

Tip: For information about how to install the extension from a `VSIX` file and run system tests on the extension, see the Developer README file in the Zowe VSCode extension GitHub repository.

You can also watch the following video to learn how to get started with Zowe Explorer.

Chapter 3

Extending

Topics:

- Developing for API Mediation Layer
- Developing for Zowe CLI
- Developing for Zowe Application Framework
- Zowe Conformance Program

Developing for API Mediation Layer

Onboarding Overview

Overview of APIs

Before identifying the API you want to expose in the Zowe™ API Mediation Layer, it is useful to consider the structure of APIs. An application programming interface (API) is a set of rules that allow programs to talk to each other. A developer creates an API on a server and allows a client to talk to the API. Representational State Transfer (REST) determines the look of an API and is a set of rules that developers follow when creating an API. One of these rules states that a user should be able to get a piece of data (resource) through URL endpoints using HTTP. These resources are usually represented in the form of JSON or XML documents. The preferred documentation type in Zowe™ is in JSON format.

A REST API service can provide one or more REST APIs and usually provides the latest version of each API. A REST service is hosted on a web server which can host one or more services, often referred to as *applications*. A web server that hosts multiple services or applications is referred to as a *web application server*. Examples of *web application servers* are Apache Tomcat or WebSphere Liberty.

Note: The Zowe™ API Mediation Layer follows the OpenAPI specification. Each API has its own title, description, and version (using Semantic Versioning 2.0.0).

REST API Onboarding Enabler Guides

Services can be updated to support the API Mediation Layer natively by updating the service code. Use the following REST API onboarding enabler guides to onboard your REST service to the Zowe API Mediation Layer when you use enabler versions 1.3 and higher:

- Onboarding a REST API service with the Plain Java Enabler (PJE) on page 167
- Onboard a Java REST APIs with Spring Boot on page 188

Use the following REST API onboarding enabler guides to onboard your REST service to the API Mediation Layer when you use enabler versions 1.2 and lower:

- Onboard a Java REST APIs with Spring Boot on page 188
- Onboard a Java REST API service without Spring Boot on page 200

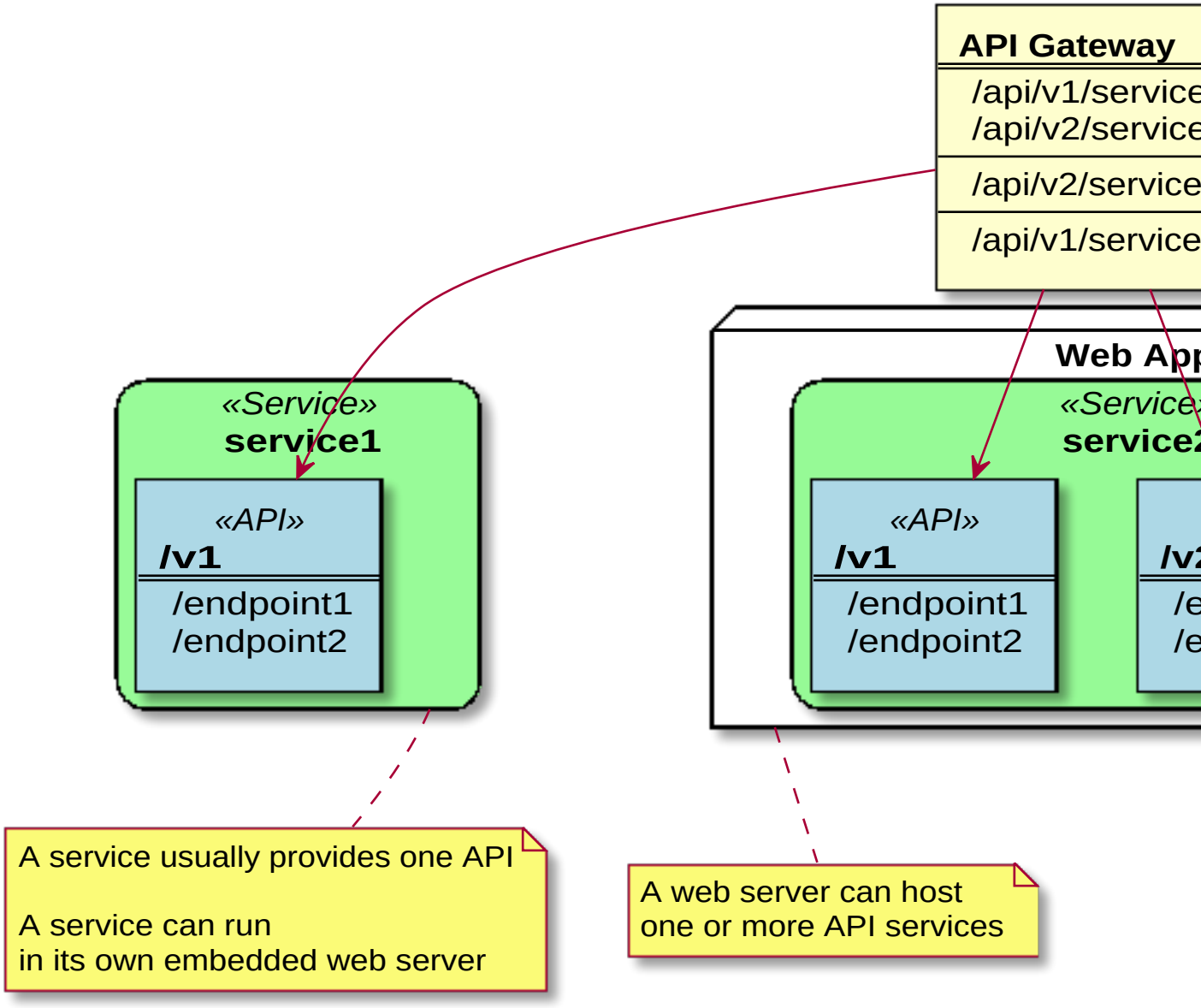
Tip: We recommend you use the enabler version 1.3 and higher to onboard your REST API service to the Zowe™ API Medaition Layer. Future fixes will not be published for enablers using version 1.2 and lower.

To onboard a REST API service to the Zowe™ API Mediation Layer without changing the code of the API service (non-natively), follow the steps in the onboarding guide to Onboard a REST API without code changes required on page 210 (static onboarding).

Tip: When developing a new service, we recommend that you update the code to support the API Mediation Layer natively. The benefit of supporting the API Mediation Layer natively is that it requires less configuration for the system administrator. Additionally, natively onboarded services can be moved to different systems, can be listened to on a different port, and additional instances can be started without the need to change API Mediation Layer configuration.

Service Relationship Diagram

The following diagram shows the relationships between various types of services, their APIs, REST API endpoints, and the API Gateway:



Sample REST API Service

In microservice architecture, a web server usually provides a single service. A typical example of a single service implementation is a Spring Boot web application.

To demonstrate the concepts that apply to REST API services, we use an example of a Spring Boot REST API service. This example is used in the REST API onboarding guide Onboard a REST API without code changes required on page 210 (static onboarding).

You can build this service using instructions in the source code of the Spring Boot REST API service example.

The Sample REST API Service has a base URL. When you start this service on your computer, the *service base URL* is: `http://localhost:8080`.

Note: If a service is deployed to a web application server, the base URL of the service (application) has the following format: `https://application-server-hostname:port/application-name`.

This sample service provides one API that has the base path `/v2`, which is represented in the base URL of the API as `http://localhost:8080/v2`. In this base URL, `/v2` is a qualifier of the base path that was chosen by the developer of this API. Each API has a base path depending on the particular implementation of the service.

This sample API has only one single endpoint:

- `/pets/{id}` - *Find pet by ID*.

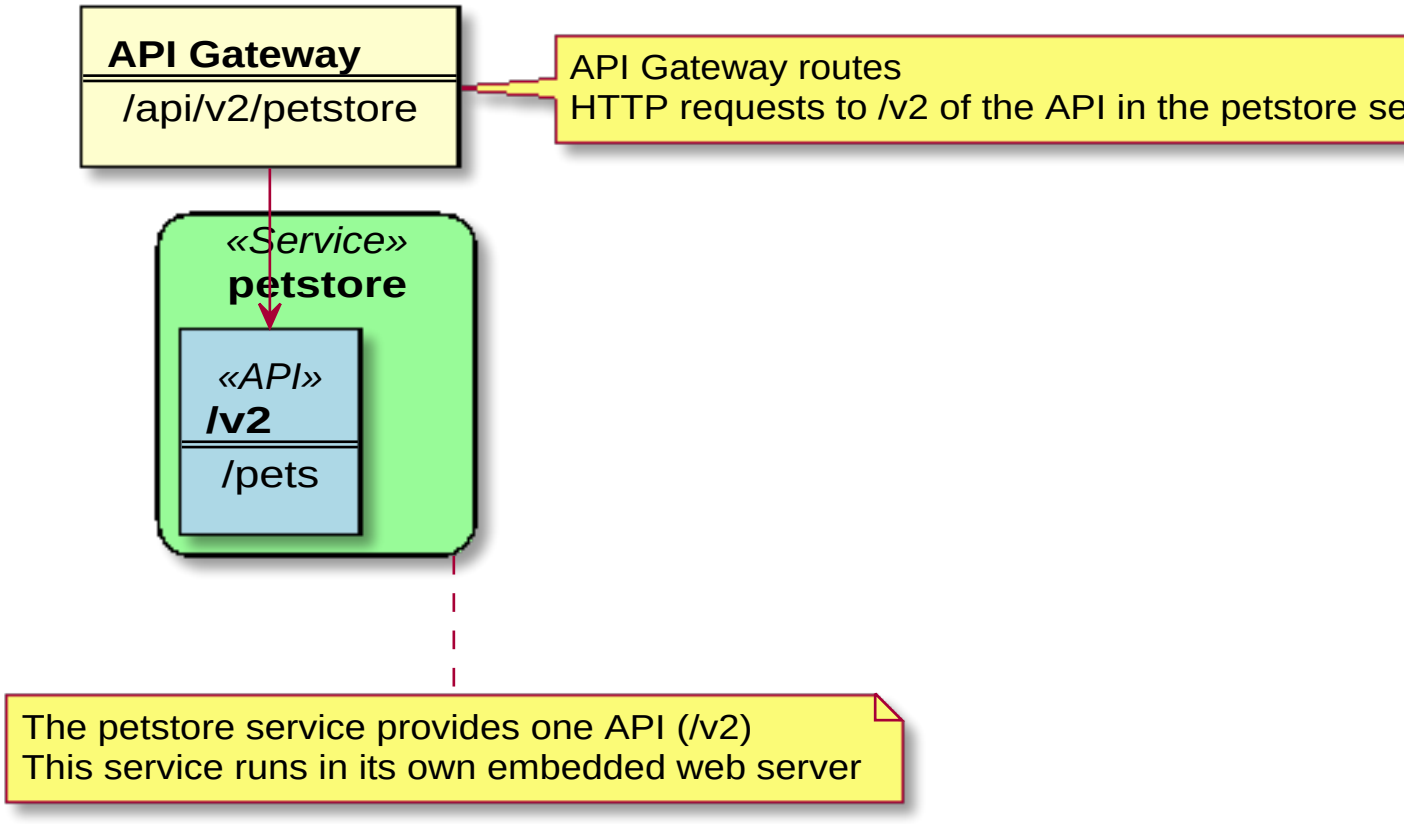
This endpoint in the sample service returns information about a pet when the `{id}` is between 0 and 10. If `{id}` is greater than 0 or a non-integer, an error is returned. These are conditions set in the sample service.

Tip: Access http://localhost:8080/v2/pets/1 to see what this REST API endpoint does. You should get the following response:

```
{
    "category": {
        "id": 2,
        "name": "Cats"
    },
    "id": 1,
    "name": "Cat 1",
    "photoUrls": [
        "url1",
        "url2"
    ],
    "status": "available",
    "tags": [
        {
            "id": 1,
            "name": "tag1"
        },
        {
            "id": 2,
            "name": "tag2"
        }
    ]
}
```

Note: The onboarding guides demonstrate how to add the Sample REST API Service to the API Mediation Layer to make the service available through the `petstore` service ID.

The following diagram shows the relations between the Sample REST API Service and its corresponding API, REST API endpoint, and API Gateway:



This sample service provides a Swagger document in JSON format at the following URL:

```
http://localhost:8080/v2/swagger.json
```

The Swagger document is used by the API Catalog to display API documentation.

Onboarding a REST API service with the Plain Java Enabler (PJE)

This article is part of a series of onboarding guides, which outline the process of onboarding REST API services to the Zowe API Mediation Layer (API ML). As a service developer, you can onboard a REST service with the API ML with the Zowe API Mediation Layer using our Plain Java Eabler (*PJE*). This enabler is built without a dependency on Spring Cloud, Spring Boot, or SpringFramework.

Tip: For more information about onboarding API services with the API ML, see the Onboarding Overview on page 164.

Introduction

Zowe API ML is a lightweight API management system based on the following Netflix components:

- Eureka - a discovery service used for services registration and discovery
- Zuul - reverse proxy / API Gateway
- Ribbon - load ballancer

The API ML Discovery Service component uses Netflix/Eureka as a REST services registry. Eureka endpoints are used to register a service with the API ML Discovery Service.

The API ML provides onboarding enabler libraries. Using these libraries is the recommended approach to onboard a REST service with the API ML. While it is possible to call the Eureka registration endpoint directly, this approach requires preparing corresponding configuration data. Doing so is unnecessarily complex and time-consuming.

Additionally, while the *PJE* library can be used in REST API projects based on SpringFramework or the Spring Boot framework, it is not recommended to use this enabler in projects that depend on SpringCloud Netflix components. Configuration settings in the *PJE* and SpringCloud Eureka Client are different. Using the two configuration settings in combination makes the result state of the discovery registry unpredictable.

Tip: For more information about how to utilize another API ML enabler, see:

- Onboard a Java REST APIs with Spring Boot on page 188
- Onboarding a service with the Zowe API Meditation Layer without an onboarding enabler on page 182
- Onboard a REST API without code changes required on page 210
- Onboard a Java REST API service without Spring Boot on page 200

Onboarding your REST service with API ML

The following steps outline the overall process to onboard a REST service with the API ML using the *PJE*. Each step is described in further detail in this article.

1. Prerequisites on page 168
2. Configuring your project on page 169
   - Gradle guide
   - Maven guide
3. Configuring your service on page 170
   - REST service identification on page 172
   - Administrative endponts
   - API info on page 173
   - API routing information
   - API Catalog information on page 174
   - API Security on page 175
   - Eureka Discovery Service on page 176
4. Registering your service with API ML on page 176
   - Add a web application context listener class
   - Register a web application context listener
   - Load service configuration
   - Initialize Eureka Client
   - Register with Eureka Discovery Service
5. Adding API documentation on page 179
6. (Optional) Validating the discoverability of your API service by the Discovery Service on page 180

Prerequisites

Ensure that the following prerequisites are met before you begin to use the *PJE* to onboard your REST service with the API ML:

- Your REST API service is written in Java.
- The service is enabled to communicate with API ML Discovery Service over a TLS v1.2 secured connection.

Notes:

- This documentation is valid for `ZoweApimlVersion 1.2.0` and higher. We recommend that you check the Giza Artifactory for newer versions.
- Following this guide enables REST services to be deployed on a z/OS environment. Deployment to a z/OS environment, however, is not required. As such, you can first develop on a local machine before you deploy on z/OS.

Configuring your project

Use either *Gradle* or *Maven* build automation systems to configure your project. Use the appropriate configuration procedure corresponding to your build automation system.

Note: You can use either the Giza Artifactory or an Artifactory of your choice. However, if you decide to build the API ML from source, you are required to publish the enabler artifact to your Artifactory. Publish the enabler artifact by using the provided *Gradle* tasks provided in the source code.

Gradle build automation system

Use the following procedure to use *Gradle* as your build automation system.

Follow these steps:

1. Create a `gradle.properties` file in the root of your project if one does not already exist.
2. In the `gradle.properties` file, set the URL of the specific Artifactory containing the *PJE* artifact. Provide the corresponding credentials to gain access to the Maven repository.

   If you are using the Giza Artifactory, use the credentials in the following code block:

   ```
   # Repository URL for getting the enabler-java artifact
   artifactoryMavenRepo=https://gizaartifactory.jfrog.io/gizaartifactory/
   libs-release

   # Artifactory credentials for builds:
   mavenUser=apilayer-build
   mavenPassword=lHj7sjJmAxL5k7obuf80Of+tCLQYZPMVpDob5oJG1NI=
   ```

3. Add the following *Gradle* code block to the `repositories` section of your `build.gradle` file:

   ```
   repositories {
       ...

       maven {
           url artifactoryMavenRepo
           credentials {
               username mavenUser
               password mavenPassword
           }
       }
   }
   ```

4. In the same `build.gradle` file, add the necessary dependencies for your service. If you use the Java enabler from the Giza Artifactory, add the following code block to your `build.gradle` script:

   ```
   implementation "org.zowe.apiml.sdk:mfaas-integration-enabler-java:
   $zoweApimlVersion"
   implementation "org.zowe.apiml.sdk:common-service-core:$zoweApimlVersion"
   ```

   Note: The published artifact from the Giza Artifactory also contains the enabler dependencies from other software packages. If you are using an Artifactory other than Giza, manually provide the following dependencies in your service `build.gradle` script:

   ```
   implementation "org.zowe.apiml.sdk:mfaas-integration-enabler-java:
   $zoweApimlVersion"
   implementation "org.zowe.apiml.sdk:common-service-core:$zoweApimlVersion"
   implementation libraries.eureka_client
   implementation libraries.httpcore
   implementation libraries.jackson_databind
   implementation libraries.jackson_dataformat_yaml

   providedCompile libraries.javax_servlet_api
   ```

```
compileOnly libraries.lombok
```

Notes:

- You may need to add more dependencies as required by your service implementation.
- The information provided in this file is valid for `ZoweApimlVersion 1.1.12` and above.

5. In your project home directory, run the `gradle clean build` command to build your project. Alternatively, you can run `gradlew` to use the specific gradle version that is working with your project.

Maven build automation system

Use the following procedure if you use *Maven* as your build automation system.

Follow these steps:

1. Add the following *XML* tags within the newly created `pom.xml` file:

```
<repositories>
    <repository>
        <id>libs-release</id>
        <name>libs-release</name>
        <url>https://gizaartifactory.jfrog.io/gizaartifactory/libs-
release</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
```

2. Create a `settings.xml` file and copy the following *XML* code block that defines the credentials for the Artifactory:

```
<?xml version="1.0" encoding="UTF-8"?>

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                  https://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
      <server>
         <id>libs-release</id>
         <username>apilayer-build</username>
         <password>lHj7sjJmAxL5k7obuf80Of+tCLQYZPMVpDob5oJG1NI=</password>
      </server>
  </servers>
</settings>
```

Tip: If you want to use *snapshot* version, set the `/servers/server/id` to `libs-snapshot`.

3. Copy the `settings.xml` file inside the `${user.home}/.m2/` directory.
4. In the directory of your project, run the `mvn package` command to build the project.

Configuring your service

Provide default service configuration in the `service-configuration.yml` file located in your service source tree resources directory.

Note: To externalize service onboarding configuration, see: API Mediation Layer onboarding configuration on page 219.

The following code snippet shows an example of `service-configuration.yml`. Some parameters values which are specific for your service deployment are written in `#{parameterValue}` format. For your service configuration file, provide actual values or externalize your onboarding configuration.

Example:

```
serviceId: sampleservice
title: Hello API ML
description: Sample API ML REST Service
baseUrl: https://${samplehost}:${sampleport}/${sampleservice}
serviceIpAddress: ${sampleHostIpAddress}

homePageRelativeUrl: /application/home
statusPageRelativeUrl: /application/info
healthCheckRelativeUrl: /application/health

discoveryServiceUrls:
    - https://${discoveryServiceHost1}:${discoveryServicePort1}/eureka
    - https://${discoveryServiceHost2}:${discoveryServicePort2}/eureka

routes:
    - gatewayUrl: api/v1
      serviceUrl: /sampleservice/api/v1

apiInfo:
    - apiId: org.zowe.sampleservice
      gatewayUrl: api/v1
      swaggerUrl: http://${sampleServiceSwaggerHost}:
${sampleServiceSwaggerPort}/sampleservice/api-doc
      doumentationUrl: http://
      version: v1
catalog:
    tile:
        id: sampleservice
        title: Hello API ML
        description: Sample application to demonstrate exposing a REST API
 in the ZOWE API ML
        version: 1.0.0

ssl:
   enabled: true
   verifySslCertificatesOfServices: true
   protocol: TLSv1.2
   ciphers: TLS_RSA_WITH_AES_128_CBC_SHA,
 TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDH_RSA_WITH
   keyAlias: localhost
   keyPassword: password
   keyStore: keystore/localhost.keystore.p12
   keyStoreType: PKCS12
   keyStorePassword: password
   trustStore: keystore/localhost.truststore.p12
   trustStoreType: PKCS12
   trustStorePassword: password
```

The onboarding configuration parameters are broken down into the following groups:

- REST service identification on page 172
- Administrative endpoints on page 172
- API info on page 173
- API routing information on page 174
- API Catalog information on page 174
- API Security on page 175
- Eureka Discovery Service on page 176

REST service identification

- **serviceId**

  The `serviceId` uniquely identifies one or more instance of a microservice in the API ML and is used as part of the service URL path in the API ML gateway address space. Additionally, the API ML Gateway uses the `serviceId` for routing to the API service instances. When two API services use the same `serviceId`, the API Gateway considers the services as clones of each other. An incoming API request can be routed to either of them through utilized load balancing mechanism.

  Important! Ensure that the `serviceId` is set properly with the following considerations:

  - The same `servicedId` should only be set for multiple API service instances for API scalability.
  - The `servicedId` value must only contain lowercase alphanumeric characters.
  - The `servicedId` cannot contain more than 40 characters.

  Example:

  - If the `serviceId` is `sampleservice`, the service URL in the API ML Gateway address space appears as the following path:

    ```
    https://gateway-host:gateway-port/api/v1/sampleservice/...
    ```

- **title**

  This parameter specifies the human readable name of the API service instance. This value is displayed in the API Catalog when a specific API service instance is selected. This parameter can be externalized and set by the customer system administrator.

  Tip: We recommend that service developer provides a default value of the `title`. Use a title that describes the service instance so that the end user knows the specific purpose of the service instance.

- **description**

  This parameter is a short description of the API service. This value is displayed in the API Catalog when a specific API service instance is selected. This parameter can be externalized and set by the customer system administrator.

  Tip: Describe the service so that the end user understands the function of the service.

- **baseUrl**

  This parameter specifies the base URL for the following administrative endpoints:

  - **homePageRelativeUrl**
  - **statusPageRelativeUrl**
  - **healthCheckRelativeUrl**

  Use the following format to include your service name in the URL path:

  `protocol://host:port/servicename`

- **serviceIpAddress** (Optional)

  This parameter specifies the IP address of the service and can be provided by system administrator in externalized service configuration. If this parameter is not present in the configuration file or is not set as a service context parameter, it will be resolved from the hostname part of the `baseUrl`.

Administrative endpoints

The following snippet presents the format of the administrative endpoint properties:

```
homePageRelativeUrl:
statusPageRelativeUrl: /application/info
healthCheckRelativeUrl: /application/health
```

where:

- **homePageRelativeUrl**

  specifies the relative path to the home page of your service. The path should start with `/`. If your service has no home page, leave this parameter blank.

  Examples:

  - `homePageRelativeUrl:` This service has no home page
  - `homePageRelativeUrl: /` This service has a home page with URL `${baseUrl}/`
- **statusPageRelativeUrl**

  specifies the relative path to the status page of your service.

  Start this path with `/`.

  Example:

  `statusPageRelativeUrl: /application/info`

  This results in the URL: `${baseUrl}/application/info`
- **healthCheckRelativeUrl**

  specifies the relative path to the health check endpoint of your service.

  Start this URL with `/`.

  Example:

  `healthCheckRelativeUrl: /application/health`

  This results in the URL: `${baseUrl}/application/health`

API info

REST services can provide multiple APIs. Add API info parameters for each API that your service wants to expose on the API ML.

The following snippet presents the information properties of a single API:

```
apiInfo:
    - apiId: org.zowe.sampleservice
    version: v1
    gatewayUrl: api/v1
    swaggerUrl: http://localhost:10021/sampleservice/api-doc
    documentationUrl: http://your.service.documentation.url
```

where:

- **apiInfo.apiId**

  specifies the API identifier that is registered in the API ML installation. The API ID uniquely identifies the API in the API ML. The `apiId` can be used to locate the same APIs that are provided by different service instances. The API developer defines this ID. The `apiId` must be a string of up to 64 characters that uses lowercase alphanumeric characters and a dot: `.`.
- **apiInfo.version**

  specifies the api `version`. This parameter is used to correctly retrieve the API documentation according to requested version of the API.
- **apiInfo.gatewayUrl**

  specifies the base path at the API Gateway where the API is available. Ensure that this value is the same path as the `gatewayUrl` value in the `routes` sections that apply to this API.
- **apiInfo.swaggerUrl** (Optional)

  specifies the Http or Https address where the Swagger JSON document is available.

- **apiInfo.documentationUrl** (Optional)

  specifies the link to the external documentation. A link to the external documentation can be included along with the Swagger documentation.

API routing information

The API routing group provides the required routing information used by the API ML Gateway when routing incoming requests to the corresponding REST API service. A single route can be used to direct REST calls to multiple resources or API endpoints. The route definition provides rules used by the API ML Gateway to rewrite the URL in the Gateway address space. Currently, the routing information consists of two parameters per route: The `gatewayUrl` and `serviceUrl`. These two parameters together specify a rule for how the API service endpoints are mapped to the API Gateway endpoints.

The following snippet is an example of the API routing information properties.

Example:

```
routes:
    - gatewayUrl: api
    serviceUrl: /sampleservice
    - gatewayUrl: api/v1
    serviceUrl: /sampleservice/api/v1
    - gatewayUrl: api/v1/api-doc
    serviceUrl: /sampleservice/api-doc
```

where:

- **routes**

  specifies the container element for the routes.
- **routes.gatewayUrl**

  The gatewayUrl parameter specifies the portion of the gateway URL which is replaced by the serviceUrl path part.
- **routes.serviceUrl**

  The serviceUrl parameter provides a portion of the service instance URL path which replaces the gatewayUrl part.

Note: The routes configuration contains a prefix before the `gatewayUrl` and `serviceUrl`. This prefix is used to differentiate the routes. It is automatically calculated by the API ML enabler.

Tip: For more information about API ML routing, see: API Gateway Routing.

API Catalog information

The API ML Catalog UI displays information about discoverable REST services registered with the API ML Discovery Service. Information displayed in the Catalog is defined by the metadata provided by your service during registration. The Catalog groups correlated services in the same tile, if these services are configured with the same `catalog.tile.id` metadata parameter.

The following code block is an example of configuration of a service tile in the Catalog:

Example:

```
   catalog:
     tile:
       id: apimediationlayer
       title:  API Mediation Layer API
       description: The API Mediation Layer for z/OS internal API services.
       version: 1.0.0
```

where:

- **catalog.tile.id**

  specifies the unique identifier for the product family of API services. This is a value used by the API ML to group multiple API services into a single tile. Each unique identifier represents a single API dashboard tile in the Catalog.

  Tip: Specify a value that does not interfere with API services from other products. We recommend that you use your company and product name as part of the ID.
- **catalog.tile.title**

  specifies the title of the product family of the API service. This value is displayed in the API Catalog dashboard as the tile title.
- **catalog.tile.description**

  is the detailed description of the API services product family. This value is displayed in the API Catalog UI dashboard as the tile description.
- **catalog.tile.version**

  specifies the semantic version of this API Catalog tile.

  Note: Ensure that you increase the version number when you introduce changes to the API service product family details.

API Security

REST services onboarded with the API ML act as both a client and a server. When communicating to API ML Discovery service, a REST service acts as a client. When the API ML Gateway is routing requests to a service, the REST service acts as a server. These two roles have different requirements. The Zowe API ML Discovery Service communicates with its clients in secure Https mode. As such, TLS/SSL configuration setup is required when a service is acting as a server. In this case, the system administrator decides if the service will communicate with its clients securely or not.

Client services need to configure several TLS/SSL parameters in order to communicate with the API ML Discovery service. When an enabler is used to onboard a service, the configuration is provided in the `ssl` section/group in the same *YAML* file that is used to configure the Eureka paramaters and the service metadata.

For more information about API ML security see: API ML security

TLS/SSL configuration consists of the following parameters:

- **verifySslCertificatesOfServices**

  This parameter makes it possible to prevent server certificate validation.

  Important! Ensure that this parameter is set to `true` in production environments. Setting this parameter to `false` in production environemnts significantly degrades the overall security of the system.
- **protocol**

  This parameter specifies the TLS protocol version currently used by Zowe API ML Discovery Service.

  Tip: We recommend you use `TLSv1.2` as your security protocol
- **keyAlias**

  This parameter specifies the `alias` used to address the private key in the keystore.
- **keyPassword**

  This parameter specifies the password associated with the private key.
- **keyStore**

  This parameter specifies the keystore file used to store the private key.
- **keyStorePassword**

  This parameter specifies the password used to unlock the keystore.

- **keyStoreType**

  This parameter specifies the type of the keystore.
- **trustStore**

  This parameter specifies the truststore file used to keep other parties public keys and certificates.
- **trustStorePassword: password**

  This parameter specifies the password used to unlock the truststore.
- **trustStoreType: PKCS12**

  This parameter specifies the truststore type. The default for this parameter is PKCS12.
- **ciphers:** (Optional)

  This parameter specifies the recommended ciphers.

  ```
  TLS_RSA_WITH_AES_128_CBC_SHA,
   TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDH_RSA_W
  ```

  To secure the transfer of data, TLS/SSL uses one or more cipher suites. A cipher suite is a combination of authentication, encryption, and message authentication code (MAC) algorithms. CIphers are used during the negotiation of security settings for a TLS/SSL connection as well as for the transfer of data.

Notes:

- Ensure that you define both the key store and the trust store even if your server is not using an Https port.
- Currently `ciphers` is not used. It is optional and serves as a place holder only.

Eureka Discovery Service

Eureka Discovery Service parameters group contains a single parameter used to address Eureka Discovery Service location. An example is presented in the following snippet:

```
discoveryServiceUrls:
- https://localhost:10011/eureka
- http://......
```

where:

- **discoveryServiceUrls**

  specifies the public URL of the Discovery Service. The system administrator at the customer site defines this parameter. It is possible to provide multiple values in order to utilize fail over and/or load balancing mechanisms.

Registering your service with API ML

The following steps outline the process of registering your service with API ML. Each step is described in detail in this article.

1. Add a web application context listener class
2. Register a web application context listener
3. Load service configuration
4. Register with Eureka discovery service
5. Unregister your service

Follow these steps:

1. Add a web application context listener class.

   The web application context listener implements two methods to perform necessary actions at application start-up time as well as when the application context is destroyed:

   - The `contextInitialized` method invokes the `apiMediationClient.register(config)` method to register the application with API Mediation Layer when the application starts.
   - The `contextDestroyed` method invokes the `apiMediationClient.unregister()` method when the application shuts down. This unregisters the application from the API Mediation Layer.
2. Register a web application context listener.

   Add the following code block to the deployment descriptor `web.xml` to register a context listener:

```
<listener>
    <listener-class>com.your.package.ApiDiscoveryListener</listener-class>
</listener>
```

3. Load the service configuration.

   Load your service configuration from a file `service-configuration.yml` file. The configuration parameters are described in the preceding section, Configuring your service on page 170.

   Use the following code as an example of how to load the service configuration.

   Example:

```
@Override
public void contextInitialized(ServletContextEvent sce) {
   ...
   String configurationFile = "/service-configuration.yml";
   ApiMediationServiceConfig config = new
 ApiMediationServiceConfigReader().loadConfiguration(configurationFile);
   ...
```

   Note: The `ApiMediationServiceConfigReader` class also provides other methods for loading the configuration from two files, `java.util.Map` instances, or directly from a string. Check the `ApiMediationServiceConfigReader` class JavaDoc for details.
4. Register with Eureka Discovery Service.

   Use the following call to register your service instance with Eureka Discovery Service:

```
  try {
      apiMediationClient = new ApiMediationClientImpl()
      apiMediationClient.register(config);
  } catch (ServiceDefinitionException sde) {
      log.error("Service configuration failed. Check log for previous
 errors: ", sde);
  }
```

5. Unregister your service.

   Use the `contextDestroyed` method to unregister your service instance from Eureka Discovery Service in the following format:

```
@Override
public void contextDestroyed(ServletContextEvent sce) {
    if (apiMediationClient != null) {
        apiMediationClient.unregister();
    }

    apiMediationClient = null;
}
```

The following code block is a full example of a context listener class implementation.

Example:

```
import org.zowe.apiml.eurekaservice.client.ApiMediationClient;
    import
 org.zowe.apiml.eurekaservice.client.config.ApiMediationServiceConfig;
    import org.zowe.apiml.eurekaservice.client.impl.ApiMediationClientImpl;
    import
 org.zowe.apiml.eurekaservice.client.util.ApiMediationServiceConfigReader;
    import org.zowe.apiml.exception.ServiceDefinitionException;
    import lombok.extern.slf4j.Slf4j;

    import javax.servlet.ServletContextEvent;
    import javax.servlet.ServletContextListener;

    /**
     *  API ML Micro service implementation of ServletContextListener
 interface.
     */
    @Slf4j
    public class ApiDiscoveryListener implements ServletContextListener {

        /**
         * @{link ApiMediationClient} instance used to register and
 unregister the service with API ML Discovery service.
         */
        private ApiMediationClient apiMediationClient;

        /**
         *  Loads a {@link ApiMediationServiceConfig} using an instance of
 class ApiMediationServiceConfigReader
         *  and registers this micro service with API ML.
         *
         *  {@link ApiMediationServiceConfigReader} has several methods for
 loading configuration from YAML file,
         *  {@link java.util.Map} or a string containing the configuration
 data.
         *
         *  Here we use the most convenient method for our Java Servlet
 based service,
         *  i.e expecting all the necessary initialization information to be
 present
         *  in the  {@link javax.servlet.ServletContext} init parameters.

         *  After successful initialization, this method creates an {@link
 ApiMediationClient} instance,
         *  which is then used to register this service with API ML
 Discovery Service.
         *
         *  The registration method of ApiMediationClientImpl catches all
 RuntimeExceptions
         *  and only can throw {@link ServiceDefinitionException} checked
 exception.
         *
         * @param sce
         */
        @Override
        public void contextInitialized(ServletContextEvent sce) {
            try {
                /*
                 * Load configuration method with ServletContext
                 */
```

```
                ApiMediationServiceConfig config = new
ApiMediationServiceConfigReader().loadConfiguration(sce.getServletContext());
                if (config  != null) {
                    /*
                     * Instantiate {@link ApiMediationClientImpl} which is
used to un/register the service with API ML Discovery Service.
                     */
                    apiMediationClient = new ApiMediationClientImpl();

                    /*
                     * Call the {@link ApiMediationClient} instance to
register your micro service with API ML Discovery Service.
                     */
                    apiMediationClient.register(config);
                }
            } catch (ServiceDefinitionException sde) {
                log.error("Service configuration failed. Check log for
previous errors: ", sde);
            }
        }

        /**
         * If apiMediationClient is not null, attmpts to unregister this
service from API ML registry.
         */
        @Override
        public void contextDestroyed(ServletContextEvent sce) {
            if (apiMediationClient != null) {
                apiMediationClient.unregister();
            }

            apiMediationClient = null;
        }
    }
```

Adding API documentation

Use the following procedure to add Swagger API documentation to your project.

Follow these steps:

1. Add a Springfox Swagger dependency.
   - For *Gradle* add the following dependency in `build.gradle`:

     ```
     compile "io.springfox:springfox-swagger2:2.8.0"
     ```

   - For *Maven* add the following dependency in `pom.xml`:

     ```
     <dependency>
         <groupId>io.springfox</groupId>
         <artifactId>springfox-swagger2</artifactId>
         <version>2.8.0</version>
     </dependency>
     ```

2. Add a Spring configuration class to your project.

   Example:

   ```
   package org.zowe.apiml.sampleservice.configuration;

   import org.springframework.context.annotation.Bean;
   import org.springframework.context.annotation.Configuration;
   import org.springframework.web.servlet.config.annotation.EnableWebMvc;
   ```

```
import
 org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

import java.util.ArrayList;

@Configuration
@EnableSwagger2
@EnableWebMvc
public class SwaggerConfiguration extends WebMvcConfigurerAdapter {
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
            .select()
            .apis(RequestHandlerSelectors.any())
            .paths(PathSelectors.any())
            .build()
            .apiInfo(new ApiInfo(
                "Spring REST API",
                "Example of REST API",
                "1.0.0",
                null,
                null,
                null,
                null,
                new ArrayList<>()
            ));
    }
}
```

3. Customize this configuration according to your specifications. For more information about customization properties, see Springfox documentation.

   Note: The current SpringFox Version 2.8 does not support OpenAPI 3.0. For more information about the open feature request see this issue.

Validating the discoverability of your API service by the Discovery Service

Once you are able to build and start your service successfully, you can use the option of validating that your service is registered correctly with the API ML Discovery Service.

Validatiing your service registration can be done in the API ML Discovery Service and the API ML Catalog. If your service appears in the Discovery Service UI but is not visible in the API Catalog, check to make sure that your configuration settings are correct.

Specific addresses and user credentials for the individual API ML components depend on your target runtime environment.

Note: If you are working with local installation of API ML and you are using our dummy identity provider, enter `user` for both `username` and `password`. If API ML was installed by system administrators, ask them to provide you with actual addresses of API ML components and the respective user credentials.

Tip: Wait for the Discovery Service to discover your service. This process may take a few minutes after your service was successfully started.

Follow these steps:

1. Use the Http `GET` method in the following format to query the Discovery Service for your service instance information:

```
http://{eureka_hostname}:{eureka_port}/eureka/apps/{serviceId}
```

2. Check your service metadata.

   Response example:

```
<application>
    <name>{serviceId}</name>
    <instanceId>{hostname}:{serviceId}:{port}</instanceId>
    <hostName>{hostname}</hostName>
    <app>{serviceId}</app>
    <ipAddr>{ipAddress}</ipAddr>
    <status>UP</status>
    <port enabled="false">{port}</port>
    <securePort enabled="true">{port}</securePort>
    <vipAddress>{serviceId}</vipAddress>
    <secureVipAddress>{serviceId}</secureVipAddress>
    <metadata>
            <apiml.service.description>Sample API service showing how to
 onboard the service</apiml.service.description>
            <apiml.routes.api__v1.gatewayUrl>api/v1</
apiml.routes.api__v1.gatewayUrl>
            <apiml.catalog.tile.version>1.0.1</apiml.catalog.tile.version>
            <apiml.routes.ws__v1.serviceUrl>/sampleclient/ws</
apiml.routes.ws__v1.serviceUrl>
            <apiml.routes.ws__v1.gatewayUrl>ws/v1</
apiml.routes.ws__v1.gatewayUrl>
            <apiml.catalog.tile.description>Applications which demonstrate
 how to make a service integrated to the API Mediation Layer ecosystem</
apiml.catalog.tile.description>
            <apiml.service.title>Sample Service ©</apiml.service.title>
            <apiml.routes.ui__v1.gatewayUrl>ui/v1</
apiml.routes.ui__v1.gatewayUrl>
            <apiml.apiInfo.0.apiId>org.zowe.sampleclient</
apiml.apiInfo.0.apiId>
            <apiml.apiInfo.0.gatewayUrl>api/v1</
apiml.apiInfo.0.gatewayUrl>
            <apiml.apiInfo.0.documentationUrl>https://www.zowe.org</
apiml.apiInfo.0.documentationUrl>
            <apiml.catalog.tile.id>samples</apiml.catalog.tile.id>
            <apiml.routes.ui__v1.serviceUrl>/sampleclient</
apiml.routes.ui__v1.serviceUrl>
            <apiml.routes.api__v1.serviceUrl>/sampleclient/api/v1</
apiml.routes.api__v1.serviceUrl>
            <apiml.apiInfo.0.swaggerUrl>https://hostname/sampleclient/api-
doc</apiml.apiInfo.0.swaggerUrl>
            <apiml.catalog.tile.title>Sample API Mediation Layer
 Applications</apiml.catalog.tile.title>
    </metadata>
</application>
```

3. Check that your API service is displayed in the API Catalog and all information including API documentation is correct.
4. Check that you can access your API service endpoints through the Gateway.
5. (Optional) Check that you can access your API service endpoints directly outside of the Gateway.

Onboarding a service with the Zowe API Meditation Layer without an onboarding enabler

This article is part of a series of guides to onboard a REST service with the Zowe API Mediation Layer (API ML). Onboarding with API ML makes services accessible through the API Gateway and visible in the API Catalog. Once a service is successfully onboarded, users can see if the service is currently available and accepting requests.

This guide describes how a REST service can be onboarded with the Zowe API ML independent of the language used to write the service. As such, this guide does not describe how to onboard a service with a specific enabler. Similarly, various Eureka client implementations are not used in this onboarding method.

Tip: If possible, we recommend that you onboard your service using the API ML enabler libraries. The approach described in this article should only be used if other methods to onboard your service are not suitable.

For more information about how to onboard a REST service, see the following links:

- Onboarding Overview on page 164
- python-eureka-client
- eureka-js-client
- Rest API developed based on Java

This article outlines a process to make an API service available in the API Mediation Layer by making a direct call to the Eureka Discovery Service.

- Introduction on page 182
- Registering with the Discovery Service
  - API Mediation Layer Service onboarding metadata
    - Catalog parameters
    - Service parameters
    - Routing parameters
    - API Info Parameters
- Sending a heartbeat to API Mediation Layer Discovery Service
- Validating successful onboarding with the API Mediation Layer
- External Resources

Introduction

The API ML Discovery Service uses Netflix/Eureka as a REST services registry. Eureka is a REST-based service that is primarily used to locate services.

Eureka endpoints are used to register a service with the API ML Discovery Service. Endpoints are also used to send a periodic heartbeat to the Discovery Service to indicate that the onboarded service is available.

Note: Required parameters should be defined and sent at registration time.

Registering with the Discovery Service

Begin the onboarding process by registering your service with the API ML Discovery Service.

Use the `POST` Http call to the Eureka server together with the registration configuration in the following format:

```
https://{eureka_hostname}:{eureka_port}/eureka/apps/{serviceId}
```

The following code block shows the format of the parameters in your `POST` call, which are sent to the Eureka registry at the time of registration.

```
<?xml version="1.0" ?>
<instance>
  <app>{serviceId}</app>
  <ipAddr>{ipAddress}</ipAddr>
```

```
   <port enabled="false">{port}</port>
   <securePort enabled="true">{port}</securePort>
   <hostName>{hostname}</hostName>
   <vipAddress>{serviceId}</vipAddress>
   <secureVipAddress>{serviceId}</secureVipAddress>
   <instanceId>{instanceId}</instanceId>
   <dataCenterInfo>
     <name>MyOwn</name>
   </dataCenterInfo>
   <metadata>
         ...
    </metadata>
</instance>
```

where:

- **app**

  uniquely identifies one or more instances of a microservice in the API ML.

  The API ML Gateway uses the `serviceId` for routing to the API service instances. As such, the `serviceId` is part of the service URL path in the API ML Gateway address space.

  Important! Ensure that the service ID is set properly with the following considerations:

  - The service ID value contains only lowercase alphanumeric characters.
  - The service ID does not contain more than 40 characters.
  - The same service ID is only set for multiple API service instances to support API scalability. When two API services use the same service ID, the API Gateway considers the services as clones of each other. An incoming API request can be routed to either of them through load balancing.

  Example:

  - If the `serviceId` is `sampleservice`, the service URL in the API ML Gateway address space appears as:

    ```
    https://gateway-host:gateway-port/api/v1/sampleservice/...
    ```

- **ipAddr**

  specifies the IP address of this specific service instance.

- **port**

  specifies the port of the instance when you use Http. For Http, set `enabled` to `true`.

- **securePort**

  specifies the port of the instance for when you use Https. For Https, set `enabled` to `true`.

- **hostname**

  specifies the hostname of the instance.

- **vipAddress**

  specifies the `serviceId` when you use Http.

  Important! Ensure that the value of `vipAddress` is the same as the value of `app`.

- **secureVipAddress**

  specifies the `serviceId` when you use Https.

  Important! Ensure that the value of `secureVipAddress` is the same as the value of `app`.

- **instanceId**

  specifies a unique id for the instance. Define a unique value for the `instanceId` in the following format:

  `{hostname}:{serviceId}:{port}`

- **metadata**

  specifies the set of parameters described in the following section addressing API ML service metadata.

API Mediation Layer Service onboarding metadata

At registration time, provide metadata in the following format. Metadata parameters contained in this code block are described in the following section.

```
<instance>
  <metadata>
      <apiml.catalog.tile.id>samples</apiml.catalog.tile.id>
      <apiml.catalog.tile.title>Sample API Mediation Layer Applications</
apiml.catalog.tile.title>
      <apiml.catalog.tile.description>Applications which demonstrate
 how to make a service integrated to the API Mediation Layer ecosystem</
apiml.catalog.tile.description>
      <apiml.catalog.tile.version>1.0.1</apiml.catalog.tile.version>
      <apiml.service.title>Sample Service ©</apiml.service.title>
      <apiml.service.description>Sample API service showing how to onboard
 the service</apiml.service.description>
      <apiml.routes.api__v1.gatewayUrl>api/v1</
apiml.routes.api__v1.gatewayUrl>
      <apiml.routes.api__v1.serviceUrl>/sampleclient/api/v1</
apiml.routes.api__v1.serviceUrl>
      <apiml.routes.ui__v1.serviceUrl>/sampleclient</
apiml.routes.ui__v1.serviceUrl>
      <apiml.routes.ui__v1.gatewayUrl>ui/v1</apiml.routes.ui__v1.gatewayUrl>
      <apiml.routes.ws__v1.gatewayUrl>ws/v1</apiml.routes.ws__v1.gatewayUrl>
      <apiml.routes.ws__v1.serviceUrl>/sampleclient/ws</
apiml.routes.ws__v1.serviceUrl>
      <apiml.apiInfo.0.apiId>org.zowe.sampleclient</apiml.apiInfo.0.apiId>
      <apiml.apiInfo.0.swaggerUrl>https://hostname/sampleclient/api-doc</
apiml.apiInfo.0.swaggerUrl>
      <apiml.apiInfo.0.gatewayUrl>api/v1</apiml.apiInfo.0.gatewayUrl>
      <apiml.apiInfo.0.documentationUrl>https://www.zowe.org</
apiml.apiInfo.0.documentationUrl>
   </metadata>
</instance>
```

Metadata parameters are broken down into the following categories:

- Catalog parameters
- Service parameters
- Routing parameters
- API Info parameters

Catalog parameters

Catalog parameters are grouped under the prefix: `apiml.catalog.tile`.

The API ML Catalog displays information about services registered with the API ML Discovery Service. Information displayed in the Catalog is defined in the metadata provided by your service during registration. The Catalog groups correlated services in the same tile when these services are configured with the same `catalog.tile.id` metadata parameter.

The following parameters are used to populate the API Catalog:

- **apiml.catalog.tile.id**

  This parameter specifies the specific identifier for the product family of API services. This is a value used by the API ML to group multiple API services into a single tile. Each identifier represents a single API dashboard tile in the Catalog.

  Important! Specify a value that does not interfere with API services from other products. We recommend that you use your company and product name as part of the ID.
- **apiml.catalog.tile.title**

  This parameter specifies the title of the API services product family. This value is displayed in the API Catalog dashboard as the tile title.
- **apiml.catalog.tile.description**

  This parameter is the detailed description of the API services product family. This value is displayed in the API Catalog UI dashboard as the tile description.
- **apiml.catalog.tile.version**

  This parameter specifies the semantic version of this API Catalog tile.

  Note: Ensure that you increase the version number when you introduce changes to the API service product family details.

Service parameters

Service parameters are grouped under the prefix: `apiml.service`

The following parameters define service information for the API Catalog:

- **apiml.service.title**

  This parameter specifies the human-readable name of the API service instance.

  This value is displayed in the API Catalog when a specific API service instance is selected.
- **apiml.service.description**

  This parameter specifies a short description of the API service.

  This value is displayed in the API Catalog when a specific API service instance is selected.

Routing parameters

Routing parameters are grouped under the prefix: `apiml.routes`

The API routing group provides necessary routing information used by the API ML Gateway when routing incoming requests to the corresponding service. A single route can be used to make direct REST calls to multiple resources or API endpoints. The route definition provides rules used by the API ML Gateway to rewrite the URL in the Gateway address space.

Routing information consists of two parameters per route:

- `gatewayUrl`
- `serviceUrl`

These two parameters together specify a rule of how the API service endpoints are mapped to the API Gateway endpoints.

The following snippet is an example of the API routing information properties.

Example:

```
<apiml.routes.api__v1.gatewayUrl>api/v1</apiml.routes.api__v1.gatewayUrl>
<apiml.routes.api__v1.serviceUrl>/sampleclient/api/v1</
apiml.routes.api__v1.serviceUrl>
```

where:

- **apiml.routes.{route-prefix}.gatewayUrl**

  The `gatewayUrl` parameter specifies the portion of the gateway URL which is replaced by the `serviceUrl` path.
- **apiml.routes.{route-prefix}.serviceUrl**

  The `serviceUrl` parameter provides a portion of the service instance URL path which replaces the `gatewayUrl` part.

Note: The routes configuration used for a direct REST call to register a service must also contain a prefix before the `gatewayUrl` and `serviceUrl`. This prefix is used to differentiate the routes. This prefix must be provided manually when *XML* configuration is used.

For more information about API ML routing, see API Gateway Routing.

API Info parameters

API Info parameters are grouped under the prefix: `apiml.apiInfo`.

REST services can provide multiple APIs. Add API info parameters for each API that your service wants to expose on the API ML. These parameters provide information for API (Swagger) documentation that is displayed in the API Catalog.

The following parameters provide the information properties of a single API:

- **apiml.apiInfo.{api-index}.apiId**

  The API ID uniquely identifies the API in the API ML. Multiple services can provide the same API. The API ID can be used to locate the same APIs that are provided by different services. The creator of the API defines this ID. The API ID needs to be a string of up to 64 characters that uses lowercase alphanumeric characters and a dot: `.`.

  Tip: We recommend that you use your organization as the prefix.
- **apiml.apiInfo.{api-index}.version**

  This parameter specifies the API version. This parameter is used to correctly retrieve the API documentation according to the requested version of the API.
- **apiml.apiInfo.{api-index}.gatewayUrl**

  This parameter specifies the base path at the API Gateway where the API is available. Ensure that this value is the same path as the `gatewayUrl` value in the `routes` sections for the routes, which belong to this API.
- **apiml.apiInfo.{api-index}.swaggerUrl**

  (Optional) This parameter specifies the Http or Https address where the Swagger JSON document is available.
- **apiml.apiInfo.{api-index}.documentationUrl**

  (Optional) This parameter specifies the link to the external documentation. A link to the external documentation can be included along with the Swagger documentation.

Note: The `{api-index}` is used to differentiate the service APIs. This index must be provided manually when *XML* configuration is used. In the following example, `0` represents the `api-index`.

```
<apiml.apiInfo.0.apiId>org.zowe.sampleclient</apiml.apiInfo.0.apiId>
<apiml.apiInfo.0.swaggerUrl>https://hostname/sampleclient/api-doc</
apiml.apiInfo.0.swaggerUrl>
<apiml.apiInfo.0.gatewayUrl>api/v1</apiml.apiInfo.0.gatewayUrl>
<apiml.apiInfo.0.documentationUrl>https://www.zowe.org</
apiml.apiInfo.0.documentationUrl>
```

Sending a heartbeat to API Meditation Layer Discovery Service

After registration, a service must send a heartbeat periodically to the Discovery Service to indicate that the service is available. When the Discovery Service does not receive a heartbeat, the service instance is deleted from the Discovery Service.

If the server does not receive a renewal in 90 seconds, it removes the instance from its registry.

Note: We recommend that the interval for the heartbeat is no more than 30 seconds.

Use the Http `PUT` method in the following format to tell the Discovery Service that your service is available:

```
https://{eureka_hostname}:{eureka_port}/eureka/apps/{serviceId}/{instanceId}
```

Validating successful onboarding with the API Meditation Layer

Ensure that you successfully onboarded a service with the API Mediation Layer.

Follow these steps:

1. In your Http client such as HTTPie, Postman, or cURL use the Http `GET` method in the following format to query the Discovery Service for your service instance information:

   ```
   http://{eureka_hostname}:{eureka_port}/eureka/apps/{serviceId}
   ```

2. Check your service metadata.

   Response example:

   ```
   <application>
       <name>{serviceId}</name>
       <instanceId>{hostname}:{serviceId}:{port}</instanceId>
       <hostName>{hostname}</hostName>
       <app>{serviceId}</app>
       <ipAddr>{ipAddress}</ipAddr>
       <status>UP</status>
       <port enabled="false">{port}</port>
       <securePort enabled="true">{port}</securePort>
       <vipAddress>{serviceId}</vipAddress>
       <secureVipAddress>{serviceId}</secureVipAddress>
       <metadata>
               <apiml.service.description>Sample API service showing how to
    onboard the service</apiml.service.description>
               <apiml.routes.api__v1.gatewayUrl>api/v1</
   apiml.routes.api__v1.gatewayUrl>
               <apiml.catalog.tile.version>1.0.1</apiml.catalog.tile.version>
               <apiml.routes.ws__v1.serviceUrl>/sampleclient/ws</
   apiml.routes.ws__v1.serviceUrl>
               <apiml.routes.ws__v1.gatewayUrl>ws/v1</
   apiml.routes.ws__v1.gatewayUrl>
               <apiml.catalog.tile.description>Applications which demonstrate
    how to make a service integrated to the API Mediation Layer ecosystem</
   apiml.catalog.tile.description>
               <apiml.service.title>Sample Service ©</apiml.service.title>
               <apiml.routes.ui__v1.gatewayUrl>ui/v1</
   apiml.routes.ui__v1.gatewayUrl>
               <apiml.apiInfo.0.apiId>org.zowe.sampleclient</
   apiml.apiInfo.0.apiId>
               <apiml.apiInfo.0.gatewayUrl>api/v1</
   apiml.apiInfo.0.gatewayUrl>
               <apiml.apiInfo.0.documentationUrl>https://www.zowe.org</
   apiml.apiInfo.0.documentationUrl>
               <apiml.catalog.tile.id>samples</apiml.catalog.tile.id>
               <apiml.routes.ui__v1.serviceUrl>/sampleclient</
   apiml.routes.ui__v1.serviceUrl>
               <apiml.routes.api__v1.serviceUrl>/sampleclient/api/v1</
   apiml.routes.api__v1.serviceUrl>
               <apiml.apiInfo.0.swaggerUrl>https://hostname/sampleclient/api-
   doc</apiml.apiInfo.0.swaggerUrl>
               <apiml.catalog.tile.title>Sample API Mediation Layer
    Applications</apiml.catalog.tile.title>
       </metadata>
   ```

```
</application>
```

3. Check that your API service is displayed in the API Catalog and all information including API documentation is correct.
4. Check that you can access your API service endpoints through the Gateway.
5. (Optional) Check that you can access your API service endpoints directly outside of the Gateway.

External Resources

- https://blog.asarkar.org/technical/netflix-eureka/
- https://medium.com/@fahimfarookme/the-mystery-of-eureka-health-monitoring-5305e3beb6e9
- https://github.com/Netflix/eureka/wiki/Eureka-REST-operations

Onboard a Java REST APIs with Spring Boot

As an API developer, use this guide to onboard your REST API service to the Zowe™ API Mediation Layer (API ML). This article outlines a step-by-step process to make your API service available in the Zowe™ API ML.

Notes:

- The following guide is for enabler versions 1.2 and lower. To onboard a Spring Boot based REST API service for enabler versions 1.3 and higher, see Onboard a Java REST APIs with Spring Boot on page 188.
- Spring is a Java-based framework that lets you build web and enterprise applications. For more information, see the Spring website.

The following procedure outlines the general steps to onboard a REST service using Spring Boot to the Zowe™ API ML. Each of these steps is described in more detail in this article.

1. Add Zowe API enablers to your service on page 188
2. Add API ML onboarding configuration on page 190
3. Externalize API ML configuration parameters on page 199
4. Test your service on page 200
5. Review the configuration examples of the discoverable client on page 200

Add Zowe API enablers to your service

The general steps to onboard a REST API to the Zowe ecosystem begins with adding the Zowe Artifactory repository definition to the list of repositories. Then add the Spring enabler to the list of your dependencies. Finally, add enabler annotations to your service code. Enablers prepare your service for discovery and swagger documentation retrieval.

Follow these steps:

1. Add the Zowe Artifactory repository definition to the list of repositories in *Gradle* or *Maven* build systems. Use the code block that corresponds to your build system.

   - In a *Gradle* build system, add the following code to the `build.gradle` file into the `repositories` block.

     ```
     maven {
       url 'https://zowe.jfrog.io/zowe/libs-release'
     }
     ```

     Note: You can define the `gradle.properties` file where you can set your username, password, and the read-only repo URL for access to the Zowe Artifactory. By defining the `gradle.properties`, you do not need to hardcode the username, password, and read-only repo URL in your `gradle.build` file.

     Example:

     ```
        # Artifactory repositories for builds
     ```

```
   artifactoryMavenRepo=https://zowe.jfrog.io/zowe/libs-release
```

- In a *Maven* build system, follow these steps:

  a) Add the following code to the `pom.xml` file:

```
<repository>
       <id>Zowe</id>
       <url>https://zowe.jfrog.io/zowe/libs-release</url>
</repository>
```

  b) Create a `settings.xml` file and copy the following XML code block which defines the login credentials for the Zowe Artifactory. Use valid credentials.

```
<?xml version="1.0" encoding="UTF-8"?>

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                  https://maven.apache.org/xsd/settings-1.0.0.xsd">
<servers>
   <server>
       <id>Zowe</id>
   </server>
</servers>
</settings>
```

  c) Copy the `settings.xml` file inside the `${user.home}/.m2/` directory.

2. Add a JAR package to the list of dependencies in *Gradle* or *Maven* build systems. Zowe API Mediation Layer supports Spring Boot versions 1.5.9 and 2.0.4.

- If you use Spring Boot release 1.5.x in a *Gradle* build system, add the following code to the `build.gradle` file into the `dependencies` block:

```
   compile group: 'org.zowe.apiml.sdk', name: 'mfaas-integration-enabler-
spring-v1-springboot-1.5.9.RELEASE', version: '1.1.0'
```

- If you use Spring Boot release 1.5.x in a *Maven* build system, add the following code to the `pom.xml` file:

```
   <dependency>
         <groupId>org.zowe.apiml.sdk</groupId>
         <artifactId>mfaas-integration-enabler-spring-v1-
springboot-1.5.9.RELEASE</artifactId>
         <version>1.1.0</version>
   </dependency>
```

- If you use the Spring Boot release 2.0.x in a *Gradle* build system, add the following code to the `build.gradle` file into the `dependencies` block:

```
compile group: 'org.zowe.apiml.sdk', name: 'mfaas-integration-enabler-
spring-v2-springboot-2.0.4.RELEASE', version: '1.1.0'
```

- If you use the Spring Boot release 2.0.x in a *Maven* build system, add the following code to the `pom.xml` file:

```
<dependency>
       <groupId>org.zowe.apiml.sdk</groupId>
       <artifactId>mfaas-integration-enabler-spring-v2-
springboot-2.0.4.RELEASE</artifactId>
       <version>1.1.0</version>
</dependency>
```

3. Add the following annotations to the main class of your Spring Boot, or add these annotations to an extra Spring configuration class:

   - `@ComponentScan({"org.zowe.apiml.enable", "org.zowe.apiml.product"})`

     This annotation makes an API documentation endpoint visible within the Spring context.
   - `@EnableApiDiscovery`

     This annotation exposes your Swagger (OpenAPI) documentation in the Zowe ecosystem to make your micro service discoverable in the Zowe ecosystem.

     Note: The `@EnableApiDiscovery` annotation uses the Spring Fox library. If your service uses the library already, some fine tuning may be necessary.

   Example:

```
package org.zowe.apiml.DiscoverableClientSampleApplication;
..
import org.zowe.apiml.enable.EnableApiDiscovery;
import org.springframework.context.annotation.ComponentScan;
..
@EnableApiDiscovery
@ComponentScan({"org.zowe.apiml.enable", "org.zowe.apiml.product"})
...
public class DiscoverableClientSampleApplication {...
```

You are now ready to build your service to include the code pieces that make it discoverable in the API Mediation Layer and to add Swagger documentation.

Add API ML onboarding configuration

As an API service developer, you set multiple configuration settings in your application.yml that correspond to the API ML. These settings enable an API to be discoverable and included in the API catalog. Some of the settings in the application.yml are internal and are set by the API service developer. Some settings are externalized and set by the customer system administrator. Those external settings are service parameters and are in the format: `${environment.*}`.

Important! Spring Boot configuration can be externalized in multiple ways. For more information see: Externalized configuration. This Zowe onboarding documentation applies to API services that use an `application.yml` file for configuration. If your service uses a different configuration option, transform the provided configuration sample to the format that your API service uses.

Tip: For information about how to set your configuration when running a Spring Boot application under an external servlet container (*TomCat*), see the following short *stackoverflow* article: External configuration for spring-boot application.

Follow these steps:

1. Add the following #MFAAS configuration section in your `application.yml`:

```
 ##########################################################################################
  # MFAAS configuration section

 ##########################################################################################
  mfaas:
      discovery:
          serviceId: ${environment.serviceId}
          locations: ${environment.discoveryLocations}
          enabled: ${environment.discoveryEnabled:true}
          endpoints:
              statusPage: ${mfaas.server.scheme}://
${mfaas.service.hostname}:${mfaas.server.port}${mfaas.server.contextPath}/
application/info
```

```
              healthPage: ${mfaas.server.scheme}://
${mfaas.service.hostname}:${mfaas.server.port}${mfaas.server.contextPath}/
application/health
              homePage: ${mfaas.server.scheme}://
${mfaas.service.hostname}:${mfaas.server.port}${mfaas.server.contextPath}/
          info:
              serviceTitle:  ${environment.serviceTitle}
              description:  ${environment.serviceDescription}
              # swaggerLocation:
 resource_location_of_your_static_swagger_doc.json
          fetchRegistry: false
          region: default
      service:
          hostname: ${environment.hostname}
          ipAddress: ${environment.ipAddress}
      catalog-ui-tile:
          id: yourProductFamilyId
          title: Your API service product family title in the API catalog
 dashboard tile
          description: Your API service product family description in the
 API catalog dashboard tile
          version:  1.0.0
      server:
          scheme: http
          port: ${environment.port}
          contextPath: /yourServiceUrlPrefix
      security:
          sslEnabled: true
          protocol: TLSv1.2
          ciphers:
 TLS_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDH_RSA_WITH_AES_
          keyAlias: localhost
          keyPassword: password
          keyStore: keystore/ssl_local/localhost.keystore.p12
          keyStoreType: PKCS12
          keyStorePassword: password
          trustStore: keystore/ssl_local/localhost.truststore.p12
          trustStoreType: PKCS12
          trustStorePassword: password

  eureka:
      instance:
          appname: ${mfaas.discovery.serviceId}
          hostname: ${mfaas.service.hostname}
          statusPageUrlPath: ${mfaas.discovery.endpoints.statusPage}
          healthCheckUrl: ${mfaas.discovery.endpoints.healthPage}
          homePageUrl: ${mfaas.discovery.endpoints.homePage}
          metadata-map:
              routed-services:
                  api_v1:
                      gateway-url: "api/v1"
                      service-url: ${mfaas.server.contextPath}
              apiml:
                  apiInfo:
                      - apiId: ${mfaas.discovery.serviceId}
                        gatewayUrl: api/v1
                        swaggerUrl: ${mfaas.server.scheme}://
${mfaas.service.hostname}:${mfaas.server.port}${mfaas.server.contextPath}/
api-doc
                        documentationUrl: https://www.zowe.org
              mfaas:
                  api-info:
                      apiVersionProperties:
                          v1:
```

```
                                title: Your API title for swagger JSON which
 is displayed in API Catalog / service / API Information
                                description: Your API description for
 swagger JSON
                                version: 1.0.0
                                basePackage:
 your.service.base.package.for.swagger.annotated.controllers
                                # apiPattern: /v1/.*  # alternative to
 basePackage for exposing endpoints which match the regex pattern to
 swagger JSON
                    discovery:
                        catalogUiTile:
                            id: ${mfaas.catalog-ui-tile.id}
                            title:  ${mfaas.catalog-ui-tile.title}
                            description: ${mfaas.catalog-ui-
tile.description}
                            version: ${mfaas.catalog-ui-tile.version}
                        enableApiDoc:
 ${mfaas.discovery.info.enableApiDoc:true}
                        service:
                            title: ${mfaas.discovery.info.serviceTitle}
                            description: ${mfaas.discovery.info.description}
      client:
          enabled: ${mfaas.discovery.enabled}
          healthcheck:
              enabled: true
          serviceUrl:
              defaultZone: ${mfaas.discovery.locations}
          fetchRegistry:  ${mfaas.discovery.fetchRegistry}
          region: ${mfaas.discovery.region}


 ##########################################################################################
  # Application configuration section

 ##########################################################################################
  server:
      # address: ${mfaas.service.ipAddress}
      port: ${mfaas.server.port}
      servlet:
          contextPath: ${mfaas.server.contextPath}
      ssl:
          enabled: ${mfaas.security.sslEnabled}
          protocol: ${mfaas.security.protocol}
          ciphers: ${mfaas.security.ciphers}
          keyStore: ${mfaas.security.keyStore}
          keyAlias: ${mfaas.security.keyAlias}
          keyPassword: ${mfaas.security.keyPassword}
          keyStorePassword: ${mfaas.security.keyStorePassword}
          keyStoreType: ${mfaas.security.keyStoreType}
          trustStore: ${mfaas.security.trustStore}
          trustStoreType: ${mfaas.security.trustStoreType}
          trustStorePassword: ${mfaas.security.trustStorePassword}

  spring:
      application:
          name: ${mfaas.discovery.serviceId}
```

In order to run your application locally, you need to define variables used under the `environment` group.

```
##########################################################################################
# Local configuration section
##########################################################################################
```

```
environment:
    serviceId: Your service id
    serviceTitle: Your service title
    serviceDescription: Your service description
    discoveryEnabled: true
    hostname: localhost
    port: Your service port
    discoveryLocations: https://localhost:10011/eureka/
    ipAddress: 127.0.0.1
```

Important: Add this configuration also to the `application.yml` used for testing. Failure to add this configuration to the `application.yml` will cause your tests to fail.

2. Change the MFaaS parameters to correspond to your API service specifications. Most of these internal parameters contain "your service" text.

   Note: `${mfaas.*}` variables are used throughout the `application.yml` sample to reduce the number of required changes.

   Tip: When existing parameters set by the system administrator are already present in your configuration file (for example, `hostname, address, contextPath,` and `port`), we recommend that you replace them with the corresponding MFaaS properties.

   a. **Discovery Parameters**

   - **mfaas.discovery.serviceId**

     This parameter specifies the service instance identifier to register in the API ML installation. The service ID is used in the URL for routing to the API service through the gateway. The service ID uniquely identifies

instances of a microservice in the API ML. The system administrator at the customer site defines this parameter.

Important! Ensure that the service ID is set properly with the following considerations:

- When two API services use the same service ID, the API Gateway considers the services to be clones. An incoming API request can be routed to either of them.
- The same service ID should be set for only multiple API service instances for API scalability.
- The service ID value must contain only lowercase alphanumeric characters.
- The service ID cannot contain more than 40 characters.
- The service ID is linked to security resources. Changes to the service ID require an update of security resources.
- The service ID must match the `spring.application.name` parameter.

Examples:

- If the customer system administrator sets the service ID to `sysviewlpr1`, the API URL in the API Gateway appears as the following URL:

  ```
  https://gateway:port/api/v1/sysviewlpr1/endpoint1/...
  ```

- If the customer system administrator sets the service ID to `vantageprod1`, the API URL in the API Gateway appears as the following URL:

  ```
  http://gateway:port/api/v1/vantageprod1/endpoint1/...
  ```

- **mfaas.discovery.locations**

  This parameter specifies the public URL of the Discovery Service. The system administrator at the customer site defines this parameter.

  Example:

  ```
  http://eureka:password@141.202.65.33:10311/eureka/
  ```

- **mfaas.discovery.enabled**

  This parameter specifies whether the API service instance is to be discovered in the API ML. The system administrator at the customer site defines this parameter. Set this parameter to `true` if the API ML is installed

and configured. Otherwise, you can set this parameter to `false` to exclude an API service instances from the API ML.

- **mfaas.discovery.fetchRegistry**

  This parameter specifies whether the API service is to receive regular update notifications from the discovery service. Under most circumstances, you can accept the default value of `false` for the parameter.

- **mfaas.discovery.region**

  This parameter specifies the geographical region. This parameter is required by the Discovery client. Under most circumstances you can accept the value `default` for the parameter.

b. **Service and Server Parameters**

- **mfaas.service.hostname**

  This parameter specifies the hostname of the system where the API service instance runs. This parameter is externalized and is set by the customer system administrator. The administrator ensures the hostname can be resolved by DSN to the IP address that is accessible by applications running on their z/OS systems.

- **mfaas.service.ipAddress**

  This parameter specifies the local IP address of the system where the API service instance runs. This IP address may or may not be a public IP address. This parameter is externalized and set by the customer system administrator.

- **mfaas.server.scheme**

  This parameter specifies whether the API service is using the HTTPS protocol. This value can be set to `https` or `http` depending on whether your service is using SSL.

- **mfaas.server.port**

  This parameter specifies the port that is used by the API service instance. This parameter is externalized and set by the customer system administrator.

- **mfaas.server.contextPath**

  This parameter specifies the prefix that is used within your API service URL path.

  Examples:

  - If your API service does not use an extra prefix in the URL (for example, `http://host:port/endpoint1/`), set this value to `/`.
  - If your API service uses an extra URL prefix set the parameter to that prefix value. For the URL: `http://host:port/filemaster/endpoint1/`, set this parameter to `/filemaster`.
  - In both examples, the API service URL appears as the following URL when routed through the Gateway:

    ```
    http://gateway:port/serviceId/endpoint1/
    ```

c. **API Catalog Parameters**

These parameters are used to populate the API Catalog. The API Catalog contains information about every registered API service. The Catalog also groups related APIs. Each API group has its own name and description. Catalog groups are constructed in real-time based on information that is provided by the API services. Each group is displayed as a tile in the API Catalog UI dashboard.

- **mfaas.catalog-ui-tile.id**

  This parameter specifies the unique identifier for the API services product family. This is the grouping value used by the API ML to group multiple API services together into "tiles". Each unique identifier represents

a single API Catalog UI dashboard tile. Specify a value that does not interfere with API services from other products.

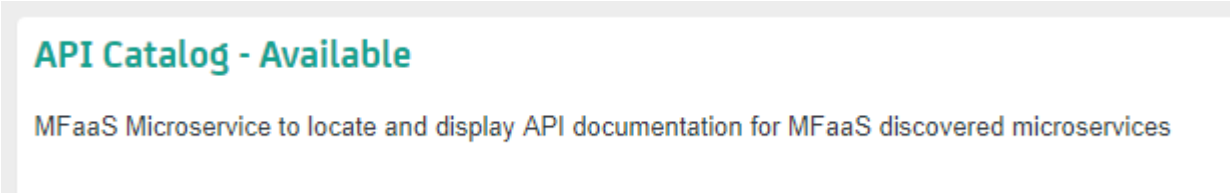
- **mfaas.catalog-ui-tile.title**

  This parameter specifies the title of the API services product family. This value is displayed in the API Catalog UI dashboard as the tile title
- **mfaas.catalog-ui-tile.description**

  This parameter specifies the detailed description of the API services product family. This value is displayed in the API Catalog UI dashboard as the tile description
- **mfaas.catalog-ui-tile.version**

  This parameter specifies the semantic version of this API Catalog tile. Increase the version when you introduce new changes to the API services product family details (title and description).
- **mfaas.discovery.info.serviceTitle**

  This parameter specifies the human readable name of the API service instance (for example, "`Endevor Prod`" or "`Sysview LPAR1`"). This value is displayed in the API Catalog when a specific API service instance is selected. This parameter is externalized and set by the customer system administrator.

  Tip: We recommend that you provide a good default value or give good naming examples to the customers.
- **mfaas.discovery.info.description**

  Specifies a short description of the API service.

  Example: "`CA Endevor SCM - Production Instance`" or "`CA SYSVIEW running on LPAR1`". This value is displayed in the API Catalog when a specific API service instance is selected. This parameter is externalized and set by the customer system administrator.

  Tip: We recommend that you provide a good default value or give good naming examples to the customers. Describe the service so that the end user knows the function of the service.
- **mfaas.discovery.info.swaggerLocation**

  This parameter specifies the location of a static swagger document. The JSON document contained in this file is displayed instead of the automatically generated API documentation. The JSON file must contain a valid OpenAPI 2.x Specification document. This value is optional and commented out by default.

  Note: Specifying a `swaggerLocation` value disables the automated JSON API documentation generation with the SpringFox library. By disabling auto-generation, you need to keep the contents of the manual swagger

definition consistent with your endpoints. We recommend to use auto-generation to prevent incorrect endpoint definitions in the static swagger documentation.

d. **Metadata Parameters**

The routing rules can be modified with parameters in the metadata configuration code block.

Note: If your REST API does not conform to Zowe API Mediation Layer REST API Building codes, configure routing to transform your actual endpoints (serviceUrl) to `gatewayUrl` format. For more information see: REST API Building Codes

- **`eureka.instance.metadata-map.routed-services.<prefix>`**

  This parameter specifies a name for the routing rules group. This parameter is only for logical grouping of additional parameters. You can specify an arbitrary value but it is a good development practice to mention the group purpose in the name.

  Examples:

  ```
  api_v1
  api_v2
  ```

- **`eureka.instance.metadata-map.routed-services.<prefix>.gatewayUrl`**

  Both gateway-url and service-url parameters specify how the API service endpoints are mapped to the API gateway endpoints. The gateway-url parameter sets the target endpoint on the gateway.

- **`metadata-map.routed-services.<prefix>.serviceUrl`**

  Both gateway-url and service-url parameters specify how the API service endpoints are mapped to the API gateway endpoints. The service-url parameter points to the target endpoint on the gateway.

- **`eureka.instance.metadata-map.apiml.apiInfo.apiId`**

  This parameter specifies the API identifier that is registered in the API Mediation Layer installation. The API ID uniquely identifies the API in the API Mediation Layer. The same API can be provided by multiple services. The API ID can be used to locate the same APIs that are provided by different services. The creator of the API defines this ID. The API ID needs to be a string of up to 64 characters that uses lowercase alphanumeric characters and a dot: `.`.

  We recommend that you use your organization as the prefix.

- **`eureka.instance.metadata-map.apiml.apiInfo.gatewayUrl`**

  This parameter specifies the base path at the API gateway where the API is available. Ensure that it is the same path as the *gatewayUrl* value in the *routes* sections.

- **`eureka.instance.metadata-map.apiml.apiInfo.documentationUrl`**

  (Optional) This parameter specifies a link to external documentation, if needed. The link to the external documentation can be included along with the Swagger documentation.

- **`eureka.instance.metadata-map.apiml.apiInfo.swaggerUrl`**

  (Optional) This parameter specifies the HTTP or HTTPS address where the Swagger JSON document is available.

  Important! Ensure that each of the values for gatewayUrl parameter are unique in the configuration. Duplicate gatewayUrl values may cause requests to be routed to the wrong service URL.

  Note: The endpoint `/api-doc` returns the API service Swagger JSON. This endpoint is introduced by the `@EnableMfaasInfo` annotation and is utilized by the API Catalog.

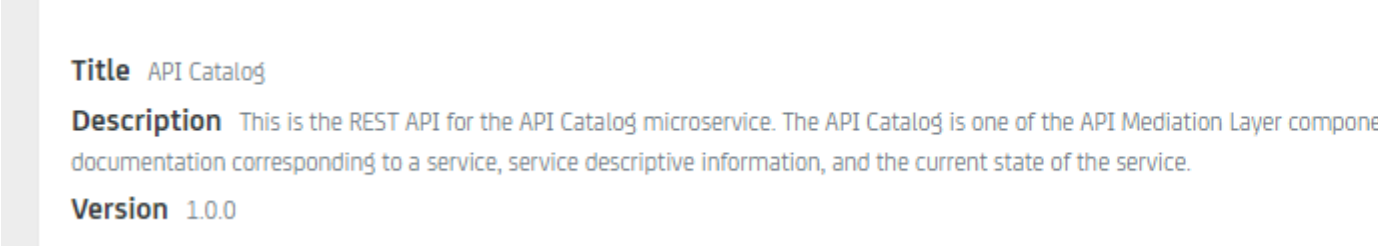
e. **Swagger Api-Doc Parameters**

Parameter in this section configure API Version Header Information, specifically the InfoObject section, and adjusts Swagger documentation that your API service returns. Use the following format:

```
api-info:
 apiVersionProperties:
```

```
   v1:
       title: Your API title for swagger JSON which is displayed in API
Catalog / service / API Information
       description: Your API description for swagger JSON
       version: 1.0.0
       basePackage:
your.service.base.package.for.swagger.annotated.controllers
       # apiPattern: /v1/.*  # alternative to basePackage for exposing
endpoints which match the regex pattern to swagger JSON
```

The following parameters describe the function of the specific version of an API. This information is included in the swagger JSON and displayed in the API Catalog:



where:

- **v1**

  Specifies the major version of your service API: `v1, v2,` etc.
- **title**

  Specifies the title of your service API.
- **description**

  Specifies the high-level function description of your service API.
- **version**

  Specifies the actual version of the API in semantic format.
- **basePackage**

  Specifies the package where the API is located. This option only exposes endpoints that are defined in a specified java package. The parameters `basePackage` and `apiPattern` are mutually exclusive. Specify only one of them and remove or comment out the second one.
- **apiPattern**

  This option exposes any endpoints that match a specified regular expression. The parameters `basePackage` and `apiPattern` are mutually exclusive. Specify just one of them and remove or comment out the second one.

  Tip: You have three options to make your endpoints discoverable and exposed: `basePackage`, `apiPattern`, or none (if you do not specify a parameter). If `basePackage` or `apiPattern` are not defined, all endpoints in the Spring Boot app are exposed.

Setup keystore with the service certificate

To register with the API Mediation Layer, a service is required to have a certificate that is trusted by API Mediation Layer.

Follow these steps:

1. Follow instructions at Generating certificate for a new service on localhost

   When a service is running on `localhost`, the command can have the following format:

```
<api-layer-repository>/scripts/apiml_cm.sh --action new-service --service-
alias localhost --service-ext SAN=dns:localhost.localdomain,dns:localhost
```

```
 --service-keystore keystore/localhost.keystore.p12 --service-truststore
 keystore/localhost.truststore.p12 --service-dname "CN=Sample REST API
 Service, OU=Mainframe, O=Zowe, L=Prague, S=Prague, C=Czechia" --service-
password password --service-validity 365 --local-ca-filename <api-layer-
repository>/keystore/local_ca/localca
```

Alternatively, for the purpose of local development, copy or use the `<api-layer-repository>/keystore/localhost.truststore.p12` in your service without generating a new certificate.

2. Update the configuration of your service `application.yml` to contain the HTTPS configuration by adding the following code:

```
server:
    ssl:
        protocol: TLSv1.2
        ciphers:
 TLS_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDH_RSA_WITH_AES_
        keyAlias: localhost
        keyPassword: password
        keyStore: keystore/localhost.keystore.p12
        keyStoreType: PKCS12
        keyStorePassword: password
        trustStore: keystore/localhost.truststore.p12
        trustStoreType: PKCS12
        trustStorePassword: password
eureka:
    instance:
        nonSecurePortEnabled: false
        securePortEnabled: true
```

Note: You need to define both keystore and truststore even if your server is not using an HTTPS port.

Externalize API ML configuration parameters

The following list summarizes the API ML parameters that are set by the customer system administrator:

- `mfaas.discovery.enabled: ${environment.discoveryEnabled:true}`
- `mfaas.discovery.locations: ${environment.discoveryLocations}`
- `mfaas.discovery.serviceID: ${environment.serviceId}`
- `mfaas.discovery.info.serviceTitle: ${environment.serviceTitle}`
- `mfaas.discovery.info.description: ${environment.serviceDescription}`
- `mfaas.service.hostname: ${environment.hostname}`
- `mfaas.service.ipAddress: ${environment.ipAddress}`
- `mfaas.server.port: ${environment.port}`

Tip: Spring Boot applications are configured in the `application.yml` and `bootstrap.yml` files that are located in the USS file system. However, system administrators prefer to provide configuration through the mainframe sequential data set (or PDS member). To override Java values, use Spring Boot with an external YAML file, environment variables, and Java System properties. For Zowe API Mediation Layer applications, we recommend that you use Java System properties.

Java System properties are defined using `-D` options for Java. Java System properties can override any configuration. Those properties that are likely to change are defined as `${environment.variableName}:`

```
IJO="$IJO -Denvironment.discoveryEnabled=.."
IJO="$IJO -Denvironment.discoveryLocations=.."

IJO="$IJO -Denvironment.serviceId=.."
IJO="$IJO -Denvironment.serviceTitle=.."
IJO="$IJO -Denvironment.serviceDescription=.."
IJO="$IJO -Denvironment.hostname=.."
IJO="$IJO -Denvironment.ipAddress=.."
```

```
IJO="$IJO -Denvironment.port=.."
```

The `discoveryLocations` (public URL of the discovery service) value is found in the API Meditation Layer configuration, in the `*.PARMLIB(MASxPRM)` member and assigned to the MFS_EUREKA variable.

Example:

```
MFS_EUREKA="http://eureka:password@141.202.65.33:10011/eureka/")
```

Test your service

To test that your API instance is working and is discoverable, use the following validation tests:

Validate that your API instance is still working

Follow these steps:

1. Disable discovery by setting `discoveryEnabled=false` in your API service instance configuration.
2. Run your tests to check that they are working as before.

Validate that your API instance is discoverable

Follow these steps:

1. Point your configuration of API instance to use the following Discovery Service:

   ```
   http://eureka:password@localhost:10011/eureka
   ```
2. Start up the API service instance.
3. Check that your API service instance and each of its endpoints are displayed in the API Catalog

   ```
   https://localhost:10010/ui/v1/caapicatalog/
   ```
4. Check that you can access your API service endpoints through the Gateway.

   Example:

   ```
   https://localhost:10010/api/v1/
   ```
5. Check that you can still access your API service endpoints directly outside of the Gateway.

Review the configuration examples of the discoverable client

Refer to the Discoverable Client API Sample Service in the API ML git repository.

Onboard a Java REST API service without Spring Boot

As an API developer, use this guide to onboard a Java REST API service that is built without Spring Boot to the Zowe™ API Mediation Layer (API ML). This article provides a step-by-step process to onboard a Java REST API application to the API ML.

Note: The following guide is to onboard a REST API service using enabler versions 1.2 and earlier. To onboard a REST API service using enabler versions 1.3 and higher, see the Onboarding Overview on page 164 for a complete list of Zowe™ API Mediation Layer onboarding methods.

The following procedure outlines the general steps to onboard a Java REST API service without Spring Boot with the Zowe™ API Mediation Layer. Each step is described in more detail in this article.

1. Get enablers from the Artifactory on page 201
   - Gradle guide on page 201
   - Maven guide on page 202
2. (Optional) Add Swagger API documentation to your project on page 202
3. Add endpoints to your API for API Mediation Layer integration on page 203

4. Add configuration for Discovery client on page 204
5. Add a context listener on page 208
    a. Add a context listener class on page 208
    b. Register a context listener on page 209
6. Run your service on page 210
7. (Optional) Validate discovery of the API service by the Discovery Service on page 210

Notes:

- This onboarding procedure uses the Spring framework for implementation of a REST API service, and describes how to generate Swagger API documentation using a *Springfox* library.
- If you use another framework that is based on a Servlet API, you can use `ServletContextListener` that is described later in this article.
- If you use a framework that does not have a `ServletContextListener` class, see the Add a context listener on page 208 section in this article for details about how to register and unregister your service with the API Mediation Layer.

Prerequisites

- Your REST API service is written in Java.
- Your service has an endpoint that generates Swagger documentation.

Get enablers from the Artifactory

The first step to onboard a Java REST API to the Zowe ecosystem is to get enabler annotations from the Artifactory. Enablers prepare your service for discovery in the API Mediation Layer and for the retrieval of Swagger documentation.

You can use either *Gradle* or *Maven* build automation systems.

Gradle guide

Use the following procedure if you use *Gradle* as your build automation system.

Follow these steps:

1. Create a `gradle.properties` file in the root of your project.
2. In the `gradle.properties` file, set the following URL of the repository. Use the values provided in the following code block for user credentials to access the Artifactory:

    ```
    # Repository URL for getting the enabler-java artifact
    artifactoryMavenRepo=https://zowe.jfrog.io/zowe/libs-release
    ```

    This file specifies the URL of the repository of the Artifactory. The enabler-java artifacts are downloaded from this repository.
3. Add the following *Gradle* code block to the `build.gradle` file:

    ```
    ext.mavenRepository = {
        maven {
            url artifactoryMavenSnapshotRepo
            credentials {
                username mavenUser
                password mavenPassword
            }
        }
    }

    repositories mavenRepositories
    ```

    The `ext` object declares the `mavenRepository` property. This property is used as the project repository.

4. In the same `build.gradle` file, add the following code to the dependencies code block to add the enabler-java artifact as a dependency of your project:

```
compile(group: 'org.zowe.apiml.sdk', name: 'mfaas-integration-enabler-
java', version: '1.1.2')
```

5. In your project directory, run the `gradle build` command to build your project.

Maven guide

Use the following procedure if you use *Maven* as your build automation system.

Follow these steps:

1. Add the following *xml* tags within the newly created `pom.xml` file:

```
<repositories>
    <repository>
        <id>libs-release</id>
        <name>libs-release</name>
        <url>https://zowe.jfrog.io/zowe/libs-release</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
```

   This file specifies the URL of the repository of the Artifactory where you download the enabler-java artifacts.

2. In the same `pom.xml` file, copy the following *xml* tags to add the enabler-java artifact as a dependency of your project:

```
<dependency>
    <groupId>org.zowe.apiml.sdk</groupId>
    <artifactId>mfaas-integration-enabler-java</artifactId>
    <version>1.1.2</version>
</dependency>
```

3. Create a `settings.xml` file and copy the following *xml* code block which defines the credentials for the Artifactory:

```
<?xml version="1.0" encoding="UTF-8"?>

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                  https://maven.apache.org/xsd/settings-1.0.0.xsd">
<servers>
    <server>
       <id>libs-release</id>
    </server>
</servers>
</settings>
```

4. Copy the `settings.xml` file inside the `${user.home}/.m2/` directory.
5. In the directory of your project, run the `mvn package` command to build the project.

(Optional) Add Swagger API documentation to your project

If your application already has Swagger API documentation enabled, skip this step. Use the following procedure if your application does not have Swagger API documentation.

Follow these steps:

1. Add a Springfox Swagger dependency.
   - For *Gradle* add the following dependency in `build.gradle`:

     ```
     compile "io.springfox:springfox-swagger2:2.8.0"
     ```

   - For *Maven* add the following dependency in `pom.xml`:

     ```
     <dependency>
         <groupId>io.springfox</groupId>
         <artifactId>springfox-swagger2</artifactId>
         <version>2.8.0</version>
     </dependency>
     ```

2. Add a Spring configuration class to your project:

```
package org.zowe.apiml.hellospring.configuration;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import
 org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

import java.util.ArrayList;

@Configuration
@EnableSwagger2
@EnableWebMvc
public class SwaggerConfiguration extends WebMvcConfigurerAdapter {
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
            .select()
            .apis(RequestHandlerSelectors.any())
            .paths(PathSelectors.any())
            .build()
            .apiInfo(new ApiInfo(
                "Spring REST API",
                "Example of REST API",
                "1.0.0",
                null,
                null,
                null,
                null,
                new ArrayList<>()
            ));
    }
}
```

3. Customize this configuration according to your specifications. For more information about customization properties, see Springfox documentation.

Add endpoints to your API for API Mediation Layer integration

To integrate your service with the API Mediation Layer, add the following endpoints to your application:

- **Swagger documentation endpoint**

  The endpoint for the Swagger documentation.
- **Health endpoint**

  The endpoint used for health checks by the Discovery Service.
- **Info endpoint**

  The endpoint to get information about the service.

The following Java code is an example of these endpoints added to the Spring Controller:

Example:

```
package org.zowe.apiml.hellospring.controller;

import org.zowe.apiml.eurekaservice.model.*;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import springfox.documentation.annotations.ApiIgnore;

@Controller
@ApiIgnore
public class MfaasController {

    @GetMapping("/api-doc")
    public String apiDoc() {
        return "forward:/v2/api-docs";
    }

    @GetMapping("/application/health")
    public @ResponseBody Health getHealth() {
        return new Health("UP");
    }

    @GetMapping("/application/info")
    public @ResponseBody ResponseEntity<EmptyJsonResponse>
 getDiscoveryInfo() {
        HttpHeaders headers = new HttpHeaders();
        headers.add("Content-Type", "application/json");
        return new ResponseEntity(new EmptyJsonResponse(), headers,
 HttpStatus.OK);
    }
}
```

Add configuration for Discovery client

After you add API Mediation Layer integration endpoints, you are ready to add service configuration for Discovery client.

Follow these steps:

1. Create the file `service-configuration.yml` in your resources directory.
2. Add the following configuration to your `service-configuration.yml`:

```
serviceId: hellospring
title: HelloWorld Spring REST API
description: POC for exposing a Spring REST API
baseUrl: http://localhost:10020/hellospring
homePageRelativeUrl:
statusPageRelativeUrl: /application/info
healthCheckRelativeUrl: /application/health
discoveryServiceUrls:
```

```
    - http://eureka:password@localhost:10011/eureka
routes:
    - gatewayUrl: api/v1
      serviceUrl: /hellospring/api/v1
apiInfo:
    - apiId: ${mfaas.discovery.serviceId}
      gatewayUrl: api/v1
      swaggerUrl: ${mfaas.server.scheme}://${mfaas.service.hostname}:
${mfaas.server.port}${mfaas.server.contextPath}/api-doc
      documentationUrl: https://docs.zowe.org
catalogUiTile:
    id: helloworld-spring
    title: HelloWorld Spring REST API
    description: Proof of Concept application to demonstrate exposing a
 REST API in the MFaaS ecosystem
    version: 1.0.0
```

3. Customize your configuration parameters to correspond with your API service specifications.

   The following list describes the configuration parameters:

   - **serviceId**

     This parameter specifies the service instance identifier that is registered in the API Mediation Layer installation. The service ID is used in the URL for routing to the API service through the gateway. The service

ID uniquely identifies instances of a microservice in the API Mediation Layer. The system administrator at the customer site defines this parameter.

Important! Ensure that the service ID is set properly with the following considerations:

- When two API services use the same service ID, the API Gateway considers the services to be clones. An incoming API request can be routed to either of them.
- The same service ID should be set only for multiple API service instances for API scalability.
- The service ID value must contain only lowercase alphanumeric characters.
- The service ID cannot contain more than 40 characters.
- The service ID is linked to security resources. Changes to the service ID require an update of security resources.

Examples:

- If the customer system administrator sets the service ID to `sysviewlpr1`, the API URL in the API Gateway appears as the following URL:

  ```
  https://gateway:port/api/v1/sysviewlpr1/endpoint1/...
  ```

- If a customer system administrator sets the service ID to vantageprod1, the API URL in the API Gateway appears as the following URL:

  ```
  http://gateway:port/api/v1/vantageprod1/endpoint1/...
  ```

- **title**

  This parameter specifies the human readable name of the API service instance (for example, "Endevor Prod" or "Sysview LPAR1"). This value is displayed in the API Catalog when a specific API service instance is selected. This parameter is externalized and set by the customer system administrator.

  Tip: We recommend that you provide a specific default value of the `title`. Use a title that describes the service instance so that the end user knows the specific purpose of the service instance.

- **description**

  This parameter is a short description of the API service.

  Example: "CA Endevor SCM - Production Instance" or "CA SYSVIEW running on LPAR1".

  This value is displayed in the API Catalog when a specific API service instance is selected. This parameter is externalized and set by the customer system administrator.

  Tip: Describe the service so that the end user knows the function of the service.

- **baseUrl**

  This parameter specifies the URL to your service to the REST resource. It will be the prefix for the following URLs:

  - **homePageRelativeUrl**
  - **statusPageRelativeUrl**
  - **healthCheckRelativeUrl**.

  Examples:

  - `http://host:port/servicename` for HTTP service
  - `https://host:port/servicename` for HTTPS service

- **homePageRelativeUrl**

  This parameter specifies the relative path to the home page of your service. The path should start with `/`. If your service has no home page, leave this parameter blank.

  Examples:

  - `homePageRelativeUrl:` The service has no home page

  - `homePageRelativeUrl: /` The service has home page with URL `${baseUrl}/`
- **statusPageRelativeUrl**

  This parameter specifies the relative path to the status page of your service. This is the endpoint that you defined in the `MfaasController` controller in the `getDiscoveryInfo` method. Start this path with `/`.

  Example:

  ```
  statusPageRelativeUrl: /application/info
  ```

  The result URL will be:

  ```
  ${baseUrl}/application/info
  ```
- **healthCheckRelativeUrl**

  This parameter specifies the relative path to the health check endpoint of your service. This is the endpoint that you defined in the `MfaasController` controller in the `getHealth` method. Start this URL with `/`.

  Example:

  ```
  healthCheckRelativeUrl: /application/health
  ```

  This results in the URL:

  ```
  ${baseUrl}/application/health
  ```
- **discoveryServiceUrls**

  This parameter specifies the public URL of the Discovery Service. The system administrator at the customer site defines this parameter.

  Example:

  ```
  http://eureka:password@141.202.65.33:10311/eureka/
  ```
- **routedServices**

  This parameter specifies the routing rules between the gateway service and your service.

  - **routedServices.gatewayUrl**

    Both gateway-url and service-url parameters specify how the API service endpoints are mapped to the API gateway endpoints. The gateway-url parameter sets the target endpoint on the gateway.
  - **routedServices.serviceUrl**

    Both gateway-url and service-url parameters specify how the API service endpoints are mapped to the API gateway endpoints. The service-url parameter points to the target endpoint on the gateway.
- **apiInfo.apiId**

  This parameter specifies the API identifier that is registered in the API Mediation Layer installation. The API ID uniquely identifies the API in the API Mediation Layer. The same API can be provided by multiple services. The API ID can be used to locate the same APIs that are provided by different services. The

creator of the API defines this ID. The API ID needs to be a string of up to 64 characters that uses lowercase alphanumeric characters and a dot: `.`.

We recommend that you use your organization as the prefix.

- **apiInfo.gatewayUrl**

  This parameter specifies the base path at the API Gateway where the API is available. Ensure that this is the same path as the *gatewayUrl* value in the *routes* sections.
- **apiInfo.swaggerUrl**

  (Optional) This parameter specifies the HTTP or HTTPS address where the Swagger JSON document is available.
- **apiInfo.documentationUrl**

  (Optional) This parameter specifies the link to external documentation, if needed. The link to the external documentation can be included along with the Swagger documentation.
- **catalogUiTile.id**

  This parameter specifies the unique identifier for the API services product family. This is the grouping value used by the API Mediation Layer to group multiple API services together into "tiles". Each unique identifier represents a single API Catalog UI dashboard tile. Specify a value that does not interfere with API services from other products.
- **catalogUiTile.title**

  This parameter specifies the title of the API services product family. This value is displayed in the API catalog UI dashboard as the tile title.
- **catalogUiTile.description**

  This parameter is the detailed description of the API services product family. This value is displayed in the API catalog UI dashboard as the tile description.
- **catalogUiTile.version**

  This parameter specifies the semantic version of this API Catalog tile. Ensure that you increase the number of the version when you introduce new changes to the product family details of the API services including the title and description.

Add a context listener

The context listener invokes the `apiMediationClient.register(config)` method to register the application with the API Mediation Layer when the application starts. The context listener also invokes the `apiMediationClient.unregister()` method before the application shuts down to unregister the application in API Mediation Layer.

Note: If you do not use a Java Servlet API based framework, you can still call the same methods for `apiMediationClient` to register and unregister your application.

Add a context listener class

Add the following code block to add a context listener class:

```
package org.zowe.apiml.hellospring.listener;

import org.zowe.apiml.eurekaservice.client.ApiMediationClient;
import org.zowe.apiml.eurekaservice.client.config.ApiMediationServiceConfig;
import org.zowe.apiml.eurekaservice.client.impl.ApiMediationClientImpl;
import
 org.zowe.apiml.eurekaservice.client.util.ApiMediationServiceConfigReader;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;


public class ApiDiscoveryListener implements ServletContextListener {
```

```
    private ApiMediationClient apiMediationClient;

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        apiMediationClient = new ApiMediationClientImpl();
        String configurationFile = "/service-configuration.yml";
        ApiMediationServiceConfig config = new
 ApiMediationServiceConfigReader(configurationFile).readConfiguration();
        apiMediationClient.register(config);
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        apiMediationClient.unregister();
    }
}
```

Register a context listener

Register a context listener to start the Discovery client. Add the following code block to the deployment descriptor `web.xml` to register a context listener:

```
<listener>
    <listener-
class>org.zowe.apiml.hellospring.listener.ApiDiscoveryListener</listener-
class>
</listener>
```

Setup key store with the service certificate

All API services require a certificate that is trusted by API Mediation Layer in order to register with it.

Follow these steps:

1. Follow instructions at Generating certificate for a new service on localhost

   If the service runs on localhost, the command uses the following format:

   ```
   <api-layer-repository>/scripts/apiml_cm.sh --action new-service --service-
   alias localhost --service-ext SAN=dns:localhost.localdomain,dns:localhost
    --service-keystore keystore/localhost.keystore.p12 --service-truststore
    keystore/localhost.truststore.p12 --service-dname "CN=Sample REST API
    Service, OU=Mainframe, O=Zowe, L=Prague, S=Prague, C=Czechia" --service-
   password password --service-validity 365 --local-ca-filename <api-layer-
   repository>/keystore/local_ca/localca
   ```

   Alternatively, copy or use the `<api-layer-repository>/keystore/localhost.truststore.p12` in your service without generating a new certificate, for local development.
2. Update the configuration of your service `service-configuration.yml` to contain the HTTPS configuration by adding the following code:

   ```
      ssl:
          protocol: TLSv1.2
          ciphers:
    TLS_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDH_RSA_WITH_AES_
          keyAlias: localhost
          keyPassword: password
          keyStore: keystore/localhost.keystore.p12
          keyStoreType: PKCS12
          keyStorePassword: password
          trustStore: keystore/localhost.truststore.p12
          trustStoreType: PKCS12
          trustStorePassword: password
   ```

```
eureka:
    instance:
        nonSecurePortEnabled: false
        securePortEnabled: true
```

Note: You need to define both key store and trust store even if your server is not using HTTPS port.

Run your service

After you add all configurations and controllers, you are ready to run your service in the API Mediation Layer ecosystem.

Follow these steps:

1. Run the following services to onboard your application:
   - Gateway Service
   - Discovery Service
   - API Catalog Service

   Tip: For more information about how to run the API Mediation Layer locally, see Running the API Mediation Layer on Local Machine.
2. Run your Java application.

   Tip: Wait for the Discovery Service to discover your service. This process may take a few minutes.
3. Go to the following URL to reach the API Catalog through the Gateway (`port 10010`):

```
https://localhost:10010/ui/v1/apicatalog/
```

You successfully onboarded your Java application to the API Mediation Layer if your service is running and you can access the API documentation.

(Optional) Validate discovery of the API service by the Discovery Service

If your service is not visible in the API Catalog, you can check if your service is discovered by the Discovery Service.

Follow these steps:

1. Go to `http://localhost:10011`.
2. Enter `eureka` as a username and `password` as a password.
3. Check if your application appears in the Discovery Service UI.

If your service appears in the Discovery Service UI but is not visible in the API Catalog, check to make sure your configuration settings are correct.

Onboard a REST API without code changes required

As a user of Zowe™ API Mediation Layer, onboard an existing REST API service with the Zowe™ API Mediation Layer without changing the code of the API service. This is process is also described as static onboarding.

Note: When developing a new service, it is not recommended to onboard a REST service using this method, as this method is non-native to the API Mediaiton Layer. For a complete list of methods to onboard a REST service natively to the API Mediation Layer, see the Onboarding Overview on page 164.

The following procedure outlines the steps to onboard an API service through the API Gateway in the API Mediation Layer without requiring code changes.

1. Identify the APIs that you want to expose on page 211
2. Route your API
3. Define your service and API in YAML format
4. Configuration parameters
5. Add and validate the definition in the API Mediation Layer running on your machine

6. Add a definition in the API Mediation Layer in the Zowe runtime
7. (Optional) Check the log of the API Mediation Layer
8. (Optional) Reload the services definition after the update when the API Mediation Layer is already started

Tip: For more information about the structure of APIs and which APIs to expose in the Zowe API Mediation Layer, see the Onboarding Overview on page 164.

Identify the APIs that you want to expose

The first step in API service onboarding is to identify the APIs that you want to expose.

Follow these steps:

1. Identify the following parameters of your API service:
   - Hostname
   - Port
   - (Optional) base path where the service is available. This URL is called base URL of the service.

   Example:

   In the sample service described earlier, the URL of the service is: `http://localhost:8080`.
2. Identify all APIs that this service provides that you want to expose through the API Gateway.

   Example:

   In the sample service, this REST API is the one available at the path `/v2` relative to base URL of the service. This API is version 2 of the Pet Store API.
3. Choose the `service ID` of your service. The `service ID` identifies the service in the API Gateway. The `service ID` is an alphanumeric string in lowercase ASCII.

   Example:

   In the sample service, the `service ID` is `petstore`.
4. Decide which URL to use to make this API available in the API Gateway. This URL is refered to as the gateway URL and is composed of the API type and the major version.

   Example:

   In the sample service, we provide a REST API. The first segment is `/api`. To indicate that this is version 2, the second segment is `/v2`.

2. Route your API

After you identify the APIs you want to expose, define the routing of your API. Routing is the process of sending requests from the API gateway to a specific API service. Route your API by using the same format as in the following `petstore` example.

Note: The API Gateway differentiates major versions of an API.

Example:

To access version 2 of the `petstore` API use the following gateway URL:

```
https://gateway-host:port/api/v2/petstore
```

The base URL of the version 2 of the `petstore` API is:

```
http://localhost:8080/v2
```

The API Gateway routes REST API requests from the gateway URL `https://gateway:port/api/v2/petstore` to the service `http://localhost:8080/v2`. This method provides access to the service in the API Gateway through the gateway URL.

Note: This method enables you to access the service through a stable URL and move the service to another machine without changing the gateway URL. Accessing a service through the API Gateway also enables you to have multiple instances of the service running on different machines to achieve high-availability.

3. Define your service and API in YAML format

Define your service and API in YAML format in the same way as presented in the following sample `petstore` service example.

Example:

To define your service in YAML format, provide the following definition in a YAML file as in the following sample `petstore` service:

```
services:
    - serviceId: petstore
      catalogUiTileId: static
      title: Petstore Sample Service
      description: This is a sample server Petstore service
      instanceBaseUrls:
        - http://localhost:8080
      routes:
        - gatewayUrl: api/v2
          serviceRelativeUrl: /v2
      apiInfo:
        - apiId: io.swagger.petstore
          gatewayUrl: api/v2
          swaggerUrl: http://localhost:8080/v2/swagger.json
          documentationUrl: https://petstore.swagger.io/
          version: 2.0.0

catalogUiTiles:
    static:
        title: Static API services
        description: Services which demonstrate how to make an API service
 discoverable in the APIML ecosystem using YAML definitions
```

In this example, a suitable name for the file is `petstore.yml`.

Notes:

- The filename does not need to follow specific naming conventions but it requires the `.yml` extension.
- The file can contain one or more services defined under the `services:` node.
- Each service has a service ID. In this example, the service ID is `petstore`. The service can have one or more instances. In this case, only one instance `http://localhost:8080` is used.
- A service can provide multiple APIs that are routed by the API Gateway. In this case, requests with the relative base path `api/v2` at the API Gateway (full gateway URL: `https://gateway:port/api/v2/...`) are routed to the relative base path `/v2` at the full URL of the service (`http://localhost:8080/v2/...`).

Tips:

- There are more examples of API definitions at this link.
- For more details about how to use YAML format, see this link.

4. Customize configuration parameters

The following list describes the configuration parameters:

- **serviceId**

  This parameter specifies the service instance identifier that is registered in the API Mediation Layer installation. The service ID is used in the URL for routing to the API service through the gateway. The service ID uniquely

identifies the service in the API Mediation Layer. The system administrator at the customer site defines this parameter.

Important! Ensure that the service ID is set properly with the following considerations:

- When two API services use the same service ID, the API gateway considers the services to be clones (two instances for the same service). An incoming API request can be routed to either of them.
- The same service ID should be set only for multiple API service instances for API scalability.
- The service ID value must contain only lowercase alphanumeric characters.
- The service ID cannot contain more than 40 characters.
- The service ID is linked to security resources. Changes to the service ID require an update of security resources.

Examples:

- If the customer system administrator sets the service ID to `sysviewlpr1`, the API URL in the API Gateway appears as the following URL:

  ```
  https://gateway:port/api/v1/sysviewlpr1/...
  ```
- If customer system administrator sets the service ID to `vantageprod1`, the API URL in the API Gateway appears as the following URL:

  ```
  http://gateway:port/api/v1/vantageprod1/...
  ```

- **title**

  This parameter specifies the human readable name of the API service instance (for example, "`Endevor Prod`" or "`Sysview LPAR1`"). This value is displayed in the API catalog when a specific API service instance is selected. This parameter is externalized and set by the customer system administrator.

  Tip: We recommend that you provide a specific default value of the `title`. Use a title that describes the service instance so that the end user knows the specific purpose of the service instance.

- **description**

  This parameter specifies a short description of the API service.

  Example: "CA Endevor SCM - Production Instance" or "CA SYSVIEW running on LPAR1".

  This value is displayed in the API Catalog when a specific API service instance is selected. This parameter is externalized and set by the customer system administrator.

  Tip: Describe the service so that the end user knows the function of the service.

- **instanceBaseUrls**

  This parameter specifies a list of base URLs to your service to the REST resource. It will be the prefix for the following URLs:

  - **homePageRelativeUrl**
  - **statusPageRelativeUrl**
  - **healthCheckRelativeUrl**

  Examples:

  - `- http://host:port/filemasterplus` for an HTTP service
  - `- https://host:port/endevor` for an HTTPS service

  You can provide one URL if your service has one instance. If your service provides multiple instances for the high-availability then you can provide URLs to these instances.

  - `- https://host1:port1/endevor`
  - `- https://host2:port2/endevor`

- **homePageRelativeUrl**

  This parameter specifies the relative path to the homepage of your service. The path should start with `/`. If your service has no homepage, omit this parameter.

  Examples:

  - `homePageRelativeUrl: /` The service has homepage with URL `${baseUrl}/`
  - `homePageRelativeUrl: /ui/` The service has homepage with URL `${baseUrl}/ui/`
  - `homePageRelativeUrl:` The service has homepage with URL `${baseUrl}`
- **statusPageRelativeUrl**

  This parameter specifies the relative path to the status page of your service. Start this path with `/`. If you service has not a status page, omit this parameter.

  Example:

  - `statusPageRelativeUrl: /application/info` the result URL will be `${baseUrl}/application/info`
- **healthCheckRelativeUrl**

  This parameter specifies the relative path to the health check endpoint of your service. Start this URL with `/`. If your service does not have a health check endpoint, omit this parameter.

  Example:

  - `healthCheckRelativeUrl: /application/health`. This results in the URL: `${baseUrl}/application/health`
- **routes**

  The following parameters specify the routing rules between the gateway service and your service.

  - **routes.gatewayUrl**

    Both *gatewayUrl* and *serviceUrl* parameters specify how the API service endpoints are mapped to the API gateway endpoints. The *gatewayUrl* parameter sets the target endpoint on the gateway.
  - **routes.serviceUrl**

    Both *gatewayUrl* and *serviceUrl* parameters specify how the API service endpoints are mapped to the API gateway endpoints. The *serviceUrl* parameter points to the target endpoint on the gateway.
- **apiInfo**

  This section defines APIs that are provided by the service. Currently, only one API is supported.
- **apiInfo.apiId**

  This parameter specifies the API identifier that is registered in the API Mediation Layer installation. The API ID uniquely identifies the API in the API Mediation Layer. The same API can be provided by multiple services. The API ID can be used to locate the same APIs that are provided by different services.

  The creator of the API defines this ID. The API ID needs to be a string of up to 64 characters that uses lowercase alphanumeric characters and a dot: `.`.

  We recommend that you use your organization as the prefix.

  Examples:

  - `org.zowe.file`
  - `com.ca.sysview`
  - `com.ibm.zosmf`
- **apiInfo.gatewayUrl**

  This parameter specifies the base path at the API gateway where the API is available. Ensure that this path is the same as the *gatewayUrl* value in the *routes* sections.

- **apiInfo.swaggerUrl**

  (Optional) This parameter specifies the HTTP or HTTPS address where the Swagger JSON document is available.
- **apiInfo.documentationUrl**

  (Optional) This parameter specifies a URL to a website where external documentation is provided. This can be used when *swaggerUrl* is not provided.
- **apiInfo.version**

  (Optional) This parameter specifies the actual version of the API in semantic versioning format. This can be used when *swaggerUrl* is not provided.
- **catalogUiTileId**

  This parameter specifies the unique identifier for the API services group. This is the grouping value used by the API Mediation Layer to group multiple API services together into "tiles". Each unique identifier represents a single API Catalog UI dashboard tile. Specify the value based on the ID of the defined tile.
- **catalogUiTile**

  This section contains definitions of tiles. Each tile is defined in a section that has its tile ID as a key. A tile can be used by multiple services.

  ```
  catalogUiTiles:
      tile1:
          title: Tile 1
          description: This is the first tile with ID tile1
      tile2:
          title: Tile 2
          description: This is the second tile with ID tile2
  ```
- **catalogUiTile.{tileId}.title**

  Specifies the title of the API services product family. This value is displayed in the API catalog UI dashboard as the tile title.
- **catalogUiTile.{tileId}.description**

  Specifies the detailed description of the API Catalog UI dashboard tile. This value is displayed in the API catalog UI dashboard as the tile description.

5. Add and validate the definition in the API Mediation Layer running on your machine

After you define the service in YAML format, you are ready to add your service definition to the API Mediation Layer ecosystem.

The following procedure describes how to add your service to the API Mediation Layer on your local machine.

Follow these steps:

1. Copy or move your YAML file to the `config/local/api-defs` directory in the directory with API Mediation Layer.
2. Start the API Mediation Layer services.

   Tip: For more information about how to run the API Mediation Layer locally, see Running the API Mediation Layer on Local Machine.
3. Run your Java application.

   Tip: Wait for the services to be ready. This process may take a few minutes.

4. Go to the following URL to reach the API Gateway (`port 10010`) and see the paths that are routed by the API Gateway:

   ```
   https://localhost:10010/application/routes
   ```

   The following line should appear:

   ```
   /api/v2/petstore/**: "petstore"
   ```

   This line indicates that requests to relative gateway paths that start with `/api/v2/petstore/` are routed to the service with the service ID `petstore`.

   You successfully defined your Java application if your service is running and you can access the service endpoints. The following example is the service endpoint for the sample application:

   ```
   https://localhost:10010/api/v2/petstore/pets/1
   ```

6. Add a definition in the API Mediation Layer in the Zowe runtime

After you define and validate the service in YAML format, you are ready to add your service definition to the API Mediation Layer running as part of the Zowe runtime installation.

Follow these steps:

1. Locate the Zowe instance directory. The Zowe instance directory is chosen during Zowe configuration. The initial location of the directory is in the `zowe-install.yaml` file in the variable `install:instanceDir`.

   Note: We use the `${zoweInstanceDir}` symbol in following instructions.
2. Copy your YAML file to the `${zoweInstanceDir}/workspace/api-mediation/api-defs` directory.

   Note: The `${zoweInstanceDir}/workspace/api-mediation/api-defs` directory is created the first time that Zowe starts, so if you have not started Zowe yet this directory might be missing.
3. Run your application.
4. Restart Zowe runtime or follow steps in section (Optional) Reload the services definition after the update when the API Mediation Layer is already started.
5. Go to the following URL to reach the API Gateway (default port 7554) and see the paths that are routed by the API Gateway:

   ```
   https://${zoweHostname}:${gatewayHttpsPort}/application/routes
   ```

   The following line should appear:

   ```
   /api/v2/petstore/**: "petstore"
   ```

   This line indicates that requests to the relative gateway paths that start with `/api/v2/petstore/` are routed to the service with service ID `petstore`.

   You successfully defined your Java application if your service is running and you can access its endpoints. The endpoint displayed for the sample application is: `https://l${zoweHostname}:${gatewayHttpsPort}/api/v2/petstore/pets/1`

7. (Optional) Check the log of the API Mediation Layer

The API Mediation Layer prints the following messages to its log when the API definitions are processed:

```
Scanning directory with static services definition: config/local/api-defs
Static API definition file: /Users/plape03/workspace/api-layer/config/local/
api-defs/petstore.yml
Adding static instance STATIC-localhost:petstore:8080 for service ID
 petstore mapped to URL http://localhost:8080
```

8. (Optional) Reload the services definition after the update when the API Mediation Layer is already started

The following procedure enables you to refresh the API definitions after you change the definitions when the API Mediation Layer is already running.

Follow these steps:

1. Use a REST API client to issue a POST request to the Discovery Service (port 10011):

   `http://localhost:10011/discovery/api/v1/staticApi`

   The Discovery Service requires authentication by a client certificate. If the API Mediation Layer is running on your local machine, the certificate is stored at `keystore/localhost/localhost.pem`.

   This example uses the HTTPie command-line HTTP client:

   ```
   http --cert=keystore/localhost/localhost.pem --verify=keystore/local_ca/
   localca.cer -j POST     https://localhost:10011/discovery/api/v1/staticApi
   ```

2. Check if your updated definition is effective.

   Note: It can take up to 30 seconds for the API Gateway to pick up the new routing.

API Mediation Layer Message Service Component

The API ML Message Service component unifies and stores REST API error messages and log messages in a single file. The Message Service component enables users to mitigate the problem of message definition redundancy which helps to optimize the development process.

- Message Definition on page 217
- Creating a message on page 218
- Mapping a message on page 218
- API ML Logger on page 219

Message Definition

API ML uses a customizable infrastructure to format both REST API error messages and log messages. `yaml` files make it possible to centralize both API error messages and log messages. Messages have the following definitions:

- Message `key` - a unique ID in the form of a dot-delimited string that describes the reason for the message. The `key` enables the UI or the console to show a meaningful and localized message.

  Tips:

  - We recommend using the format `org.zowe.sample.apiservice.{TYPE}.greeting.empty` to define the message key. `{TYPE}` can be the api or log keyword.
  - Use the message `key` and not the message `number`. The message `number` makes the code less readable, and increases the possibility of errors when renumbering values inside the `number`.
- Message `number` - a typical mainframe message ID (excluding the severity code)
- Message `type` - There are two Massage types:
  - REST API error messages: `ERROR`
  - Log messages: `ERROR`, `WARNING`, `INFO`, `DEBUG`, or `TRACE`
- Message `text` - a description of the issue

The following example shows the message definition.

Example:

```
messages:
    - key: org.zowe.sample.apiservice.{TYPE}.greeting.empty
      number: ZWEASA001
      type: ERROR
```

```
      text: "The provided '%s' name is empty."
```

Creating a message

Use the following classes when you create a message:

- `org.zowe.apiml.message.core.MessageService` - lets you create a message from a file.
- `org.zowe.apiml.message.yaml.YamlMessageService` - implements `org.zowe.apiml.message.core.MessageService` so that `org.zowe.apiml.message.yaml.YamlMessageService` can read message information from a `yaml` file, and create a message with message parameters.

Use the following process to create a message.

Follow these steps:

1. Load messages from the `yaml` file.

   Example:

   ```
   MessageService messageService = new YamlMessageService();
   messageService.loadMessages("/api-messages.yml");
   messageService.loadMessages("/log-messages.yml");
   ```

2. Use the `Message createMessage(String key, Object... parameters);` method to create a message.

   Example:

   ```
   Message message =
    messageService.createMessage("org.zowe.sample.apiservice.
   {TYPE}.greeting.empty", "test");
   ```

Mapping a message

You can map the `Message` either to a REST API response or to a log message.

When you map a REST API response, use the following methods:

- `mapToView` - returns a UI model as a list of API Message, and can be used for Rest API error messages
- `mapToApiMessage` - returns a UI model as a single API Message

The following example is a result of using the `mapToView` method.

Example:

```
{
"messages": [
  {
    "messageKey": "org.zowe.sample.apiservice.{TYPE}.greeting.empty",
    "messageType": "ERROR",
    "messageNumber": "ZWEASA001",
    "messageContent": "The provided 'test' name is empty."
  }
  ]
}
```

The following example is the result of using the `mapToApiMessage` method.

Example:

```
{
  "messageKey": "org.zowe.sample.apiservice.{TYPE}.greeting.empty",
  "messageType": "ERROR",
```

```
  "messageNumber": "ZWEASA001",
  "messageContent": "The provided 'test' name is empty."
}
```

API ML Logger

The `org.zowe.apiml.message.log.ApimLogger` component controls messages through the Message Service component.

The following example uses the `log` message definition in a `yaml` file.

Example:

```
messages:
    - key: org.zowe.sample.apiservice.log.greeting.empty
      number: ZWEASA001
      type: DEBUG
      text: "The provided '%s' name is empty."
```

When you map a log message, use `mapToLogMessage` to return a log message as text. The following example is the output of the `mapToLogMessage`.

Example:

```
ZWEASA001D The provided ‘test’ name is empty. {43abb594-3415-4ed5-
a0b5-23e306a91124}
```

Use the `ApimlLogger` to log messages which are defined in the yaml file.

Example:

```
package org.zowe.apiml.client.configuration;

import org.zowe.apiml.message.core.MessageService;
import org.zowe.apiml.message.core.MessageType;
import org.zowe.apiml.message.log.ApimlLogger;

public class SampleClass {

   private final ApimlLogger logger;

   public SampleClass(MessageService messageService) {
       logger = ApimlLogger.of(SampleClass.class, messageService);
   }

   public void process() {
       logger.log(“org.zowe.sample.apiservice.log.greeting.empty”, “test”);

   }

}
```

The following example shows the output of a successful `ApimlLogger` usage.

Example:

```
DEBUG (c.c.m.c.c.SampleClass) ZWEASA001D The provided 'test' name  is empty.
 {43abb594-3415-4ed5-a0b5-23e306a91124}
```

API Mediation Layer onboarding configuration

This article describes the process of configuring a REST service to onboard with the Zowe API Mediation Layer using the API ML Plain Java Enabler. As a service developer, you can provide basic configuration of a service to

onboard with the API ML. You can also externalize configuration parameters for subsequent customization by a systems administrator.

- Introduction on page 220
- Configuring a REST service for API ML onboarding on page 220
- Plain Java Enabler service onboarding API on page 221
  - Automatic initialization of the onboarding configuration by a single method call on page 221
- Loading YAML configuration files on page 222
  - Loading a single YAML configuration file on page 222
  - Loading and merging two YAML configuration files on page 222

Introduction

The API ML Plain Java Enabler (*PJE*) is a library which helps to simplify the process of onboarding a REST service with the API ML. This article describes how to provide and externalize the Zowe API ML onboarding configuration of your REST service using the *PJE*.

Note: For more information about specific configuration parameters and their possible values, and the service registration process, see the specific documentation of the onboarding approach you are using for your project:

- Direct REST call registration (No enabler)
- Plain Java Enabler

The *PJE* is the most universal Zowe API ML enabler. This enabler uses only Java, and does not use advanced Inversion of Control (*IoC*) or Dependency Injection (*DI*) technologies. The *PJE* enables you to onboard any REST service implemented in Java, avoiding dependencies, versions collisions, unexpected application behavior, and unnecessarily large service executables.

Service developers provide onboarding configuration as part of the service source code. While this configuration is valid for the development system environment, it is likely to be different for an automated integration environment. Typically, system administrators need to deploy a service on multiple sites that have different system environments and requirements such as security.

PJE supports both the service developer and the system administrator with the functionality of externalizing the service onboarding configuration.

The *PJE* provides a mechanism to load API ML onboarding service configuration from one or two *YAML* files.

Configuring a REST service for API ML onboarding

In most cases, the API ML Discovery Service, Gateway, and service endpoint addresses are not known at the time of building the service executables. Similarly, security material such as certificates, private/public keys, and their corresponding passwords depend on the specific deployment environment, and are not intended to be publicly accessible. Therefore, to provide a higher level of flexibility, the *PJE* implements routines to build service onboarding configuration by locating and loading one or two *YAML* file sources:

- **internal *service-configuration.yml***

  The first configuration file is typically internal to the service deployment artifact. This file must be accessible on the service `classpath`. This file contains basic API ML configuration based on values known at development time. Usually, this basic API ML configuration is provided by the service developer and is located in the `/resources` folder of the Java project source tree. This file is usually found in the deployment artifacts under `/WEB-INF/classes`. The configuration contained in this file is provided by the service developer or builder. As such, it will not match every possible production environment and its corresponding requirements.
- **external or additional *service-configuration.yml***

  The second configuration file is used to externalize the configuration. This file can be stored anywhere on the local file system, as long as that the service has access to that location. This file is provided by the service deployer/system administrator and contains the correct parameter values for the specific production environment.

At service start-up time, both *YAML* configuration files are merged, where the externalized configuration (if provided) has higher priority.

The values of parameters in both files can be rewritten by Java system properties or servlet context parameters that were defined during service installation/configuration, or at start-up time.

In the *YAML* file, standard rewriting placeholders for parameter values use the following format:

`${apiml.parameter.key}`

The actual values are taken from pairs defined as Java system properties or servlet context parameters. The system properties can be provided directly on a command line. The servlet context parameters can be provided in the service `web.xml` or in an external file.

The specific approach of how to provide the servlet context to the user service application depends on the application loading mechanism and the specific Java servlet container environment.

Example:

If the service is deployed in a Tomcat servlet container, you can configure the context by placing an *XML* file with the same name as the application deployment unit into `_$CATALINA_BASE/conf/[enginename]/[hostname]/_`.

Other containers provide different mechanisms for the same purpose.

Plain Java Enabler service onboarding API

You can initialize your service onboarding configuration using different methods of the Plain Java Enabler class `ApiMediationServiceConfigReader`:

Automatic initialization of the onboarding configuration by a single method call

The following code block shows automatic initialization of the onboarding configuration by a single method call:

```
public ApiMediationServiceConfig initializeAPIMLConfiguration(ServletContext
 context);
```

This method receives the `ServletContext` parameter, which holds a map of parameters that provide all necessary information for building the onboarding configuration. The following code block is an example of Java Servlet context configuration.

Example:

```
  <Context>

         <Parameter name="apiml.config.location" value="/service-
config.yml"/>
         <!-- Relative path to configuration file:
         <Parameter name="apiml.config.additional-location" value="../conf/
Catalina/localhost/apiml-plugin-poc_plain-java-enabler.yml" />
       -->
         <Parameter name="apiml.config.additional-location" value="/home/
pin/bin/apache-tomcat-9.0.14/conf/Catalina/localhost/apiml-plugin-poc_plain-
java-enabler.yml" />

         <Parameter name="apiml.serviceId" value="discopin" />
         <Parameter name="apiml.serviceIpAddress" value="127.0.0.2" />
         <Parameter name="apiml.discoveryService.hostname"
 value="localhost" />
         <Parameter name="apiml.discoveryService.port" value="10011" />

         <Parameter name="apiml.ssl.enabled" value="true" />
         <Parameter name="apiml.ssl.verifySslCertificatesOfServices"
 value="true" />
         <Parameter name="apiml.ssl.keyPassword" value="password" />
```

```
         <Parameter name="apiml.ssl.keystore.password" value="password" />
         <Parameter name="apiml.ssl.truststore.password" value="password" />
         <Parameter name="apiml.ssl.keystore" value="../keystore/localhost/
localhost.truststore.p12" />
         <Parameter name="apiml.ssl.truststore" value="../keystore/
localhost/localhost.truststore.p12" />

      </Context>
```

Where the two parameters corresponding to the location of the configuration files are:

- `apiml.config.location`

  This parameter describes the location of the basic configuration file.
- `apiml.config.additional-location`

  This parameter describes the location of the external configuration file.

The method in this example uses the provided configuration file names in order to load them as *YAML* files into the internal Java configuration object of type *ApiMediationServiceConfig*.

The other context parameters with the *apiml* prefix are used to rewrite values of properties in the configuration files.

Loading YAML configuration files

YAML configuration files can be loaded either as a single *YAML* file, or by merging two *YAML* files. Use the `loadConfiguration` method described later in this article that corresponds to your service requirements.

After successfully loading a configuration file, the loading method `loadConfiguration` uses Java system properties to substitute corresponding configuration properties.

Loading a single YAML configuration file

To build your configuration from multiple sources, load a single configuration file, and then rewrite parameters as needed using values from another configuration source. See: Loading and merging two YAML configuration files described later in this article.

Use the following method to load a single *YAML* configuration file:

```
public ApiMediationServiceConfig loadConfiguration(String
 configurationFileName);
```

This method receives a single *String* parameter and can be used to load an internal or an external configuration file.

Note: This method first attempts to load the configuration as a Java resource. If the file is not found, the method attempts to resolve the file name as an absolute. If the file name still cannot be found, this method attempts to resolve the file as a relative path. When the file is found, the method loads the contents of the file and maps them to internal data classes. After loading the configuration file, the method attempts to substitute/rewrite configuration property values with corresponding Java System properties.

Loading and merging two YAML configuration files

To load and merge two configuration files, use the following method:

```
public ApiMediationServiceConfig loadConfiguration(String
 internalConfigurationFileName, String externalizedConfigurationFileName)
```

where:

- **String internalConfigurationFileName**

  references the basic configuration file name.
- **String externalizedConfigurationFileName**

  references the external configuration file name.

Note: The external configuration file takes precedence over the basic configuration file in order to match the target deployment environment. After loading and before merging, each configuration will be separately patched using Java System properties.

The following code block presents an example of how to load and merge onboarding configuration from *YAML* files.

Example:

```
@Slf4j
    public class ApiDiscoveryListener implements ServletContextListener {

        /**
         * @{link ApiMediationClient} instance used to register and
unregister the service with API ML Discovery service.
         */
        private ApiMediationClient apiMediationClient;

        /**
         *  Creates {@link ApiMediationServiceConfig}
         *  Creates and initializes {@link ApiMediationClient} instance,
which is then used to register this service
         *  with API ML discovery service. The registration method of
ApiMediationClientImpl catches all RuntimeExceptions
         *  and only can throw {@link ServiceDefinitionException} checked
exception.
         *
         * @param sce
         */
        @Override
        public void contextInitialized(ServletContextEvent sce) {

            ServletContext context = sce.getServletContext();

            /*
             * Call loadConfiguration method with both config file names
initialized above.
             */
            ApiMediationServiceConfig defaultConfig = new
ApiMediationServiceConfigReader().initializeAPIMLConfiguration(context);

            /*
             * Instantiate {@link ApiMediationClientImpl} which is used to
un/register the service with API ML Discovery service.
             */
            apiMediationClient = new ApiMediationClientImpl();

            /*
             * Call the {@link ApiMediationClient} instance to register
your REST service with API ML Discovery service.
             */
            try {
                apiMediationClient.register(defaultConfig);
            } catch (ServiceDefinitionException sde) {
                log.error("Service configuration failed. Check log for
previous errors: ", sde);
            }
        }

        /**
         * If apiMediationClient is not null, attmpts to unregister this
service from API ML registry.
         */
        @Override
        public void contextDestroyed(ServletContextEvent sce) {
```

```
            if (apiMediationClient != null) {
                apiMediationClient.unregister();
            }
        }
    }
```

Developing for Zowe CLI

Developing for Zowe CLI

You can extend Zowe™ CLI by developing plug-ins and contributing code to the base Zowe CLI or existing plug-ins.

How can I contribute?

You can contribute to Zowe CLI in the following ways:

1. Add new commands, options, or other improvements to the base CLI.
2. Develop a plug-in that users can install to Zowe CLI.

You might want to contribute to Zowe CLI to accomplish the following:

- Provide new scriptable functionality for yourself, your organization, or to a broader community.
- Make use of Zowe CLI infrastructure (profiles and programmatic APIs).
- Participate in the Zowe CLI community space.

Getting started

If you want to start working with the code immediately, check out the Zowe CLI core repository and the contribution guidelines. The zowe-cli-sample-plugin GitHub repository is a sample plug-in that adheres to the guidelines for contributing to Zowe CLI projects.

Tutorials

Follow these tutorials to get started working with the sample plug-in:

1. **Setting up your development environment on page 225** - Clone the project and prepare your local environment.
2. **Installing the sample plug-in on page 226** - Install the sample plug-in to Zowe CLI and run as-is.
3. **Extending a plug-in on page 228** - Extend the sample plug-in with a new by creating a programmatic API, definition, and handler.
4. **Developing a new plug-in on page 231** - Create a new CLI plug-in that uses Zowe CLI programmatic APIs and a diff package to compare two data sets.
5. **Implementing profiles in a plug-in on page 236** - Implement user profiles with the plug-in.

Plug-in Development Overview

At a high level, a plug-in must have `imperative-framework` configuration (sample here). This configuration is discovered by `imperative-framework` through the package.json `imperative` key.

A Zowe CLI plug-in will minimally contain the following:

1. **Programmatic API** - Node.js programmatic APIs to be called by your handler or other Node.js applications.
2. **Command definition** - The syntax definition for your command.
3. **Handler implementation** - To invoke your programmatic API to display information in the format that you defined in the definition.

The following guidelines and documentation will assist you during development:

Imperative CLI Framework Documentation

Imperative CLI Framework documentation is a key source of information to learn about the features of Imperative CLI Framework (the code framework that you use to build plug-ins for Zowe CLI). Refer to these supplementary documents during development to learn about specific features such as:

- Auto-generated help
- JSON responses
- User profiles
- Logging, progress bars, experimental commands, and more!

Contribution Guidelines

The Zowe CLI contribution guidelines contain standards and conventions for developing Zowe CLI plug-ins.

The guidelines contain critical information about working with the code, running/writing/maintaining automated tests, developing consistent syntax in your plug-in, and ensuring that your plug-in integrates with Zowe CLI properly:

For more information about ...	See:
General guidelines that apply to contributing to Zowe CLI and Plug-ins	Contribution Guidelines
Conventions and best practices for creating packages and plug-ins for Zowe CLI	Package and Plug-in Guidelines
Guidelines for running tests on Zowe CLI	Testing Guidelines
Guidelines for running tests on the plug-ins that you build	Plug-in Testing Guidelines
Versioning conventions for Zowe CLI and Plug-ins	Versioning Guidelines

Setting up your development environment

Before you follow the development tutorials for creating a Zowe™ CLI plug-in, follow these steps to set up your environment.

Prequisites

Methods to install Zowe CLI on page 104.

Initial setup

To create your development space, you will clone and build zowe-cli-sample-plugin from source.

Before you clone the repository, create a local development folder named `zowe-tutorial`. You will clone and build all projects in this folder.

Branches

There are two branches in the repository that correspond to different Zowe CLI versions. You can develop two branches of your plug-in so that users can install your plug-in into either `@latest` or `@lts-incremental` CLI. Developing for both versions will let you take advantage of new core features quickly and expose your plug-in to a wider range of users.

The `lts-incremental` branch of Sample Plug-in is compatible with the `@lts-incremental` version of core CLI (Zowe Stable release). The `master` branch of Sample Plug-in is compatible with `@latest` version of core CLI (Zowe Active Development release).

For more information about the versioning scheme, see Maintaner Versioning in the Zowe CLI repository.

Clone zowe-cli-sample-plugin and build from source

Clone the repository into your development folder to match the following structure:

```
zowe-tutorial
### zowe-cli-sample-plugin
```

Follow these steps:

1. `cd` to your `zowe-tutorial` folder.
2. `git clone https://github.com/zowe/zowe-cli-sample-plugin`
3. `cd` to your `zowe-cli-sample-plugin` folder.
4. `git checkout lts-incremental`
5. `npm install`
6. `npm run build`

(Optional) Run the automated tests

We recommend running automated tests on all code changes. Follow these steps:

1. `cd` to the `__tests__/__resources__/properties` folder.
2. Copy `example_properties.yaml` to `custom_properties.yaml`.
3. Edit the properties within `custom_properties.yaml` to contain valid system information for your site.
4. `cd` to your `zowe-cli-sample-plugin` folder
5. `npm run test`

Next steps

After you complete your setup, follow the Installing the sample plug-in on page 226 tutorial to install this sample plug-in to Zowe CLI.

Installing the sample plug-in

Before you begin, Setting up your development environment on page 225 your local environment to install a plug-in.

Overview

This tutorial covers installing and running this bundled Zowe™ CLI plugin as-is (without modification), which will display your current directory contents.

The plug-in adds a command to the CLI that lists the contents of a directory on your computer.

Installing the sample plug-in to Zowe CLI

To begin, `cd` into your `zowe-tutorial` folder.

Issue the following commands to install the sample plug-in to Zowe CLI:

```
zowe plugins install ./zowe-cli-sample-plugin
```

Viewing the installed plug-in

Issue `zowe --help` in the command line to return information for the installed `zowe-cli-sample` command group:

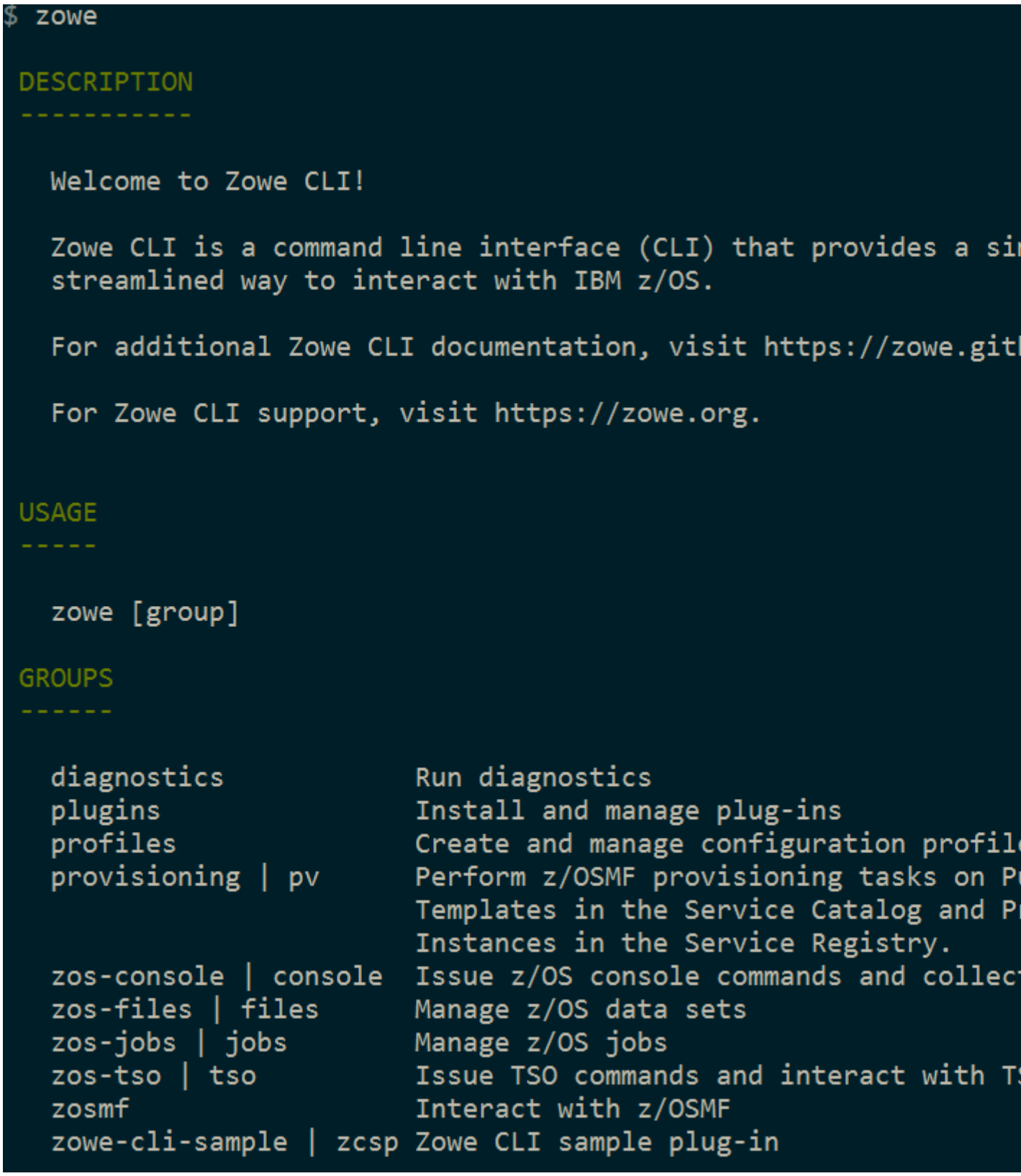


Figure 2: Installed Sample Plugin

Using the installed plug-in

To use the plug-in functionality, issue: `zowe zowe-cli-sample list directory-contents`:

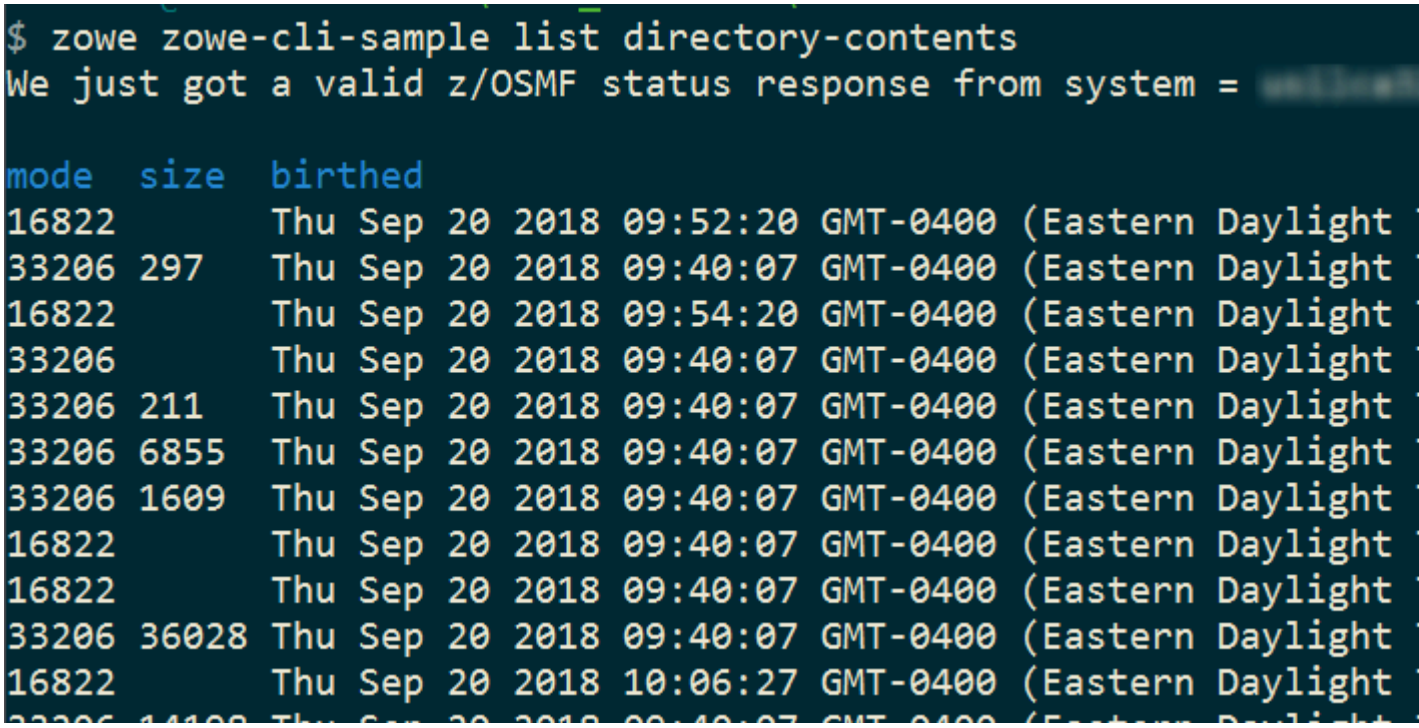


Figure 3: Sample Plugin Output

Testing the installed plug-in

To run automated tests against the plug-in, `cd` into your `zowe-tutorial/zowe-cli-sample-plugin` folder.

Issue the following command:

- `npm run test`

Next steps

You successfully installed a plug-in to Zowe CLI! Next, try the Extending a plug-in on page 228 tutorial to learn about developing new commands for this plug-in.

Extending a plug-in

Before you begin, be sure to complete the Installing the sample plug-in on page 226 tutorial.

Overview

This tutorial demonstrates how to extend the plug-in that is bundled with this sample by:

1. Creating a new programmatic API
2. Creating a new command definition
3. Creating a new handler

We'll do this by using `@brightside/imperative` infrastructure to surface REST API data on our Zowe™ CLI plug-in.

Specifically, we're going to show data from this URI by Typicode. Typicode serves sample REST JSON data for testing purposes.

At the end of this tutorial, you will be able to use a new command from the Zowe CLI interface: `zowe zowe-cli-sample list typicode-todos`

Completed source for this tutorial can be found on the `typicode-todos` branch of the zowe-cli-sample-plugin repository.

Creating a Typescript interface for the Typicode response data

First, we'll create a Typescript interface to map the response data from a server.

Within `zowe-cli-sample-plugin/src/api`, create a folder named `doc` to contain our interface (sometimes referred to as a "document" or "doc"). Within the doc folder, create a file named `ITodo.ts`.

The `ITodo.ts` file will contain the following:

```
export interface ITodo {
    userId: number;
    id: number;
    title: string;
    completed: boolean;
}
```

Creating a programmatic API

Next, we'll create a Node.js API that our command handler uses. This API can also be used in any Node.js application, because these Node.js APIs make use of REST APIs, Node.js APIs, other NPM packages, or custom logic to provide higher level functions than are served by any single API.

Adjacent to the existing file named `zowe-cli-sample-plugin/src/api/Files.ts`, create a file `Typicode.ts`.

`Typicode.ts`should contain the following:

```
import { ITodo } from "./doc/ITodo";
import { RestClient, AbstractSession, ImperativeExpect, Logger } from
 "@brightside/imperative";

export class Typicode {

    public static readonly TODO_URI = "/todos";

    public static getTodos(session: AbstractSession): Promise<ITodo[]> {
        Logger.getAppLogger().trace("Typicode.getTodos() called");
        return RestClient.getExpectJSON<ITodo[]>(session,
 Typicode.TODO_URI);
    }

    public static getTodo(session: AbstractSession, id: number):
 Promise<ITodo> {
        Logger.getAppLogger().trace("Typicode.getTodos() called with id " +
 id);
        ImperativeExpect.toNotBeNullOrUndefined(id, "id must be provided");
        const resource = Typicode.TODO_URI + "/" + id;
        return RestClient.getExpectJSON<ITodo>(session, resource);
    }
}
```

The `Typicode` class provides two programmatic APIs, `getTodos` and `getTodo`, to get an array of `ITodo` objects or a specific `ITodo` respectively. The Node.js APIs use `@brightside/imperative` infrastructure to provide logging, parameter validation, and to call a REST API. See the Imperative CLI Framework documentation for more information.

Exporting interface and programmatic API for other Node.js applications

Update zowe-cli-sample-plugin/src/index.ts to contain the following:

```
export * from "./api/doc/ITodo";
export * from "./api/Typicode";
```

A sample invocation of your API might look similar to the following, if it were used by a separate, standalone Node.js application:

```
import { Typicode } from "@brightside/zowe-cli-sample-plugin";
import { Session, Imperative } from "@brightside/imperative";
import { inspect } from "util";

const session = new Session({ hostname: "jsonplaceholder.typicode.com"});
(async () => {
    const firstTodo = await Typicode.getTodo(session, 1);
    Imperative.console.debug("First todo was: " + inspect(firstTodo));
})();
```

Checkpoint

Issue `npm run build` to verify a clean compilation and confirm that no lint errors are present. At this point in this tutorial, you have a programmatic API that will be used by your handler or another Node.js application. Next you'll define the command syntax for the command that will use your programmatic Node.js APIs.

Defining command syntax

Within Zowe CLI, the full command that we want to create is `zowe zowe-cli-sample list typicode-todos`. Navigate to `zowe-cli-sample-plugin/src/cli/list` and create a folder `typicode-todos`. Within this folder, create `TypicodeTodos.definition.ts`. Its content should be as follows:

```
import { ICommandDefinition } from "@brightside/imperative";
export const TypicodeTodosDefinition: ICommandDefinition = {
    name: "typicode-todos",
    aliases: ["td"],
    summary: "Lists typicode todos",
    description: "List typicode REST sample data",
    type: "command",
    handler: __dirname + "/TypicodeTodos.handler",
    options: [
        {
            name: "id",
            description: "The todo to list",
            type: "number"
        }
    ]
};
```

This describes the syntax of your command.

Defining command handler

Also within the `typicode-todos` folder, create `TypicodeTodos.handler.ts`. Add the following code to the new file:

```
import { ICommandHandler, IHandlerParameters, TextUtils, Session } from
 "@brightside/imperative";
import { Typicode } from "../../../api/Typicode";
export default class TypicodeTodosHandler implements ICommandHandler {

    public static readonly TYPICODE_HOST = "jsonplaceholder.typicode.com";
    public async process(params: IHandlerParameters): Promise<void> {

        const session = new Session({ hostname:
 TypicodeTodosHandler.TYPICODE_HOST});
        if (params.arguments.id) {
            const todo = await Typicode.getTodo(session,
 params.arguments.id);
            params.response.data.setObj(todo);
```

```
            params.response.console.log(TextUtils.prettyJson(todo));
        } else {
            const todos = await Typicode.getTodos(session);
            params.response.data.setObj(todos);
            params.response.console.log(TextUtils.prettyJson(todos));
        }
    }
}
```

The `if` statement checks if a user provides an `--id` flag. If yes, we call `getTodo`. Otherwise, we call `getTodos`. If the Typicode API throws an error, the `@brightside/imperative` infrastructure will automatically surface this.

Defining command to list group

Within the file `zowe-cli-sample-plugin/src/cli/list/List.definition.ts`, add the following code below other `import` statements near the top of the file:

```
import { TypicodeTodosDefinition } from "./typicode-todos/
TypicodeTodos.definition";
```

Then add `TypicodeTodosDefinition` to the children array. For example:

```
children: [DirectoryContentsDefinition, TypicodeTodosDefinition]
```

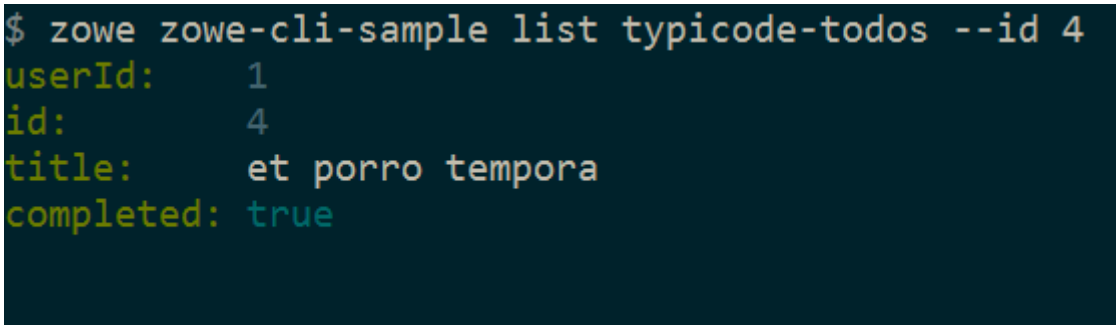
Checkpoint

Issue `npm run build` to verify a clean compilation and confirm that no lint errors are present. You now have a handler, definition, and your command has been defined to the `list` group of the command.

Using the installed plug-in

Issue the command: `zowe zowe-cli-sample list typicode-todos`

Refer to `zowe zowe-cli-sample list typicode-todos --help` for more information about your command and to see how text in the command definition is presented to the end user. You can also see how to use your optional `--id` flag:



Summary

You extended an existing Zowe CLI plug-in by introducing a Node.js programmatic API, and you created a command definition with a handler. For an official plugin, you would also add JSDoc to your code and create automated tests.

Next steps

Try the Developing a new plug-in on page 231 tutorial next to create a new plug-in for Zowe CLI.

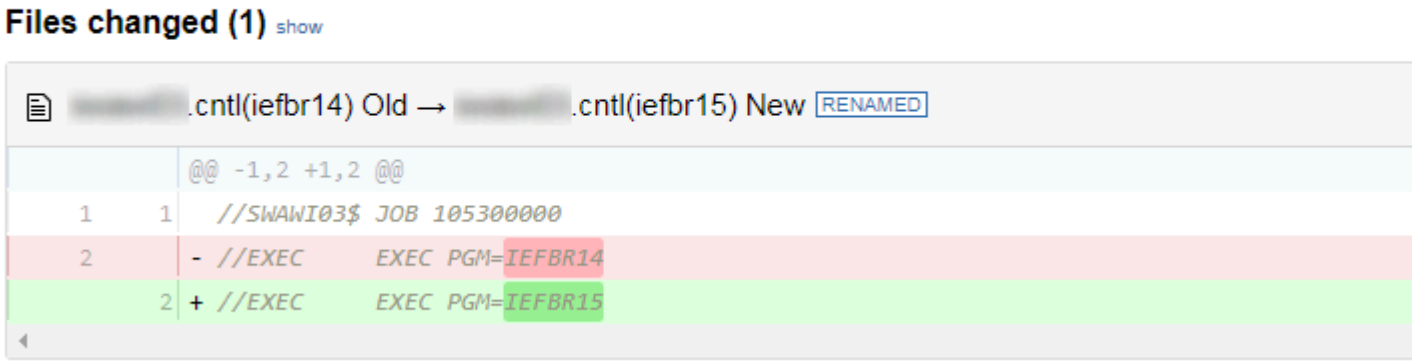
Developing a new plug-in

Before you begin this tutorial, make sure that you completed the Extending a plug-in on page 228 tutorial.

Overview

This tutorial demonstrates how to create a brand new Zowe™ CLI plug-in that uses Zowe CLI Node.js programmatic APIs.

At the end of this tutorial, you will have created a data set diff utility plug-in for Zowe CLI, from which you can pipe your plugin's output to a third-party utility for a side-by-side diff of data set member contents.



Completed source for this tutorial can be found on the `develop-a-plugin` branch of the zowe-cli-sample-plugin repository.

Cloning the sample plug-in source

Clone the sample repo, delete the irrelevant source, and create a brand new plug-in. Follow these steps:

1. `cd` into your `zowe-tutorial` folder
2. `git clone https://github.com/zowe/zowe-cli-sample-plugin files-util`
3. `cd files-util`
4. Delete the `.git` (hidden) folder.
5. Delete all content within the `src/api`, `src/cli`, and `docs` folders.
6. Delete all content within the `__tests__/__system__/api`, `__tests__/__system__/cli`, `__tests__/api`, and `__tests__/cli` folders
7. `git init`
8. `git add .`
9. `git commit -m "initial"`

Changing package.json

Use a unique `npm` name for your plugin. Change `package.json` name field as follows:

```
  "name": "@brightside/files-util",
```

Issue the command `npm install` against the local repository.

Adjusting Imperative CLI Framework configuration

Change `imperative.ts` to contain the following:

```
import { IImperativeConfig } from "@brightside/imperative";

const config: IImperativeConfig = {
    commandModuleGlobs: ["**/cli/*/*.definition!(.d).*s"],
    rootCommandDescription: "Files utilty plugin for Zowe CLI",
    envVariablePrefix: "FILES_UTIL_PLUGIN",
```

```
    defaultHome: "~/.files_util_plugin",
    productDisplayName: "Files Util Plugin",
    name: "files-util"
};

export = config;
```

Here we adjusted the description and other fields in the `imperative` JSON configuration to be relevant to this plug-in.

Adding third-party packages

We'll use the following packages to create a programmatic API:

- `npm install --save diff`
- `npm install -D @types/diff`

Creating a Node.js programmatic API

In `files-util/src/api`, create a file named `DataSetDiff.ts`. The content of `DataSetDiff.ts` should be the following:

```
import { AbstractSession } from "@brightside/imperative";
import { Download, IDownloadOptions, IZosFilesResponse } from "@brightside/
core";
import * as diff from "diff";
import { readFileSync } from "fs";

export class DataSetDiff {

    public static async diff(session: AbstractSession, oldDataSet: string,
 newDataSet: string) {

        let error;
        let response: IZosFilesResponse;

        const options: IDownloadOptions = {
            extension: "dat",
        };

        try {
            response = await Download.dataSet(session, oldDataSet, options);
        } catch (err) {
            error = "oldDataSet: " + err;
            throw error;
        }

        try {
            response = await Download.dataSet(session, newDataSet, options);
        } catch (err) {
            error = "newDataSet: " + err;
            throw error;
        }

        const regex = /\.|\(/gi; // Replace . and ( with /
        const regex2 = /\)/gi;   // Replace ) with .

        // convert the old data set name to use as a path/file
        let file = oldDataSet.replace(regex, "/");
        file = file.replace(regex2, ".") + "dat";
        // Load the downloaded contents of 'oldDataSet'
        const oldContent = readFileSync(`${file}`).toString();

        // convert the new data set name to use as a path/file
```

```
        file = newDataSet.replace(regex, "/");
        file = file.replace(regex2, ".") + "dat";
        // Load the downloaded contents of 'oldDataSet'
        const newContent = readFileSync(`${file}`).toString();

        return diff.createTwoFilesPatch(oldDataSet, newDataSet, oldContent,
 newContent, "Old", "New");
    }
}
```

Exporting your API

In `files-util/src`, change `index.ts` to contain the following:

```
export * from "./api/DataSetDiff";
```

Checkpoint

At this point, you should be able to rebuild the plug-in without errors via `npm run build`. You included third party dependencies, created a programmatic API, and customized this new plug-in project. Next, you'll define the command to invoke your programmatic API.

Defining commands

In `files-util/src/cli`, create a folder named `diff`. Within the `diff` folder, create a file `Diff.definition.ts`. Its content should be as follows:

```
import { ICommandDefinition } from "@brightside/imperative";
import { DataSetsDefinition } from "./data-sets/DataSets.definition";
const IssueDefinition: ICommandDefinition = {
    name: "diff",
    summary: "Diff two data sets content",
    description: "Uses open source diff packages to diff two data sets
 content",
    type: "group",
    children: [DataSetsDefinition]
};

export = IssueDefinition;
```

Also within the `diff` folder, create a folder named `data-sets`. Within the `data-sets` folder create `DataSets.definition.ts` and `DataSets.handler.ts`.

`DataSets.definition.ts` should contain:

```
import { ICommandDefinition } from "@brightside/imperative";

export const DataSetsDefinition: ICommandDefinition = {
    name: "data-sets",
    aliases: ["ds"],
    summary: "data sets to diff",
    description: "diff the first data set with the second",
    type: "command",
    handler: __dirname + "/DataSets.handler",
    positionals: [
        {
            name: "oldDataSet",
            description: "The old data set",
            type: "string"
        },
        {
            name: "newDataSet",
            description: "The new data set",
```

```
            type: "string"
        }
    ],
    profile: {
        required: ["zosmf"]
    }
};
```

`DataSets.handler.ts` should contain the following:

```
import { ICommandHandler, IHandlerParameters, TextUtils, Session } from
 "@brightside/imperative";
import { DataSetDiff } from "../../../api/DataSetDiff";

export default class DataSetsDiffHandler implements ICommandHandler {
    public async process(params: IHandlerParameters): Promise<void> {

        const profile = params.profiles.get("zosmf");
        const session = new Session({
            type: "basic",
            hostname: profile.host,
            port: profile.port,
            user: profile.user,
            password: profile.pass,
            base64EncodedAuth: profile.auth,
            rejectUnauthorized: profile.rejectUnauthorized,
        });
        const resp = await DataSetDiff.diff(session,
 params.arguments.oldDataSet, params.arguments.newDataSet);
        params.response.console.log(resp);
    }
}
```

Trying your command

Be sure to build your plug-in via `npm run build`.

Install your plug-in into Zowe CLI via `zowe plugins install`.

Issue the following command. Replace the data set names with valid mainframe data set names on your system:

```
$ zowe files-util diff data-sets "[redacted].cntl(iefbr14)" "[redacted]
```

The raw diff output is displayed as a command response:

```
$ zowe files-util diff data-sets "[redacted].cntl(iefbr14)" "[redacted]
==========================================================================
--- [redacted].cntl(iefbr14)       Old
+++ [redacted].cntl(iefbr15)       New
@@ -1,2 +1,2 @@
 //[redacted] $ JOB 105300000
-//EXEC     EXEC PGM=IEFBR14
+//EXEC     EXEC PGM=IEFBR15
```

Bringing together new tools!

The advantage of Zowe CLI and of the CLI approach in mainframe development is that it allows for combining different developer tools for new and interesting uses.

diff2html is a free tool to generate HTML side-by-side diffs to help see actual differences in diff output.

Install the `diff2html` CLI via `npm install -g diff2html-cli`. Then, pipe your Zowe CL plugin's output into `diff2html` to generate diff HTML and launch a web browser that contains the content in the screen shot at the Overview on page 232.

- `zowe files-util diff data-sets "kelda16.work.jcl(iefbr14)" "kelda16.work.jcl(iefbr15)" | diff2html -i stdin`

Next steps

Try the Implementing profiles in a plug-in on page 236 tutorial to learn about using profiles with your plug-in.

Implementing profiles in a plug-in

You can use this profile template to create a profile for your product.

The profile definition is placed in the `imperative.ts` file.

`someproduct` will be the profile name that you might require on various commands to have credentials loaded from a secure credential manager and retain host/port information (so that you can easily swap to different servers) from the CLI).

By default, if your plug-in is installed into Zowe™ CLI that contains a profile definition like this, commands will automatically be created under `zowe profiles ...` to create, validate, set default, list, etc... for your profile.

```
profiles: [
    {
      type: "someproduct",
      schema: {
        type: "object",
        title: "Configuration profile for SOME PRODUCT",
        description: "Configuration profile for SOME PRODUCT ",
        properties: {
          host: {
            type: "string",
            optionDefinition: {
              type: "string",
              name: "host",
              alias:["H"],
              required: true,
              description: "Host name of your SOME PRODUCT REST API server"
            }
          },
          port: {
            type: "number",
            optionDefinition: {
              type: "number",
              name: "port",
               alias:["P"],
              required: true,
              description: "Port number of your SOME PRODUCT REST API
 server"
            }
          },
          user: {
            type: "string",
            optionDefinition: {
              type: "string",
```

```
              name: "user",
              alias:["u"],
              required: true,
              description: "User name to authenticate to your SOME PRODUCT
 REST API server"
            },
            secure: true
          },
          password: {
            type: "string",
            optionDefinition: {
              type: "string",
              name: "password",
              alias:["p"],
              required: true,
              description: "Password to authenticate to your SOME PRODUCT
 REST API server"
            },
            secure: true
          },
        },
        required: ["host", "port", "user", "password"],
      },
      createProfileExamples: [
        {
         options: "spprofile --host zos123 --port 1234 --user ibmuser --
password myp4ss",
         description: "Create a SOME PRODUCT profile named 'spprofile' to
 connect to SOME PRODUCT at host zos123 and port 1234"
        }
      ]
    }
]
```

Next steps

If you completed all previous tutorials, you now understand the basics of extending and developing plug-ins for Zowe CLI. Next, we recommend reviewing the project Contribution Guidelines on page 225 and Imperative CLI Framework Documentation on page 225 to learn more.

Developing for Zowe Application Framework

Overview

You can create application plug-ins to extend the capabilities of the Zowe™ Application Framework. An application plug-in is an installable set of files that present resources in a web-based user interface, as a set of RESTful services, or in a web-based user interface and as a set of RESTful services.

Read the following topics to get started with extending the Zowe Application Framework.

How Zowe Application Framework works

Read the following topics to learn how Zowe Application Framework works:

- Building plugin apps on page 242
- Plug-ins definition and structure on page 239
- Dataservices on page 246
- Zowe Desktop and window management on page 258
- Configuration Dataservice on page 261
- URI Broker on page 267

- Application-to-application communication on page 268
- Error reporting UI on page 273
- Logging utility on page 275

Tutorials

The following tutorials are available in Github.

- **Stand up a local version of the Example Zowe Application Server**

  :::tip Github Repo: zlux-app-server :::
- **User Browser Workshop App**

  :::tip Github Repo: User Browser Workshop App :::
- **Internationalization in Angular Templates in Zowe Application Server**

  :::tip Github Sample Repo: sample-angular-app (Internationalization) :::
- **App to app communication**

  :::tip Github Sample Repo : sample-angular-app (App to app communication) :::
- **Using the Widgets Library**

  :::tip Github Sample Repo: sample-angular-app (Widgets) :::
- **Configuring user preferences (configuration dataservice)**

  :::tip Github Sample Repo: sample-angular-app (configuration dataservice) :::

Samples

Zowe allows extensions to be written in any UI framework through the use of an Iframe, or Angular and React natively. In this section, code samples of various use-cases will be provided with install instructions.

::: warning Troubleshooting Suggestions: If you are running into issues, try these suggestions:

- Restart the Zowe Server/ VM.
- Double check that the name in the plugins folder matches your identifier in `pluginsDefinition.json` located in the Zowe root.
- After logging into the Zowe desktop, use the Chrome or Firefox developer tools and navigate to the "network" tab to see what errors you are getting.
- Check each file with `cat <filename>` to be sure it wasn't corrupted while uploading. If files were corrupted, try uploading using a different method like SCP or SFTP. :::

Sample Iframe App

:::tip Github Sample Repo: sample-iframe-app :::

Sample Angular App

:::tip Github Sample Repo: sample-angular-app :::

Sample React App

:::tip Github Sample Repo: sample-react-app :::

User Browser Workshop Starter App

:::tip Github Sample Repo: workshop-starter-app :::

This sample is included as the first part of a tutorial detailing communication between separate Zowe apps.

It should be installed on your system before starting the User Browser Workshop App Tutorial

The App's scenario is that it has been opened to submit a task report to a set of users who can handle the task. In this case, it is a bug report. We want to find engineers who can fix this bug, but this App does not contain a directory listing for engineers in the company, so we need to communicate with some App that does provide this information.

In this tutorial, you must build an App which is called by this App in order to list engineers, is able to be filtered by the office that they work from, and is able to submit a list of engineers which would be able to handle the task.

After installing this app on your system, follow directions in the User Browser Workshop App Tutorial to enable app-to-app communication.

Plug-ins definition and structure

The Zowe™ Application Server (`zlux-server-framework`) enables extensiblity with application plug-ins. Application plug-ins are a subcategory of the unit of extensibility in the server called a *plug-in*.

The files that define a plug-in are located in the `pluginsDir` directory.

Application plug-in filesystem structure

An application plug-in can be loaded from a filesystem that is accessible to the Zowe Application Server, or it can be loaded dynamically at runtime. When accessed from a filesystem, there are important considerations for the developer and the user as to where to place the files for proper build, packaging, and operation.

Root files and directories

The root of an application plug-in directory contains the following files and directories.

pluginDefinition.json

This file describes an application plug-in to the Zowe Application Server. (A plug-in is the unit of extensibility for the Zowe Application Server. An application plug-in is a plug-in of the type "Application", the most common and visible type of plug-in.) A definition file informs the server whether the application plug-in has server-side dataservices, client-side web content, or both. The attributes of this file and how it is found by the server are described in the Plugin Definition article.

Dev and source content

Aside from demonstration or open source application plug-ins, the following directories should not be visible on a deployed server because the directories are used to build content and are not read by the server.

nodeServer

When an application plug-in has router-type dataservices, they are interpreted by the Zowe Application Server by attaching them as ExpressJS routers. It is recommended that you write application plug-ins using Typescript, because it facilitates well-structured code. Use of Typescript results in build steps because the pre-transpilation Typescript content is not to be consumed by NodeJS. Therefore, keep server-side source code in the `nodeServer` directory. At runtime, the server loads router dataservices from the lib on page 239 directory.

webClient

When an application plug-in has the *webContent* attribute in its definition, the server serves static content for a client. To optimize loading of the application plug-in to the user, use Typescript to write the application plug-in and then package it using Webpack. Use of Typescript and Webpack result in build steps because the pre-transpilation Typescript and the pre-webpack content are not to be consumed by the browser. Therefore, separate the source code from the served content by placing source code in the `webClient` directory.

Runtime content

At runtime, the following set of directories are used by the server and client.

lib

The `lib` directory is where router-type dataservices are loaded by use in the Zowe Application Server. If the JS files that are loaded from the `lib` directory require NodeJS modules, which are not provided by the server base (the modules `zlux-server-framework` requires are added to `NODE_PATH` at runtime), then you must include these modules in `lib/node_modules` for local directory lookup or ensure that they are found on the `NODE_PATH` environment variable. `nodeServer/node_modules` is not automatically accessed at runtime because it is a dev and build directory.

web

The `web` directory is where the server serves static content for an application plug-in that includes the *webContent* attribute in its definition. Typically, this directory contains the output of a webpack build. Anything you place in this directory can be accessed by a client, so only include content that is intended to be consumed by clients.

Packaging applications as compressed files

Application plug-in files can be served to browsers as compressed files in brotli (.br) or gzip (.gz) format. The file must be below the application's `/web` directory, and the browser must support the compression method. If there are multiple compressed files in the `/web` directory, the Zowe Application Server and browser perform runtime negotiation to decide which file to use.

Location of plug-in files

The files that define a plug-in are located in the `plugins` directory.

pluginsDir directory

At startup, the server reads from the `plugins` directory. The server loads the valid plug-ins that are found by the information that is provided in the JSON files.

Within the `pluginsDir` directory are a collection of JSON files. Each file has two attributes, which serve to locate a plug-in on disk:

location: This is a directory path that is relative to the server's executable (such as `zlux-app-server/bin/nodeServer.sh`) at which a `pluginDefinition.json` file is expected to be found.

identifier: The unique string (commonly styled as a Java resource) of a plug-in, which must match what is in the `pluginDefinition.json` file.

Plug-in definition file

`pluginDefinition.json` is a file that describes a plug-in. Each plug-in requires this file, because it defines how the server will register and use the backend of an application plug-in (called a *plug-in* in the terminology of the proxy server). The attributes in each file are dependent upon the `pluginType` attribute. Consider the following `pluginDefinition.json` file from `sample-app`:

```
{
  "identifier": "com.rs.mvd.myplugin",
  "apiVersion": "1.0",
  "pluginVersion": "1.0",
  "pluginType": "application",
  "webContent": {
    "framework": "angular2",
    "launchDefinition": {
      "pluginShortNameKey": "helloWorldTitle",
      "pluginShortNameDefault": "Hello World",
      "imageSrc": "assets/icon.png"
    },
    "descriptionKey": "MyPluginDescription",
    "descriptionDefault": "Base MVD plugin template",
    "isSingleWindowApp": true,
    "defaultWindowStyle": {
      "width": 400,
      "height": 300
    }
  },
  "dataServices": [
    {
      "type": "router",
      "name": "hello",
      "serviceLookupMethod": "external",
      "fileName": "helloWorld.js",
      "routerFactory": "helloWorldRouter",
```

```
      "dependenciesIncluded": true
    }
  ]
}
```

Plug-in attributes

There are two categories of attributes: General and Application.

General attributes

identifier

Every application plug-in must have a unique string ID that associates it with a URL space on the server.

apiVersion

The version number for the pluginDefinition scheme and application plug-in or dataservice requirements. The default is 1.0.0.

pluginVersion

The version number of the individual plug-in.

pluginType

A string that specifies the type of plug-in. The type of plug-in determines the other attributes that are valid in the definition.

- **application**: Defines the plug-in as an application plug-in. Application plug-ins are composed of a collection of web content for presentation in the Zowe web component (such as the Zowe Desktop), or a collection of dataservices (REST and websocket), or both.
- **library**: Defines the plug-in as a library that serves static content at a known URL space.
- **node authentication**: Authentication and Authorization handlers for the Zowe Application Server.

Application attributes

When a plug-in is of *pluginType* application, the following attributes are valid:

webContent

An object that defines several attributes about the content that is shown in a web UI.

dataServices

An array of objects that describe REST or websocket dataservices.

configurationData

An object that describes the resource structure that the application plug-in uses for storing user, group, and server data.

Application web content attributes

An application that has the *webContent* attribute defined provides content that is displayed in a Zowe web UI.

The following attributes determine some of this behavior:

framework

States the type of web framework that is used, which determines the other attributes that are valid in *webContent*.

- **angular2**: Defines the application as having an Angular (2+) web framework component. This is the standard for a "native" framework Zowe application.
- **iframe**: Defines the application as being external to the native Zowe web application environment, but instead embedded in an iframe wrapper.

launchDefinition

An object that details several attributes for presenting the application in a web UI.

- **pluginShortNameDefault**: A string that gives a name to the application when i18n is not present. When i18n is present, i18n is applied by using the *pluginShortNameKey*.
- **descriptionDefault**: A longer string that specifies a description of the application within a UI. The description is seen when i18n is not present. When i18n is present, i18n is applied by using the *descriptionKey*.
- **imageSrc**: The relative path (from `/web`) to a small image file that represents the application icon.

defaultWindowStyle

An object that details the placement of a default window for the application in a web UI.

- **width**: The default width of the application plug-in window, in pixels.
- **height**: The default height of the application plug-in window, in pixels.

IFrame application web content

In addition to the general web content attributes, when the framework of an application is "iframe", you must specify the page that is being embedded in the iframe. To do so, incude the attribute *startingPage* within *webContent*. *startingPage* is relative to the application's `/web` directory.

Specify *startingPage* as a relative path rather than an absolute path because the `pluginDefinition.json` file is intended to be read-only, and therefore would not work well when the hostname of a page changes.

Within an IFrame, the application plug-in still has access to the globals that are used by Zowe for application-to-application communication; simply access *window.parent.ZoweZLUX*.

Building plugin apps

You can build a plugin app by using the following steps as a model. Alternatively, you can follow the Sample Angular App tutorial.

The basic requirement for a plugin app is that static web content must be in a `/web` directory, and server and other backend files must be in a `/lib` directory. You can place other plugin source code anywhere.

Before you can build a plugin app you must install all prerequisites.

Building web content

1. On the computer where the virtual desktop is installed, use the the following command to specify a value for the `MVD_DESKTOP_DIR` environment variable:

   ```
   export MVD_DESKTOP_DIR=/<path>/zowe/zlux-app-manager/virtual-desktop
   ```

   Where `<path>` is the install location of the virtual desktop.
2. Navigate to `/<plugin_dir>/webClient`. If there is no `/webClient` directory, proceed to the **Building server content** section below.
3. Run the `npm install` command to install any application dependencies. Check for successful return code.
4. Run one of the following commands to build the application code:
   - Run the `npm run build` command to generate static content in the `/web` directory. (You can ignore warnings as long as the build is successful.)
   - Run the `npm run start` command to compile in real-time. Until you stop the script, it compiles code changes as you make them.

Building server content

1. Navigate to the plugin directory. If there is no `/nodeServer` directory in the plugin directory, procede to the **Building Javascript content (*.js files)** section below.
2. Run the `npm install` command to install any application dependencies. Check for successful return code.

3. Run one of the following commands to build the application code:
   - Run the `npm run build` command to generate static content in the `/lib` directory.
   - Run the `npm run start` command to compile in real-time. Until you stop the script, it compiles code changes as you make them.

Building Javascript content (*.js files)

Unlike Typescript, Javascript is an interpreted language and does not need to be built. In most cases, reloading the page should build new code changes. For Iframes or other JS-based apps, close and open the app.

Installing

Follow the steps described in Installing Plugins on page 243 to add your built plugin to the Zowe desktop.

Packaging

For more information on how to package your Zowe app, developers can see Plug-ins definition and structure on page 239.

Installing Plugins

Plugins can be added or removed from the Zowe App Server, as well as upgraded. There are two ways to do these actions: By REST API or by filesystem. The instructions below assume you have administrative permissions either to access the correct REST APIs or to have the necessary permissions to update server directories & files.

NOTE: To successfully install, you must pre-build plugins with the correct directory structure, and meet all dependencies. If a plugin has successfully installed but does not display in the Zowe desktop, see the server log in the `<INSTANCE_DIR>/log/install-app.log` directory (for example, `~/.zowe/log/install-app.log`) to troubleshoot the problem.

By filesystem

The App server uses directories of JSON files, described in the wiki. Defaults are located in the folder `zlux-app-server/defaults/plugins`, but the server reads the list of plugins instead from the instance directory, at `<INSTANCE_DIR>/workspace/app-server/plugins` (for example, `~/.zowe/workspace/app-server/plugins` which includes JSON files describing where to find a plugin. Adding or removing JSONs from this folder will add or remove plugins upon server restart, or you can use REST APIs and cluster mode to add or remove plugins without restarting).

Old plugins folder

Prior to Zowe release 1.8.0, the location of the default and instance plugins directory were located within `zlux-app-server` folder unless otherwise customized. 1.8.0 has backwards compatibility for the existence of these directories, but they can and should be migrated to take advantage of future enhancements.

Folder	New Location	Old Location	Note
Default plugins	`zlux-app-server/defaults/plugins`	`zlux-app-server/plugins`	
Instance plugins	`<INSTANCE_DIR>/workspace/app-server/plugins`	`zlux-app-server/instance/ZLUX/plugins`	INSTANCE_DIR is ~/.zowe if not otherwise defined

Adding/Installing

To add or install a plugin, run the script `zlux-app-server/bin/install-app.sh` providing the location to a plugin folder. For example:

```
./install-app.sh /home/john/zowe/sample-angular-app
```

This will generate a JSON file that states the plugin's ID and its location on disk. These JSON files tell the Desktop where to find apps. For example, if we were to install the sample angular-app in the folder `/home/john/zowe/sample-angular-app`, then the JSON would be:

```
{
  "identifier":"org.zowe.zlux.sample.angular",
  "location": "/home/john/zowe/sample-angular-app"
}
```

Removing

To remove a plugin, locate the server's instance plugin directory `<INSTANCE_DIR>/workspace/app-server/plugins` (for example, `~/.zowe/workspace/app-server/plugins`) and remove the locator JSON that is associated with that plugin. Remove the plugin's content by deleting it from the file system if applicable.

Upgrading

Currently, only one version of a plugin can exist per server. So, to upgrade, you either upgrade the plugin within its pre-existing directory by rebuilding it (with more up to date code), or you alter the locator JSON of that app to point to the content of the upgraded version.

Modifying without server restart (Exercise to the reader)

The server's reading of the locator JSONs and initializing of plugins only happens during bootstrapping at startup. However, in cluster mode the bootstrapping happens once per worker process. Therefore, it is possible to manage plugins without a server restart by killing & respawning all worker processes without killing the cluster master process. This is what the REST API does, internally. To do this without the REST API, it may be possible to script knowing the parent process ID, and running a kill command on all child processes of the App server cluster process.

By REST API

The server REST APIs allow plugin management without restarting the server - you can add, remove, and upgrade plugins in real-time. However, removal or upgrade must be done carefully as it can disrupt users of those plugins.

This swagger file documents the REST API for plugin management

The API only works when RBAC is configured, and an RBAC-compatible security plugin is being used. An example of this is zss-auth, and use of RBAC is described in this documentation and in the wiki.

Embedding plugins

Add these imports to a component where you want to embed another plugin:

app.component.ts

```
import {Inject, Injector, ViewChild, ViewContainerRef} from '@angular/core';

import {Angular2InjectionTokens, Angular2PluginEmbedActions,
 EmbeddedInstance} from 'pluginlib/inject-resources';
```

Inject Angular2PluginEmbedActions into your component constructor:

app.component.ts

```
 constructor(@Inject(Angular2InjectionTokens.PLUGIN_EMBED_ACTIONS) private
 embedActions: Angular2PluginEmbedActions) {
    }
```

In the component template prepare a container where you want to embed the plugin:

app.component.html

```
    <div class="container-for-embedded-window">
       <ng-container #embedded></ng-container>
    </div>
```

In the component class add a reference to the container:

app.component.ts

```
    @ViewChild('embedded', {read: ViewContainerRef}) viewContainerRef:
 ViewContainerRef;
```

In the component class add a reference to the embedded instance:

app.component.ts

```
    private embeddedInstance: EmbeddedInstance;
```

Everything is ready to start embedding, you just need to know the pluginId that you want to embed:

app.component.ts

```
    embedPlugin(): void {
      const pluginId = 'org.zowe.zlux.sample.angular';
      const launchMetadata = null;
      this.embedActions.createEmbeddedInstance(pluginId, launchMetadata,
 this.viewContainerRef)
        .then(embeddedInstance => this.embeddedInstance = embeddedInstance)

        .catch(e => console.error(`couldn't embed plugin ${pluginId} because
 ${e}`));
    }
```

How to interact with embedded plugin

If the main component of embedded plugin declares Input and Output properties then you can interact with it. ApplicationManager provides methods to set Input properties and get Output properties of the embedded plugin. Suppose, that the embedded plugin declares Input and Output properties like this:

plugin.component.ts

```
    @Input() sampleInput: string;
    @Output() sampleOutput: EventEmitter<string> = new
 EventEmitter<string>();
```

Obtain a reference to AppicationManager in your component constructor:

app.component.ts

```
    private applicationManager: MVDHosting.ApplicationManagerInterface;
    constructor(
      @Inject(Angular2InjectionTokens.PLUGIN_EMBED_ACTIONS) private
 embedActions: Angular2PluginEmbedActions,
```

```
      // @Inject(MVDHosting.Tokens.ApplicationManagerToken) private
 applicationManager: MVDHosting.ApplicationManagerInterface
      injector: Injector
    ) {
      this.applicationManager =
 this.injector.get(MVDHosting.Tokens.ApplicationManagerToken);
    }
```

Note: We are unable to inject `AppicationManager` with `@Inject()` until an AoT-compiler issue with namespaces is resolved: angular/angular#15613

Now you can set `sampleInput` property, obtain `sampleOutput` property and subscribe to it:

app.component.ts

```
this.applicationManager.setEmbeddedInstanceInput(this.embeddedInstance,
 'sampleInput', 'some value');

    const sampleOutput: Observable<string> =
 this.applicationManager.getEmbeddedInstanceOutput(this.embeddedInstance,
 'sampleOutput');
    sampleOutput.subscribe(value => console.log(`get new value ${value} from
 sampleOutput`));
```

How to destroy embedded plugin

There is no special API to destroy embedded plugin. If you want to destroy the embedded plugin just clear the container for the embedded plugin and set `embeddedInstance` to null:

app.component.ts

```
    this.viewContainerRef.clear();
    this.embeddedInstance = null;
```

How to style a container for the embedded plugin

It is hard to give a universal recipe for a container style. At least, the container needs `position: "relative"` because the embedded plugin may have absolutely positioned elements. Here is sample styles you can start with if your component utilizes flexbox layout:

app.component.css

```
    .container-for-embedded-window {
      position: relative;
      flex: 1 1 auto;
      align-self: stretch;
      display: flex;
      flex-direction: column;
      align-items: stretch;
    }
```

Applications that use embedding

Workflow app demonstrates advanced usage.

Dataservices

Dataservices are dynamic backend components of Zowe™ plug-in applications. You can optionally add them to your applications to make the application do more than receive static content from the proxy server. Each dataservice

defines a URL space that the server can use to run extensible code from the application. Dataservices are mainly intended to create REST APIs and WebSocket channels.

Defining dataservices

You define dataservices in the application's `pluginDefinition.json` file. Each application requires a definition file to specify how the server registers and uses the application's backend. You can see an example of a `pluginDefinition.json` file in the top directory of the sample-angular-app.

In the definition file is a top level attribute called `dataServices`, for example:

```
"dataServices": [
  {
    "type": "router",
    "name": "hello",
    "serviceLookupMethod": "external",
    "fileName": "helloWorld.js",
    "routerFactory": "helloWorldRouter",
    "dependenciesIncluded": true
  }
]
```

To define your dataservice, create a set of keys and values for your dataservice in the `dataservices` array. The following values are valid:

type

Specify one of the following values:

- **router**: Router dataservices run under the proxy server and use ExpressJS Routers for attaching actions to URLs and methods.
- **service**: Service dataservices run under ZSS and utilize the API of ZSS dataservices for attaching actions to URLs and methods.
- **java-war**: See the topic *Defining Java dataservices* below.

name

The name of the service. Names must be unique within each `pluginDefinition.json` file. The name is used to reference the dataservice during logging and to construct the URL space that the dataservice occupies.

serviceLookupMethod

Specify `external` unless otherwise instructed.

fileName

The name of the file that is the entry point for construction of the dataservice, relative to the application's `/lib` directory. For example, for the `sample-app` the `fileName` value is `"helloWorld.js"` - without a path. So its typescript code is transpiled to JavaScript files that are placed directly into the `/lib` directory.

routerFactory (Optional)

When you use a router dataservice, the dataservice is included in the proxy server through a `require()` statement. If the dataservice's exports are defined such that the router is provided through a factory of a specific name, you must state the name of the exported factory using this attribute.

dependenciesIncluded

Specify `true` for anything in the `pluginDefinition.json` file. Only specify `false` when you are adding dataservices to the server dynamically.

Defining Java dataservices

In addition to other types of dataservice, you can use Java (also called java-war) dataservices in your applications. Java dataservices are powered by Java Servlets.

To use a Java dataservice you must meet the prerequisites, define the dataservice in your plug-in definition, and define the Java Application Server library to the Zowe Application Server.

Prerequisites

- Install a Java Application Server library. In this release, Tomcat is the only supported library.
- Make sure your plug-in's compiled Java program is in the application's `/lib` directory, in either a `.war` archive file or a directory extracted from a `.war` archive file. Extracting your file is recommended for faster start-up time.

Defining Java dataservices

To define the dataservice in the `pluginDefinition.json` file, specify the type as `java-war`, for example:

```
"dataServices": [
    {
      "type": "java-war",
      "name": "javaservlet",
      "filename": "javaservlet.war",
      "dependenciesIncluded": true,
      "initializerLookupMethod": "external",
      "version": "1.0.0"
    }
  ],
```

To access the service at runtime, the plug-in can use the Zowe dataservice URL standard: `/ZLUX/plugins/[PLUGINID]/services/[SERVICENAME]/[VERSIONNUMBER]`

Using the example above, a request to get users might be: `/ZLUX/plugins/[PLUGINID]/services/javaservlet/1.0.0/users`

Note: If you extracted your servlet contents from a `.war` file to a directory, the directory must have the same name as the file would have had. Using the example above, `javaservlet.war` must be extracted to a directory named `\javaservlet`.

Defining Java Application Server libraries

In the `zlux-app-server/zluxserver.json` file, use the example below to specify Java Application Server library parameters:

```
"languages": {
    "java": {
      "runtimes": {
          "name": {
              "home": "<java_runtime_root_path>"
          }
      }
      "war": {
        "defaultGrouping": "<value>"
        "pluginGrouping": []
        "javaAppServer": {
          "type": "tomcat",
          "path": "../../zlux-server-framework/lib/java/apache-tomcat",
          "config": "../deploy/instance/ZLUX/serverConfig/tomcat.xml",
          "https": {
            "key": "../deploy/product/ZLUX/serverConfig/zlux.keystore.key",
            "certificate": "../deploy/product/ZLUX/serverConfig/
zlux.keystore.cer"
          }
        }
      },
      "portRange": [8545,8600]
    }
  }
```

Specify the following parameters in the `languages.java` object:

- `runtimes` (object) - The name and location of a Java runtime that can be used by one or more services. Used to load a Tomcat instance.
  - `name` (object) - The name of the runtime.
    - `home` (string) - The path to the runtime root. Must include `/bin` and `/lib` directories.
- `ports` (array`<number>`)(Optional) - An array of port numbers that can be used by instances of Java Application Servers or microservices. Must contain as many ports as distinct servers that will be spawned, which is defined by other configuration values within `languages.java`. Either `ports` or `portRange` is required, but `portRange` has a higher priority.
- `portRange` (array`<number>`)(Optional) - An array of length 2, which contains a start number and end number to define a range of ports to be used by instances of application servers or microservices. You will need as many ports as distinct servers that will be spawned, which is defined by other configuration values within `languages.java`. Either `ports` or `portRange` is required, but `portRange` has a higher priority.
- `war` (object) - Defines how the Zowe Application Server should handle `java-war` dataservices.
  - **defaultGrouping** (string)(Optional) - Defines how services should be grouped into instances of Java Application Servers. Valid values: `appserver` or `microservice`. Default: `appserver`. `appserver` means 1 server instance for all services. `microservice` means one server instance per service.
  - **pluginGrouping** (array`<object>`)(Optional) - Defines groups of plug-ins to have their `java-war` services put within a single Java Application Server instance.
    - **plugins** (Array`<string>`) - Lists the plugins by identifier which should be put into this group. Plug-ins with no `java-war` services are skipped. Being in a group excludes a plugin from being handled by `defaultGrouping`.
    - **runtime** (string)(Optional) - States the runtime to be used by the Tomcat server instance, as defined in `languages.java.runtimes`.
  - **javaAppServer** (object) - Java Application Server properties.
    - **type** (string) - Type of server. In this release, `tomcat` is the only valid value.
    - **path** (string) - Path of the server root, relative to `zlux-app-server/lib`. Must include `/bin` and `/lib` directories.
    - **config** (string) - Path of the server configuration file, relative to `zlux-app-server/lib`.
    - **https** (object) - HTTPS parameters.
      - **key** (string) - Path of a private key, relative to `zlux-app-server/lib`.
      - **certificate** (string) - Path of an HTTPS certificate, relative to `zlux-app-server/lib`.

Java dataservice logging

The Zowe Application Server creates the Java Application Server instances required for the `java-war` dataservices, so it logs the stdout and stderr streams for those processes in its log file. Java Application Server logging is not managed by Zowe at this time.

Java dataservice limitations

Using Java dataservices with a Zowe Application Server installed on a Windows computer, the source and Java dataservice code must be located on the same storage volume.

To create multiple instances of Tomcat on non-Windows computers, the Zowe Application Server establishes symbolic links to the service logic. On Windows computers, symbolic links require administrative privilege, so the server establishes junctions instead. Junctions only work when the source and destination reside on the same volume.

Using dataservices with RBAC

If your administrator configures the Zowe Application Framework to use role-based access control (RBAC), then when you create a dataservice you must consider the length of its paths.

To control access to dataservices, administrators can enable RBAC, then use a z/OS security product such as RACF to map roles and authorities to a System Authorization Facility (SAF) profile. For information on RBAC, see Applying role-based access control to dataservices.

SAF profiles have the following format:

```
<product>.<instance id>.SVC.<pluginid_with_underscores>.<service>.<HTTP
method>.<dataservice path with forward slashes '/' replaced by periods '.'>
```

For example, to access this dataservice endpoint:

```
/ZLUX/plugins/org.zowe.foo/services/baz/_current/users/fred
```

Users must have READ access to the following profile:

```
ZLUX.DEFAULT.SVC.ORG_ZOWE_FOO.BAZ.POST.USERS.FRED
```

Profiles cannot contain more than 246 characters. If the path section of an endpoint URL makes the profile name exceed limit, the path is trimmed to only include elements that do not exceed the limit. For example, imagine that each path section in this endpoint URL contains 64 characters:

```
/ZLUX/plugins/org.zowe.zossystem.subsystems/services/data/_current/aa..a/
bb..b/cc..c/dd..d
```

So `aa..a` is 64 "a" characters, `bb..b` is 64 "b" characters, and so on. The URL could then map to the following example profile:

```
ZLUX.DEFAULT.SVC.ORG_ZOWE_ZOSSYSTEM_SUBSYSTEMS.DATA.GET.AA..A.BB..B
```

The profile ends at the `BB..B` section because adding `CC..C` would put it over 246 characters. So in this example, all dataservice endpoints with paths that start with `AA..A.BB..B` are controlled by this one profile.

To avoid this issue, we recommend that you maintain relatively short endpoint URL paths.

Dataservice APIs

Dataservice APIs can be categorized as Router-based or ZSS-based, and either WebSocket or not.

Router-based dataservices

Each Router dataservice can safely import Express, express-ws, and bluebird without requiring the modules to be present, because these modules exist in the proxy server's directory and the *NODE_MODULES* environment variable can include this directory.

HTTP/REST Router dataservices

Router-based dataservices must return a (bluebird) Promise that resolves to an ExpressJS router upon success. For more information, see the ExpressJS guide on use of Router middleware: Using Router Middleware.

Because of the nature of Router middleware, the dataservice need only specify URLs that stem from a root '/' path, as the paths specified in the router are later prepended with the unique URL space of the dataservice.

The Promise for the Router can be within a Factory export function, as mentioned in the `pluginDefinition` specification for *routerFactory* above, or by the module constructor.

An example is available in `sample-app/nodeServer/ts/helloWorld.ts`

WebSocket Router dataservices

ExpressJS routers are fairly flexible, so the contract to create the Router for WebSockets is not significantly different.

Here, the express-ws package is used, which adds WebSockets through the ws package to ExpressJS.
The two changes between a WebSocket-based router and a normal router are that the method is 'ws', as in `router.ws(<url>,<callback>)`, and the callback provides the WebSocket on which you must define event listeners.

See the ws and express-ws topics on www.npmjs.com for more information about how they work, as the API for WebSocket router dataservices is primarily provided in these packages.

An example is available in `zlux-server-framework/plugins/terminal-proxy/lib/terminalProxy.js`

Router dataservice context

Every router-based dataservice is provided with a `Context` object upon creation that provides definitions of its surroundings and the functions that are helpful. The following items are present in the `Context` object:

serviceDefinition

The dataservice definition, originally from the `pluginDefinition.json` file within a plug-in.

serviceConfiguration

An object that contains the contents of configuration files, if present.

logger

An instance of a Zowe Logger, which has its component name as the unique name of the dataservice within a plug-in.

makeSublogger

A function to create a Zowe Logger with a new name, which is appended to the unique name of the dataservice.

addBodyParseMiddleware

A function that provides common body parsers for HTTP bodies, such as JSON and plaintext.

plugin

An object that contains more context from the plug-in scope, including:

- **pluginDef**: The contents of the `pluginDefinition.json` file that contains this dataservice.
- **server**: An object that contains information about the server's configuration such as:
  - **app**: Information about the product, which includes the *productCode* (for example: `ZLUX`).
  - **user**: Configuration information of the server, such as the port on which it is listening.

Documenting dataservices

It is recommended that you document your RESTful application dataservices in OpenAPI (Swagger) specification documents. The Zowe Application Server hosts Swagger files for users to view at runtime.

To document a dataservice, take the following steps:

1. Create a `.yaml` or `.json` file that describes the dataservice in valid Swagger 2.0 format. Zowe validates the file at runtime.
2. Name the file with the same name as the dataservice. Optionally, you can include the dataservice version number in the format: `<name>_<number>`. For example, a Swagger file for a dataservice named `user` must be named either `users.yaml` or `users_1.1.0.yaml`.
3. Place the Swagger file in the `/doc/swagger` directory below your application plug-in directory, for example:

   `/zlux-server-framework/plugins/<servicename>/doc/swagger/<servicename_1.1.0>.yaml`

At runtime, the Zowe Application Server does the following:

- Dynamically substitutes known values in the files, such as the hostname and whether the endpoint is accessible using HTTP or HTTPS.
- Builds documentation for each dataservice and for each application plug-in, in the following locations:
  - Dataservice documentation: `/ZLUX/plugins/<app_name>/catalogs/swagger/servicename`
  - Application plug-in documentation: `/ZLUX/plugins/<app_name>/catalogs/swagger`
- In application plug-in documentation, displays only stubs for undocumented dataservices, stating that the dataservice exists but showing no details. Undocumented dataservices include non-REST dataservices such as WebSocket services.

Authentication API

This topic describes the web service API for user authentication.

The authentication mechanism of the ZLUX server allows for an administrator to gate access to services by a given auth handler, while on the user side the authentication structure allows for a user to login to one or more endpoints at once provided they share the same credentials given.

Check status

Returns the current authentication status of the user to the caller.

```
GET /auth
```

Response example:

```
{
  "categories": {
    "zss": {
      "authenticated": true,
      "plugins": {
        "org.zowe.zlux.auth.zss": {
          "authenticated": true,
          "username":"foo"
        }
      }
    },
    "zosmf": {
      "authenticated": false,
      "plugins": {
        "org.zowe.zlux.auth.zosmf": {
          "authenticated": false
        }
      }
    }
  }
}
```

Every key in the response object is a registered auth type. The value object is guaranteed to have a Boolean field named "authenticated" which indicates that at least one plugin in the category was able to authenticate the user.

Each item also has a field called "plugins", where every property value is a plugin-specific object.

Authenticate

Authenticates the user against authentication back-ends.

```
POST /auth
```

Request body example:

```
{
  "categories": ["zosmf"],
  "username": "foo",
  "password": "1970-01-01"
}
```

The categories parameter is optional. If omitted, all auth plugins are invoked with the username and password
Response example:

```
{
  "success": true,
```

```
  "categories": {
    "zss": {
      "success": true,
      "plugins": {
        "org.zowe.zlux.auth.zss": {
          "success": true
        }
      }
    },
    "zosmf": {
      "success": true,
      "plugins": {
        "org.zowe.zlux.auth.zosmf": {
          "success": true
        }
      }
    }
  }
}
```

First-level keys are authentication categories or types. "success" means that all of the types requested have been successful. For example typeA successful AND typeB succesful AND ...

Second-level keys are auth plugin IDs. "success" on this level means that there's at least one successful result in that auth type. For example, pluginA successful OR pluginB successful OR ...

User not authenticated or not authorized

The response received by the browser when calling any service, when the user is either not authenticated or not allowed to access the service.

Not authenticated

```
HTTP 401

{
  "category": "zss",
  "pluginID": "org.zowe.zlux.auth.zss",
  "result": {
    "authenticated": false,
    "authorized": false
  }
}
```

The client is supposed to address this by showing the user a login form which will later invoke the login service for the plugin mentioned and repeat the request.

Not authorized

```
HTTP 403

{
  "category": "zss",
  "pluginID": "org.zowe.zlux.auth.zss",
  "result": {
    "authenticated": true,
    "authorized": false
  }
}
```

There's no general way for the client to address this, except than show the user an error message.

Internationalizing applications

You can internationalize Zowe™ application plug-ins using Angular and React frameworks. Internationalized applications display in translated languages and include structures for ongoing translation updates.

The steps below use the Zowe Sample Angular Application and Zowe Sample React Application as examples. Your applications might have slightly different requirements, for example the React Sample Application requires the react-i18next library, but your application might require a different React library.

For detailed information on Angular or React, see their documentation. For detailed information on specific internationalization libraries, see their documentation. You can also reference the Sample Angular Application internationalization tutorial, and watch a video on how to internationalize your Angular application.

After you internationalize your application, you can view it by following steps in Changing the desktop language on page 146.

Internationalizing Angular applications

Zowe applications that use the Angular framework depend on `.xlf` formatted files to store static translated content and `.json` files to store dynamic translated content. These files must be in the application's `web/assets/i18n` folder at runtime. Each translated language will have its own file.

To internationalize an application, you must install Angular-compatible internationalization libraries. Be aware that libraries can be better suited to either static or dynamic HTML elements. The examples in this task use the ngx-i18nsupport library for static content and angular-l10n for dynamic content.

To internationalize Zowe Angular applications, take the following steps:

1. To install internationalization libraries, use the `npm` command, for example:

   ```
   npm install --save-dev ngx-i18nsupport
   npm install --save-dev angular-l10n
   ```

   Note `--save-dev` commits the library to the application's required libraries list for future use.
2. To support the CLI tools and to control output, create a `webClient/tsconfig.i18n.json` typescript file and add the following content:

   ```
      {
     "extends": "../../zlux-app-manager/virtual-desktop/plugin-config/
   tsconfig.ngx-i18n.json",

     "include": [
    "./src"
     ],

     "compilerOptions": {
       "outDir": "./src/assets/i18n",
       "skipLibCheck": true
     }
   }
   ```

   For example, see this file in the Sample Angular Application.
3. In the static elements in your HTML files, tag translatable content with the i18n attribute within an Angular template, for example:

   ```
   <div>
       <p i18n="welcome message@@welcome">Welcome</p>
   </div>
   ```

   The attribute should include a message ID, for example the `@@welcome` above.

4. To configure static translation builds, take the following steps:

   a. In the `webClient/package.json` script, add the following line:

   ```
   "i18n": "ng-xi18n -p tsconfig.i18n.json --i18nFormat=xlf --
   outFile=messages.xlf && xliffmerge -p xliffmerge.json",
   ```

   b. In the in `webClient` directory, create a `xliffmerge.json` file, add the following content, and specify the codes for each language you will translate in the `languages` parameter:

   ```
   {
     "xliffmergeOptions": {
       "srcDir": "src/assets/i18n",
       "genDir": "src/assets/i18n",
       "i18nFile": "messages.xlf",
       "i18nBaseFile": "messages",
       "i18nFormat": "xlf",
       "encoding": "UTF-8",
       "defaultLanguage": "en",
       "languages": ["fr","ru"],
       "useSourceAsTarget": true
     }
   }
   ```

   When you run the i18n script, it reads this file and generates a `messages.[lang].xlf` file in the `src/assets/i18n directory` for each language specified in the `languages` parameter. Each file contains the untranslated text from the i18n-tagged HTML elements.
5. Run the following command to run the i18n script and extract i18n tagged HTML elements to `.xlf` files:

   ```
   npm run i18n
   ```

   Note If you change static translated content, you must run the `npm run build` command to build the application, and then re-run the `npm run i18n` command to extract the tagged content again.
6. In each `.xlf` file, replace `target` element strings with translated versions of the `source` element strings. For example:

   ```
   <source>App Request Test</source>
   <target>Test de Demande à l'App</target>
   ```
7. Run the following command to rebuild the application:

   ```
   npm run build
   ```

   When you Changing the desktop language on page 146 to one of the application's translated languages, the application displays the translated strings.
8. For dynamic translated content, follow these steps:

   a. Import and utilize angular-l10n objects within an Angular component, for example:

   ```
   import { LocaleService, TranslationService, Language } from 'angular-
   l10n';
   Component({
     selector: 'app-root',
     templateUrl: './app.component.html',
     styleUrls: ['./app.component.css'],
     providers: [HelloService]
   })

   export class AppComponent {
     @Language() lang: string;
   ```

```
   public myDynamicMessage:string = '';

  constructor(
    public locale: LocaleService,
    public translation: TranslationService) { }

  sayHello() {
     this.myDynamicMessage =    `
${this.translation.translate('my_message')}`;
    });
  }
}
```

b. In the related Angular template, you can implement `myDynamicMessage` as an ordinary substitutable string, for example:

```
<div>
  <textarea class="response" placeholder="Response" i18n-
placeholder="@@myStaticPlaceholder" >{{myDynamicMessage}}</textarea>
</div>
```

9. Create logic to copy the translation files to the `web/assets` directory during the webpack process, for example in the sample application, the following JavaScript in the `copy-webpack-plugin` file copies the files:

```
 var config = {
 'entry': [
   path.resolve(__dirname, './src/plugin.ts')
   ],
 'output': {
   'path': path.resolve(__dirname, '../web'),
   'filename': 'main.js',
   },
'plugins': [
   new CopyWebpackPlugin([
     {
       from: path.resolve(__dirname, './src/assets'),
       to: path.resolve('../web/assets')
     }
   ])
 ]
};
```

Note: Do not edit files in the `web/assets/i18n` directory. They are overwritten by each build.

Internationalizing React applications

To internationalize Zowe applications using the React framework, take the following steps:

Note: These examples use the recommended react-i18next library, which does not differentiate between dynamic and static content, and unlike the Angular steps above does not require a separate build process.

1. To install the React library, run the following command:

```
npm install --save-dev react-i18next
```

2. In the directory that contains your `index.js` file, create an `i18n.js` file and add the translated content, for example:

```
import i18n from "i18next";
import { initReactI18next } from "react-i18next";

// the translations
// (tip move them in a JSON file and import them)
const resources = {
```

```
  en: {
    translation: {
      "Welcome to React": "Welcome to React and react-i18next"
    }
  }
};

i18n
  .use(initReactI18next) // passes i18n down to react-i18next
  .init({
    resources,
    lng: "en",

    keySeparator: false, // we do not use keys in form messages.welcome

    interpolation: {
      escapeValue: false // react already safes from xss
    }
  });

export default i18n;
```

3. Import the `i18n` file from the previous step into `index.js` file so that you can use it elsewhere, for example:

```
import React, { Component } from "react";
import ReactDOM from "react-dom";
import './i18n';
import App from './App';

// append app to dom
ReactDOM.render(
  <App />,
  document.getElementById("root")
);
```

4. To internationalize a component, include the `useTranslation` hook and reference it to substitute translation keys with their translated values. For example:

```
import React from 'react';

 // the hook
import { useTranslation } from 'react-i18next';

function MyComponent () {
   const { t, i18n } = useTranslation(); // use
   return <h1>{t('Welcome to React')}</h1>
}
```

Internationalizing application desktop titles

To display the translated application name and description in the Desktop, take the following steps:

1. For each language, create a `pluginDefinition.i18n.<lang_code>.json` file. For example, for German create a `pluginDefinition.i18n.de.json` file.
2. Place the `.json` files in the `web/assets/i18n` directory.
3. Translate the `pluginShortNameKey` and `descriptionKey` values in the application's `pluginDefinition.json` file. For example, for the file below you would translate the values `"sampleangular"` and `"sampleangulardescription"`:

```
{
  "identifier": "org.zowe.zlux.sample.angular",
  "apiVersion": "1.0.0",
```

```
  "pluginVersion": "1.1.0",
  "pluginType": "application",
  "webContent": {
    "framework": "angular2",
    "launchDefinition": {
      "pluginShortNameKey": "sampleangular",
      "pluginShortNameDefault": "Angular Sample",
      "imageSrc": "assets/icon.png"
    },
 "descriptionKey": "sampleangulardescription",
 "descriptionDefault": "Sample App Showcasing Angular Adapter",
```

4. Add the translated values to the translation file. For example, the German translation file example, `pluginDefinition.i18n.de.json`, would look like this:

```
{
  "sampleangular": "Beispiel Angular",
  "sampleangulardescription": "Beispiel Angular Anwendung"
}
```

5. Create logic to copy the translation files to the `web/assets` directory during the webpack process. For example, in the Sample Angular Application the following JavaScript in the `webClient/webpack.config.js` file copies files to the `web/assets` directory:

```
var config = {
  'entry': [
    path.resolve(__dirname, './src/plugin.ts')
  ],
  'output': {
    'path': path.resolve(__dirname, '../web'),
    'filename': 'main.js',
  },
  'plugins': [
    new CopyWebpackPlugin([
      {
        from: path.resolve(__dirname, './src/assets'),
        to: path.resolve('../web/assets')
      }
    ])
  ]
};
```

Zowe Desktop and window management

The Zowe™ Desktop is a web component of Zowe, which is an implementation of `MVDWindowManagement`, the interface that is used to create a window manager.

The code for this software is in the `zlux-app-manager` repository.

The interface for building an alternative window manager is in the `zlux-platform` repository.

Window Management acts upon Windows, which are visualizations of an instance of an application plug-in. Application plug-ins are plug-ins of the type "application", and therefore the Zowe Desktop operates around a collection of plug-ins.

Note: Other objects and frameworks that can be utilized by application plug-ins, but not related to window management, such as application-to-application communication, Logging, URI lookup, and Auth are not described here.

Loading and presenting application plug-ins

Upon loading the Zowe Desktop, a GET call is made to `/plugins?type=application`. The GET call returns a JSON list of all application plug-ins that are on the server, which can be accessed by the user. Application plug-ins

can be composed of dataservices, web content, or both. Application plug-ins that have web content are presented in the Zowe Desktop UI.

The Zowe Desktop has a taskbar at the bottom of the page, where it displays each application plug-in as an icon with a description. The icon that is used, and the description that is presented are based on the application plug-in's `PluginDefinition`'s `webContent` attributes.

Plug-in management

Application plug-ins can gain insight into the environment in which they were spawned through the Plugin Manager. Use the Plugin Manager to determine whether a plug-in is present before you act upon the existence of that plug-in. When the Zowe Desktop is running, you can access the Plugin Manager through `ZoweZLUX.PluginManager`

The following are the functions you can use on the Plugin Manager:

- getPlugin(pluginID: string)
  - Accepts a string of a unique plug-in ID, and returns the Plugin Definition Object (DesktopPluginDefinition) that is associated with it, if found.

Application management

Application plug-ins within a Window Manager are created and acted upon in part by an Application Manager. The Application Manager can facilitate communication between application plug-ins, but formal application-to-application communication should be performed by calls to the Dispatcher. The Application Manager is not normally directly accessible by application plug-ins, instead used by the Window Manager.

The following are functions of an Application Manager:

Function	Description
`spawnApplication(plugin: DesktopPluginDefinition, launchMetadata: any): Promise<MVDHosting.InstanceId>;`	Opens an application instance into the Window Manager, with or without context on what actions it should perform after creation.
`killApplication(plugin:ZLUX.Plugin, appId:MVDHosting.InstanceId): void;`	Removes an application instance from the Window Manager.
`showApplicationWindow(plugin: DesktopPluginDefinitionImpl): void;`	Makes an open application instance visible within the Window Manager.
`isApplicationRunning(plugin: DesktopPluginDefinitionImpl): boolean;`	Determines if any instances of the application are open in the Window Manager.

Windows and Viewports

When a user clicks an application plug-in's icon on the taskbar, an instance of the application plug-in is started and presented within a Viewport, which is encapsulated in a Window within the Zowe Desktop. Every instance of an application plug-in's web content within Zowe is given context and can listen on events about the Viewport and Window it exists within, regardless of whether the Window Manager implementation utilizes these constructs visually. It is possible to create a Window Manager that only displays one application plug-in at a time, or to have a drawer-and-panel UI rather than a true windowed UI.

When the Window is created, the application plug-in's web content is encapsulated dependent upon its framework type. The following are valid framework types:

- *"angular2"*: The web content is written in Angular, and packaged with Webpack. Application plug-in framework objects are given through @injectables and imports.
- *"iframe"*: The web content can be written using any framework, but is included through an iframe tag. Application plug-ins within an iframe can access framework objects through *parent.RocketMVD* and callbacks.

In the case of the Zowe Desktop, this framework-specific wrapping is handled by the Plugin Manager.

Viewport Manager

Viewports encapsulate an instance of an application plug-in's web content, but otherwise do not add to the UI (they do not present Chrome as a Window does). Each instance of an application plug-in is associated with a viewport, and operations to act upon a particular application plug-in instance should be done by specifying a viewport for an application plug-in, to differentiate which instance is the target of an action. Actions performed against viewports should be performed through the Viewport Manager.

The following are functions of the Viewport Manager:

Function	Description
`createViewport(providers: ResolvedReflectiveProvider[]): MVDHosting.ViewportId;`	Creates a viewport into which an application plug-in's webcontent can be embedded.
`registerViewport(viewportId: MVDHosting.ViewportId, instanceId: MVDHosting.InstanceId): void;`	Registers a previously created viewport to an application plug-in instance.
`destroyViewport(viewportId: MVDHosting.ViewportId): void;`	Removes a viewport from the Window Manager.
`getApplicationInstanceId(viewportId: MVDHosting.ViewportId): MVDHosting.InstanceId | null;`	Returns the ID of an application plug-in's instance from within a viewport within the Window Manager.

Injection Manager

When you create Angular application plug-ins, they can use injectables to be informed of when an action occurs. iframe application plug-ins indirectly benefit from some of these hooks due to the wrapper acting upon them, but Angular application plug-ins have direct access.

The following topics describe injectables that application plug-ins can use.

Plug-in definition

```
@Inject(Angular2InjectionTokens.PLUGIN_DEFINITION) private pluginDefinition:
 ZLUX.ContainerPluginDefinition
```

Provides the plug-in definition that is associated with this application plug-in. This injectable can be used to gain context about the application plug-in. It can also be used by the application plug-in with other application plug-in framework objects to perform a contextual action.

Logger

```
@Inject(Angular2InjectionTokens.LOGGER) private logger: ZLUX.ComponentLogger
```

Provides a logger that is named after the application plug-in's plugin definition ID.

Launch Metadata

```
@Inject(Angular2InjectionTokens.LAUNCH_METADATA) private launchMetadata: any
```

If present, this variable requests the application plug-in instance to initialize with some context, rather than the default view.

Viewport Events

```
@Inject(Angular2InjectionTokens.VIEWPORT_EVENTS) private viewportEvents:
 Angular2PluginViewportEvents
```

Presents hooks that can be subscribed to for event listening. Events include:

`resized: Subject<{width: number, height: number}>`

Fires when the viewport's size has changed.

Window Events

```
@Inject(Angular2InjectionTokens.WINDOW_ACTIONS) private windowActions:
 Angular2PluginWindowActions
```

Presents hooks that can be subscribed to for event listening. The events include:

Event	Description
`maximized: Subject<void>`	Fires when the Window is maximized.
`minimized: Subject<void>`	Fires when the Window is minimized.
`restored: Subject<void>`	Fires when the Window is restored from a minimized state.
`moved: Subject<{top: number, left: number}>`	Fires when the Window is moved.
`resized: Subject<{width: number, height: number}>`	Fires when the Window is resized.
`titleChanged: Subject<string>`	Fires when the Window's title changes.

Window Actions

```
@Inject(Angular2InjectionTokens.WINDOW_ACTIONS) private windowActions:
 Angular2PluginWindowActions
```

An application plug-in can request actions to be performed on the Window through the following:

Item	Description
`close(): void`	Closes the Window of the application plug-in instance.
`maximize(): void`	Maximizes the Window of the application plug-in instance.
`minimize(): void`	Minimizes the Window of the application plug-in instance.
`restore(): void`	Restores the Window of the application plug-in instance from a minimized state.
`setTitle(title: string):void`	Sets the title of the Window.
`setPosition(pos: {top: number, left: number, width: number, height: number}): void`	Sets the position of the Window on the page and the size of the window.
`spawnContextMenu(xPos: number, yPos: number, items: ContextMenuItem[]): void`	Opens a context menu on the application plug-in instance, which uses the Context Menu framework.
`registerCloseHandler(handler: () => Promise<void>): void`	Registers a handler, which is called when the Window and application plug-in instance are closed.

Configuration Dataservice

The Configuration Dataservice is an essential component of the Zowe™ Application Framework, which acts as a JSON resource storage service, and is accessible externally by REST API and internally to the server by dataservices.

The Configuration Dataservice allows for saving preferences of applications, management of defaults and privileges within a Zowe ecosystem, and bootstrapping configuration of the server's dataservices.

The fundamental element of extensibility of the Zowe Application Framework is a *plug-in*. The Configuration Dataservice works with data for plug-ins. Every resource that is stored in the Configuration Service is stored for a particular plug-in, and valid resources to be accessed are determined by the definition of each plug-in in how it uses the Configuration Dataservice.

The behavior of the Configuration Dataservice is dependent upon the Resource structure for a plug-in. Each plug-in lists the valid resources, and the administrators can set permissions for the users who can view or modify these resources.

Resource Scope

Data is stored within the Configuration Dataservice according to the selected *Scope*. The intent of *Scope* within the Dataservice is to facilitate company-wide administration and privilege management of Zowe data.

When a user requests a resource, the resource that is retrieved is an override or an aggregation of the broader scopes that encompass the *Scope* from which they are viewing the data.

When a user stores a resource, the resource is stored within a *Scope* but only if the user has access privilege to update within that *Scope*.

Scope is one of the following:

Product

Configuration defaults that come with the product. Cannot be modified.

Site

Data that can be used between multiple instances of the Zowe Application Server.

Instance

Data within an individual Zowe Application Server.

Group

Data that is shared between multiple users in a group.(Pending)

User

Data for an individual user.(Pending)

Note: While Authorization tuning can allow for settings such as GET from Instance to work without login, *User* and *Group* scope queries will be rejected if not logged in due to the requirement to pull resources from a specific user. Because of this, *User* and *Group* scopes will not be functional until the Security Framework is merged into the mainline.

Where *Product* is the broadest scope and *User* is the narrowest scope.

When you specify *Scope User*, the service manages configuration for your particular username, using the authentication of the session. This way, the *User* scope is always mapped to your current username.

Consider a case where a user wants to access preferences for their text editor. One way they could do this is to use the REST API to retrieve the settings resource from the *Instance* scope.

The *Instance* scope might contain editor defaults set by the administrator. But, if there are no defaults in *Instance*, then the data in *Group* and *User* would be checked.

Therefore, the data the user receives would be no broader than what is stored in the *Instance* scope, but might have only been the settings they saved within their own *User* scope (if the broader scopes do not have data for the resource).

Later, the user might want to save changes, and they try to save them in the *Instance* scope. Most likely, this action will be rejected because of the preferences set by the administrator to disallow changes to the *Instance* scope by ordinary users.

REST API

When you reach the Configuration Service through a REST API, HTTP methods are used to perform the desired operation.

The HTTP URL scheme for the configuration dataservice is:

```
<Server>/plugins/com.rs.configjs/services/data/<plugin ID>/<Scope>/<resource>/
<optional subresources>?<query>
```

Where the resources are one or more levels deep, using as many layers of subresources as needed.

Think of a resource as a collection of elements, or a directory. To access a single element, you must use the query parameter "name="

REST query parameters

Name (string)

Get or put a single element rather than a collection.

Recursive (boolean)

When performing a DELETE, specifies whether to delete subresources too.

Listing (boolean)

When performing a GET against a resource with content subresources, `listing=true` will provide the names of the subresources rather than both the names and contents.

REST HTTP methods

Below is an explanation of each type of REST call.

Each API call includes an example request and response against a hypothetical application called the "code editor".

GET

GET `/plugins/com.rs.configjs/services/data/<plugin>/<scope>/<resource>?name=<element>`

- This returns JSON with the attribute "content" being a JSON resource that is the entire configuration that was requested. For example:

```
/plugins/com.rs.configjs/services/data/org.openmainframe.zowe.codeeditor/user/
sessions/default?name=tabs
```

The parts of the URL are:

- Plugin: org.openmainframe.zowe.codeeditor
- Scope: user
- Resource: sessions
- Subresource: default
- Element = tabs

The response body is a JSON config:

```
{
 "_objectType" : "com.rs.config.resource",
 "_metadataVersion" : "1.1",
 "resource" : "org.openmainframe.zowe.codeeditor/USER/sessions/default",
 "contents" : {
  "_metadataVersion" : "1.1",
  "_objectType" : "org.openmainframe.zowe.codeeditor.sessions.tabs",
  "tabs" : [{
    "title" : "TSSPG.REXX.EXEC(ARCTEST2)",
    "filePath" : "TSSPG.REXX.EXEC(ARCTEST2)",
    "isDataset" : true
```

```
    }, {
     "title" : ".profile",
     "filePath" : "/u/tsspg/.profile"
    }
   ]
  }
}
```

GET `/plugins/com.rs.configjs/services/data/<plugin>/<scope>/<resource>`

This returns JSON with the attribute `content` being a JSON object that has each attribute being another JSON object, which is a single configuration element.

GET `/plugins/com.rs.configjs/services/data/<plugin>/<scope>/<resource>`

(When subresources exist.)

This returns a listing of subresources that can, in turn, be queried.

PUT

PUT `/plugins/com.rs.configjs/services/data/<plugin>/<scope>/<resource>?name=<element>`

Stores a single element (must be a JSON object {...}) within the requested scope, ignoring aggregation policies, depending on the user privilege. For example:

`/plugins/com.rs.configjs/services/data/org.openmainframe.zowe.codeeditor/user/sessions/default?name=tabs`

Body:

```
{
  "_metadataVersion" : "1.1",
  "_objectType" : "org.openmainframe.zowe.codeeditor.sessions.tabs",
  "tabs" : [{
      "title" : ".profile",
      "filePath" : "/u/tsspg/.profile"
    }, {
      "title" : "TSSPG.REXX.EXEC(ARCTEST2)",
      "filePath" : "TSSPG.REXX.EXEC(ARCTEST2)",
      "isDataset" : true
    }, {
      "title" : ".emacs",
      "filePath" : "/u/tsspg/.emacs"
    }
  ]
}
```

Response:

```
{
  "_objectType" : "com.rs.config.resourceUpdate",
  "_metadataVersion" : "1.1",
  "resource" : "org.openmainframe.zowe.codeeditor/USER/sessions/default",
  "result" : "Replaced item."
}
```

DELETE

DELETE `/plugins/com.rs.configjs/services/data/<plugin>/<scope>/<resource>?recursive=true`

Deletes all files in all leaf resources below the resource specified.

DELETE `/plugins/com.rs.configjs/services/data/<plugin>/<scope>/<resource>?name=<element>`

Deletes a single file in a leaf resource.

DELETE `/plugins/com.rs.configjs/services/data/<plugin>/<scope>/<resource>`

- Deletes all files in a leaf resource.
- Does not delete the directory on disk.

Administrative access and group

By means not discussed here, but instead handled by the server's authentication and authorization code, a user might be privileged to access or modify items that they do not own.

In the simplest case, it might mean that the user is able to do a PUT, POST, or DELETE to a level above *User*, such as *Instance*.

The more interesting case is in accessing another user's contents. In this case, the shape of the URL is different. Compare the following two commands:

GET `/plugins/com.rs.configjs/services/data/<plugin>/user/<resource>`

Gets the content for the current user.

GET `/plugins/com.rs.configjs/services/data/<plugin>/users/<username>/<resource>`

Gets the content for a specific user if authorized.

This is the same structure that is used for the *Group* scope. When requesting content from the *Group* scope, the user is checked to see if they are authorized to make the request for the specific group. For example:

GET `/plugins/com.rs.configjs/services/data/<plugin>/group/<groupname>/<resource>`

Gets the content for the given group, if the user is authorized.

Application API

Retrieves and stores configuration information from specific scopes.

Note: This API should only be used for configuration administration user interfaces.

```
ZLUX.UriBroker.pluginConfigForScopeUri(pluginDefinition: ZLUX.Plugin, scope: string, resourcePath:string, resourceName:string): string;
```

A shortcut for the preceding method, and the preferred method when you are retrieving configuration information, is simply to "consume" it. It "asks" for configurations using the *User* scope, and allows the configuration service to decide which configuration information to retrieve and how to aggregate it. (See below on how the configuration service evaluates what to return for this type of request).

```
ZLUX.UriBroker.pluginConfigUri(pluginDefinition: ZLUX.Plugin, resourcePath:string, resourceName:string): string;
```

Internal and bootstrapping

Some dataservices within plug-ins can take configuration that affects their behavior. This configuration is stored within the Configuration Dataservice structure, but it is not accessible through the REST API.

Within the instance configuration directory of a zLUX installation, each plugin may optionally have an `_internal` directory. An example of such a path would be:

```
~/.zowe/workspace/app-server/ZLUX/pluginStorage/<pluginName>/_internal
```

Within each `_internal` directory, the following directories might exist:

- `services/<servicename>`: Configuration resources for the specific service.
- `plugin`: Configuration resources that are visible to all services in the plug-in.

The JSON contents within these directories are provided as Objects to dataservices through the dataservice context Object.

Plug-in definition

Because the Configuration Dataservices stores data on a per-plug-in basis, each plug-in must define their resource structure to make use of the Configuration Dataservice. The resource structure definition is included in the plug-in's `pluginDefinition.json` file.

For each resource and subresource, you can define an `aggregationPolicy` to control how the data of a broader scope alters the resource data that is returned to a user when requesting a resource from a narrower Scope.

For example:

```
  "configurationData": { //is a direct attribute of the pluginDefinition
 JSON
    "resources": { //always required
      "preferences": {
        "locationType": "relative", //this is the only option for now, but
 later absolute paths may be accepted
        "aggregationPolicy": "override" //override and none for now, but
 more in the future
      },
      "sessions": { //the name at this level represents the name
 used within a URL, such as /plugins/com.rs.configjs/services/data/
org.openmainframe.zowe.codeeditor/user/sessions
        "aggregationPolicy": "none",
        "subResources": {
          "sessionName": {
            "variable": true, //if variable=true is present, the resource
 must be the only one in that group but the name of the resource is
 substituted for the name given in the REST request, so it represents more
 than one
            "aggregationPolicy": "none"
          }
        }
      }
    }
  }
```

Aggregation policies

Aggregation policies determine how the Configuration Dataservice aggregates JSON objects from different Scopes together when a user requests a resource. If the user requests a resource from the *User* scope, the data from the User scope might replace or be merged with the data from a broader scope such as *Instance*, to make a combined resource object that is returned to the user.

Aggregation policies are defined by a plug-in developer in the plug-in's definition for the Configuration Service, as the attribute `aggregationPolicy` within a resource.

The following policies are currently implemented:

- **NONE**: If the Configuration Dataservice is called for *Scope User*, only user-saved settings are sent, unless there are no user-saved settings for the query, in which case the dataservice attempts to send data that is found at a broader scope.
- **OVERRIDE**: The Configuration Dataservice obtains data for the resource that is requested at the broadest level found, and joins the resource's properties from narrower scopes, overriding broader attributes with narrower ones, when found.

URI Broker

The URI Broker is an object in the application plug-in web framework, which facilitates calls to the Zowe™ Application Server by constructing URIs that use the context from the calling application plug-in.

Accessing the URI Broker

The URI Broker is accessible independent of other frameworks involved such as Angular, and is also accessible through iframe. This is because it is attached to a global when within the Zowe Desktop. For more information, see Zowe Desktop and window management on page 258. Access the URI Broker through one of two locations:

Natively:

```
window.ZoweZLUX.uriBroker
```

In an iframe:

```
window.parent.ZoweZLUX.uriBroker
```

Functions

The URI Broker builds the following categories of URIs depending upon what the application plug-in is designed to call.

Accessing an application plug-in's dataservices

Dataservices can be based on HTTP (REST) or Websocket. For more information, see Dataservices on page 246.

HTTP Dataservice URI

```
pluginRESTUri(plugin:ZLUX.Plugin, serviceName: string, relativePath:string): string
```

Returns: A URI for making an HTTP service request.

Websocket Dataservice URI

```
pluginWSUri(plugin: ZLUX.Plugin, serviceName:string, relativePath:string): string
```

Returns: A URI for making a Websocket connection to the service.

Accessing application plug-in's configuration resources

Defaults and user storage might exist for an application plug-in such that they can be retrieved through the Configuration Dataservice.

There are different scopes and actions to take with this service, and therefore there are a few URIs that can be built:

Standard configuration access

```
pluginConfigUri(pluginDefinition: ZLUX.Plugin, resourcePath:string, resourceName?:string): string
```

Returns: A URI for accessing the requested resource under the user's storage.

Scoped configuration access

```
pluginConfigForScopeUri(pluginDefinition: ZLUX.Plugin, scope: string, resourcePath:string, resourceName?:string): string
```

Returns: A URI for accessing a specific scope for a given resource.

Accessing static content

Content under an application plug-in's `web` directory is static content accessible by a browser. This can be accessed through:

```
pluginResourceUri(pluginDefinition: ZLUX.Plugin, relativePath: string): string
```

Returns: A URI for getting static content.

For more information about the `web` directory, see Application plug-in filesystem structure on page 239.

Accessing the application plug-in's root

Static content and services are accessed off of the root URI of an application plug-in. If there are other points that you must access on that application plug-in, you can get the root:

```
pluginRootUri(pluginDefinition: ZLUX.Plugin): string
```

Returns: A URI to the root of the application plug-in.

Server queries

A client can find different information about a server's configuration or the configuration as seen by the current user by accessing specific APIs.

Accessing a list of plug-ins

```
pluginListUri(pluginType: ZLUX.PluginType): string
```

Returns: A URI, which when accessed returns the list of existing plug-ins on the server by type, such as "Application" or "all".

Application-to-application communication

Zowe™ application plug-ins can opt-in to various application framework abilities, such as the ability to have a Logger, use of a URI builder utility, and more. One ability that is unique to a Zowe environment with multiple application plug-ins is the ability for one application plug-in to communicate with another. The application framework provides constructs that facilitate this ability. The constructs are: the Dispatcher, Actions, Recognizers, Registry, and the features that utilize them such as the framework's Context menu.

1. Why use application-to-application communication? on page 268
2. Actions on page 268
3. Recognizers on page 270
4. Dispatcher on page 272

Why use application-to-application communication?

When working with a computer, people often use multiple applications to accomplish a task, for example checking a dashboard before using a detailed program or checking email before opening a bank statement in a browser. In many environments, the relationship between one program and another is not well defined (you might open one program to learn of a situation, which you solve by opening another program and typing or pasting in content). Or perhaps a hyperlink is provided or an attachment, which opens a program using a lookup table of which the program is the default for handling a certain file extension. The application framework attempts to solve this problem by creating structured messages that can be sent from one application plug-in to another. An application plug-in has a context of the information that it contains. You can use this context to invoke an action on another application plug-in that is better suited to handle some of the information discovered in the first application plug-in. Well-structured messages facilitate knowing what application plug-in is "right" to handle a situation, and explain in detail what that application plug-in should do. This way, rather than finding out that the attachment with the extension ".dat" was not meant for a text editor, but instead for an email client, one application plug-in might instead be able to invoke an action on an application plug-in, which can handle opening of an email for the purpose of forwarding to others (a more specific task than can be explained with filename extensions).

Actions

To manage communication from one application plug-in to another, a specific structure is needed. In the application framework, the unit of application-to-application communication is an Action. The typescript definition of an Action is as follows:

```
export class Action implements ZLUX.Action {
    id: string;           // id of action itself.
```

```
    i18nNameKey: string;  // future proofing for I18N
    defaultName: string;  // default name for display purposes, w/o I18N
    description: string;
    targetMode: ActionTargetMode;
    type: ActionType;   // "launch", "message"
    targetPluginID: string;
    primaryArgument: any;

    constructor(id: string,
                defaultName: string,
                targetMode: ActionTargetMode,
                type: ActionType,
                targetPluginID: string,
                primaryArgument:any) {
       this.id = id;
       this.defaultName = defaultName;
       // proper name for ID/type
       this.targetPluginID = targetPluginID;
       this.targetMode = targetMode;
       this.type = type;
       this.primaryArgument = primaryArgument;
    }

    getDefaultName():string {
      return this.defaultName;
    }
}
```

An Action has a specific structure of data that is passed, to be filled in with the context at runtime, and a specific target to receive the data. The Action is dispatched to the target in one of several modes, for example: to target a specific instance of an application plug-in, an instance, or to create a new instance. The Action can be less detailed than a message. It can be a request to minimize, maximize, close, launch, and more. Finally, all of this information is related to a unique ID and localization string such that it can be managed by the framework.

Action target modes

When you request an Action on an application plug-in, the behavior is dependent on the instance of the application plug-in you are targeting. You can instruct the framework how to target the application plug-in with a target mode from the `ActionTargetMode` `enum`:

```
export enum ActionTargetMode {
  PluginCreate,                // require pluginType
  PluginFindUniqueOrCreate,    // required AppInstance/ID
  PluginFindAnyOrCreate,       // plugin type
  //TODO PluginFindAnyOrFail
  System,                      // something that is always present
}
```

Action types

The application framework performs different operations on application plug-ins depending on the type of an Action. The behavior can be quite different, from simple messaging to requesting that an application plug-in be minimized. The types are defined by an `enum`:

```
export enum ActionType {       // not all actions are meaningful for all
 target modes
  Launch,                      // essentially do nothing after target mode
  Focus,                       // bring to fore, but nothing else
  Route,                       // sub-navigate or "route" in target
  Message,                     // "onMessage" style event to plugin
  Method,                      // Method call on instance, more strongly
 typed
  Minimize,
```

```
  Maximize,
  Close,                        // may need to call a "close handler"
}
```

Loading actions

Actions can be created dynamically at runtime, or saved and loaded by the system at login.

Dynamically

You can create Actions by calling the following Dispatcher method: `makeAction(id: string, defaultName: string, targetMode: ActionTargetMode, type: ActionType, targetPluginID: string, primaryArgument: any):Action`

Saved on system

Actions can be stored in JSON files that are loaded at login. The JSON structure is as follows:

```
{
  "actions": [
    {
      "id":"org.zowe.explorer.openmember",
      "defaultName":"Edit PDS in MVS Explorer",
      "type":"Launch",
      "targetMode":"PluginCreate",
      "targetId":"org.zowe.explorer",
      "arg": {
        "type": "edit_pds",
        "pds": {
          "op": "deref",
          "source": "event",
          "path": [
            "full_path"
          ]
        }
      }
    }
  ]
}
```

Recognizers

Actions are meant to be invoked when certain conditions are met. For example, you do not need to open a messaging window if you have no one to message. Recognizers are objects within the application framework that use the context that the application plug-in provides to determine if there is a condition for which it makes sense to execute an Action. Each recognizer has statements about what condition to recognize, and upon that statement being met, which Action can be executed at that time. The invocation of the Action is not handled by the Recognizer; it simply detects that an Action can be taken.

Recognition clauses

Recognizers associate a clause of recognition with an action, as you can see from the following class:

```
export class RecognitionRule {
  predicate:RecognitionClause;
  actionID:string;

  constructor(predicate:RecognitionClause, actionID:string){
    this.predicate = predicate;
    this.actionID = actionID;
  }
}
```

A clause, in turn, is associated with an operation, and the subclauses upon which the operation acts. The following operations are supported:

```
export enum RecognitionOp {
  AND,
  OR,
  NOT,
  PROPERTY_EQ,
  SOURCE_PLUGIN_TYPE,      // syntactic sugar
  MIME_TYPE,        // ditto
}
```

Loading Recognizers at runtime

You can add a Recognizer to the application plug-in environment in one of two ways: by loading from Recognizers saved on the system, or by adding them dynamically.

Dynamically

You can call the Dispatcher method, `addRecognizer(predicate:RecognitionClause, actionID:string):void`

Saved on system

Recognizers can be stored in JSON files that are loaded at login. The JSON structure is as follows:

```
{
  "recognizers": [
    {
      "id":"<actionID>",
      "clause": {
        <clause>
      }
    }
  ]
}
```

clause can take on one of two shapes:

```
"prop": ["<keyString>", <"valueString">]
```

Or,

```
"op": "<op enum as string>",
"args": [
  {<clause>}
]
```

Where this one can again, have subclauses.

Recognizer example

Recognizers can be as simple or complex as you write them to be, but here is an example to illustrate the mechanism:

```
{
  "recognizers":[
    {
      "id":"org.zowe.explorer.openmember",
      "clause": {
        "op": "AND",
        "args": [
         {"prop":["sourcePluginID","org.zowe.terminal.tn3270"]},{"prop":
["screenID","ISRUDSM"]}
```

```
        ]
      }
    }
  ]
}
```

In this case, the Recognizer detects whether it is possible to run the `org.zowe.explorer.openmember` Action when the TN3270 Terminal application plug-in is on the screen ISRUDSM (an ISPF panel for browsing PDS members).

Dispatcher

The dispatcher is a core component of the application framework that is accessible through the Global `ZLUX` Object at runtime. The Dispatcher interprets Recognizers and Actions that are added to it at runtime. You can register Actions and Recognizers on it, and later, invoke an Action through it. The dispatcher handles how the Action's effects should be carried out, acting in combination with the Window Manager and application plug-ins to provide a channel of communication.

Registry

The Registry is a core component of the application framework, which is accessible through the Global `ZLUX` Object at runtime. It contains information about which application plug-ins are present in the environment, and the abilities of each application plug-in. This is important to application-to-application communication, because a target might not be a specific application plug-in, but rather an application plug-in of a specific category, or with a specific featureset, capable of responding to the type of Action requested.

Pulling it all together in an example

The standard way to make use of application-to-application communication is by having Actions and Recognizers that are saved on the system. Actions and Recognizers are loaded at login, and then later, through a form of automation or by a user action, Recognizers can be polled to determine if there is an Action that can be executed. All of this is handled by the Dispatcher, but the description of the behavior lies in the Action and Recognizer that are used. In the Action and Recognizer descriptions above, there are two JSON definitions: One is a Recognizer that recognizes when the Terminal application plug-in is in a certain state, and another is an Action that instructs the MVS Explorer to load a PDS member for editing. When you put the two together, a practical application is that you can launch the MVS Explorer to edit a PDS member that you have selected within the Terminal application plug-in.

Configuring IFrame communication

The Zowe Application Framework provides the following shared resource functions through a ZoweZLUX object: `pluginManager`, `uriBroker`, `dispatcher`, `logger`, `registry`, `notificationManager`, and `globalization`

Like REACT and Angular apps, IFrame apps can use the ZoweZLUX object to communicate with the framework and other apps. To enable communication in an IFrame app, you must add the following javascript to your app, for example in your `index.html` file:

```
<script>
 if(exports){
  var ZoweZLUX_tempExports = exports;
 }
 var exports = {"__esModule": true};

</script>
<script type="text/javascript" src="../../../../../lib/
org.zowe.zlux.logger/0.9.0/logger.js"></script>
<script type="text/javascript" src="../../../org.zowe.zlux.bootstrap/web/
iframe-adapter.js"></script>
```

`logger.js` is the javascript version of `logger.ts` and is capable of the same functions, including access to the `Logger` and `ComponentLogger` classes. The `Logger` class determines the behavior of all the `ComponentLoggers` created from it. `ComponentLoggers` are what the user implements to perform logging.

`Iframe-adapter.js` is designed to mimic the ZoweZLUX object that is available to apps within the virtual-desktop, and serves as the middle-man for communication between IFrame apps and the Zowe desktop.

You can see an implementation of this functionality in the sample IFrame app.

The version of ZoweZLUX adapted for IFrame apps is not complete and only implements the functions needed to allow the Sample IFrame App to function. The `notificationManager`, `logger`, `globalization`, `dispatcher`, `windowActions`, `windowEvents`, and `viewportEvents` are fully implemented. The `pluginManager` and `uriBroker` are only partially implemented. The `registry` is not implemented.

Unlike REACT and Angular apps, in IFrame apps the ZoweZLUX and initialization objects communicate with Zowe using the browser's onmessage and postmessage APIs. That means that communication operations are asynchronous, and you must account for this in your app, for example by using Promise objects and `await` or `then` functions.

Error reporting UI

The `zLUX Widgets` repository contains shared widget-like components of the Zowe™ Desktop, including Button, Checkbox, Paginator, various pop-ups, and others. To maintain consistency in desktop styling across all applications, use, reuse, and customize existing widgets to suit the purpose of the application's function and look.

Ideally, a program should have little to no logic errors. Once in a while a few occur, but more commonly an error occurs from misconfigured user settings. A user might request an action or command that requires certain prerequisites, for example: a proper ZSS-Server configuration. If the program or method fails, the program should notify the user through the UI about the error and how to fix it. For the purposes of this discussion, we will use the Workflow application plug-in in the `zlux-workflow` repository.

ZluxPopupManagerService

The `ZluxPopupManagerService` is a standard popup widget that can, through its `reportError()` method, be used to display errors with attributes that specify the title or error code, severity, text, whether it should block the user from proceeding, whether it should output to the logger, and other options you want to add to the error dialog. `ZluxPopupManagerService` uses both `ZluxErrorSeverity` and `ErrorReportStruct`.

```
`export declare class ZluxPopupManagerService {`

    eventsSubject: any;
    listeners: any;
    events: any;
    logger: any;
    constructor();
    setLogger(logger: any): void;
    on(name: any, listener: any): void;
    broadcast(name: any, ...args: any[]): void;
    processButtons(buttons: any[]): any[];
    block(): void;
    unblock(): void;
    getLoggerSeverity(severity: ZluxErrorSeverity): any;
    reportError(severity: ZluxErrorSeverity, title: string, text: string,
 options?: any): Rx.Observable<any>;
`}`
```

ZluxErrorSeverity

`ZluxErrorSeverity` classifies the type of report. Under the popup-manager, there are the following types: error, warning, and information. Each type has its own visual style. To accurately indicate the type of issue to the user, the error or pop-up should be classified accordingly.

```
`export declare enum ZluxErrorSeverity {`
```

```
    ERROR = "error",
    WARNING = "warning",
    INFO = "info",
`}`
```

ErrorReportStruct

`ErrorReportStruct` contains the main interface that brings the specified parameters of `reportError()` together.

```
`export interface ErrorReportStruct {`

    severity: string;
    modal: boolean;
    text: string;
    title: string;
    buttons: string[];
`}`
```

Implementation

Import `ZluxPopupManagerService` and `ZluxErrorSeverity` from widgets. If you are using additional services with your error prompt, import those too (for example, `LoggerService` to print to the logger or `GlobalVeilService` to create a visible semi-transparent gray veil over the program and pause background tasks). Here, widgets is imported from `node_modules\@zlux\` so you must ensure zLUX widgets is used in your `package-lock.json` or `package.json` and you have run `npm install`.

```
import { ZluxPopupManagerService, ZluxErrorSeverity } from '@zlux/widgets';
```

Declaration

Create a member variable within the constructor of the class you want to use it for. For example, in the Workflow application plug-in under `\zlux-workflow\src\app\app\zosmf-server-config.component.ts` is a `ZosmfServerConfigComponent` class with the pop-up manager service variable. To automatically report the error to the console, you must set a logger.

```
`export class ZosmfServerConfigComponent {`

    constructor(
    private popupManager: ZluxPopupManagerService, )
    {  popupManager.setLogger(logger); } //Optional
`}`
```

Usage

Now that you have declared your variable within the scope of your program's class, you are ready to use the method. The following example describes an instance of the `reload()` method in Workflow that catches an error when the program attempts to retrieve a configuration from a `configService` and set it to the program's `this.config`. This method fails when the user has a faulty zss-Server configuration and the error is caught and then sent to the class' `popupManager` variable from the constructor above.

```
`reload(): void {`

    this.globalVeilService.showVeil();
    this.configService
      .getConfig()
      .then(config => (this.config = config))
      .then(_ => setTimeout(() => this.test(), 0))
      .then(_ => this.globalVeilService.hideVeil())
      .catch(err => {
        this.globalVeilService.hideVeil()
```

```
        let errorTitle: string = "Error";
        let errorMessage: string = "Server configuration not found. Please
 check your zss server.";
        const options = {
          blocking: true
        };
          this.popupManager.reportError(ZluxErrorSeverity.ERROR,
 errorTitle.toString()+": "+err.status.toString(), errorMessage
+"\n"+err.toString(), options);
        });
`}`
```

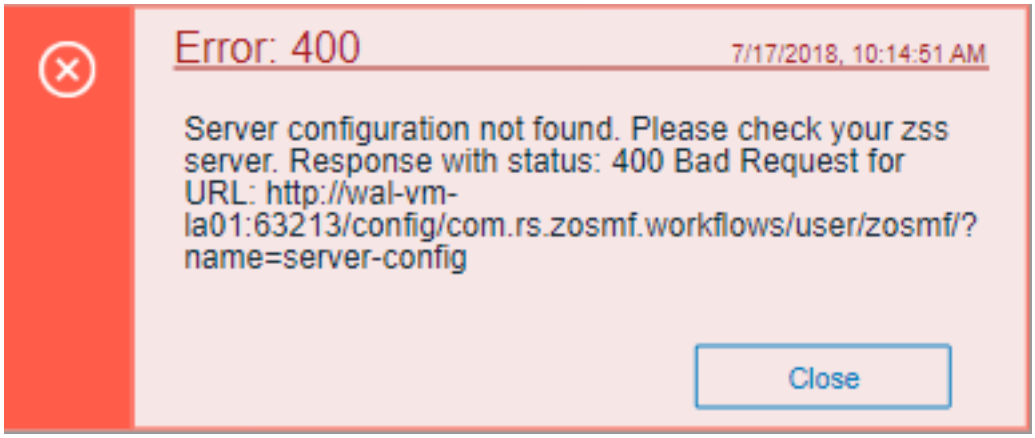
Here, the `errorMessage` clearly describes the error with a small degree of ambiguity as to account for all types of errors that might occur from that method. The specifics of the error are then generated dynamically and are printed with the `err.toString()`, which contains the more specific information that is used to pinpoint the problem. The `this.popupManager.report()` method triggers the error prompt to display. The error severity is set with `ZluxErrorSeverity.ERROR` and the `err.status.toString()` describes the status of the error (often classified by a code, for example: `404`). The optional parameters in `options` specify that this error will block the user from interacting with the application plug-in until the error is closed or it until goes away on its own. `globalVeilService` is optional and is used to create a gray veil on the outside of the program when the error is caused. You must import `globalVeilService` separately (see the `zlux-workflow` repository for more information).

HTML

The final step is to have the recently created error dialog display in the application plug-in. If you do `this.popupManager.report()` without adding the component to your template, the error will not be displayed. Navigate to your component's `.html` file. On the Workflow application plug-in, this file will be in `\zlux-workflow\src\app\app\zosmf-server-config.component.html` and the only item left is to add the popup manager component alongside your other classes.

```
<zlux-popup-manager></zlux-popup-manager>
```

So now when the error is called, the new UI element should resemble the following:



The order in which you place the pop-up manager determines how the error dialog will overlap in your UI. If you want the error dialog to overlap other UI elements, place it at the end of the `.html` file. You can also create custom styling through a CSS template, and add it within the scope of your application plug-in.

Logging utility

The `zlux-shared` repository provides a logging utility for use by dataservices and web content for an application plug-in.

Logging objects

The logging utility is based on the following objects:

- **Component Loggers**: Objects that log messages for an individual component of the environment, such as a REST API for an application plug-in or to log user access.
- **Destinations**: Objects that are called when a component logger requests a message to be logged. Destinations determine how something is logged, for example, to a file or to a console, and what formatting is applied.
- **Logger**: Central logging object, which can spawn component loggers and attach destinations.

Logger IDs

Because Zowe™ application plug-ins have unique identifiers, both dataservices and an application plug-in's web content are provided with a component logger that knows this unique ID such that messages that are logged can be prefixed with the ID. With the association of logging to IDs, you can control verbosity of logs by setting log verbosity by ID.

Accessing logger objects

Logger

The core logger object is attached as a global for low-level access.

App Server

NodeJS uses `global` as its global object, so the logger is attached to: `global.COM_RS_COMMON_LOGGER`

Web

Browsers use `window` as the global object, so the logger is attached to: `window.COM_RS_COMMON_LOGGER`

Component logger

Component loggers are created from the core logger object, but when working with an application plug-in, allow the application plug-in framework to create these loggers for you. An application plug-in's component logger is presented to dataservices or web content as follows.

App Server

See **Router Dataservice Context** in the topic Dataservices on page 246.

Web

(Angular App Instance Injectible). See **Logger** in Zowe Desktop and window management on page 258.

Using log message IDs

To make technical support for your application easier, create IDs for common log messages and use substitution to generate them. When you use IDs, people fielding support calls can identify and solve problems more quickly. IDs are particularly helpful if your application is translated, because it avoids users having to explain problems using language that the tech support person might not understand.

To use log message IDs, take the following steps:

1. Depending on how your application is structured, create message files in the following locations:
   - Web log messages: `\src\assets\i18n\log\messages_{language}.json`
   - App server log messages: `\lib\assets\i18n\log\messages_{language}.json`
2. In the files, create ID-message pairs using the following format:

   ```
   { "id1": "value1", "id2": "value2" [...] }
   ```

   Where "id#" is the message ID and "value#" is the text. For example:

   ```
   { "A001": "Application created.", "A002": "Application deleted." [...] }
   ```

3. Reference the IDs in your code, for example:

```
this.log.info("A0001")
```

Which compiles to:

```
DATE TIME:TIME:TIME.TIME <ZWED:> username INFO (org.zowe.app.name,:) A0001
 - Application created.
```

Or in another supported language, such as Russian:

```
DATE TIME:TIME:TIME.TIME <ZWED:> username INFO (org.zowe.app.name,:) A0001
 - ########## #######.
```

Logger API

The following constants and functions are available on the central logging object.

Attribute	Type	Description	Arguments
`makeComponentLogger`	function	Returns an existing logger of this name, or creates a new component logger if no logger of the specified name exists - Automatically done by the application framework for dataservices and web content	`componentIDString`
`setLogLevelForComponentName`	function	Sets the verbosity of an existing component logger	`componentIDString`, `logLevel`

Component Logger API

The following constants and functions are available to each component logger.

Attribute	Type	Description	Arguments
`SEVERE`	const	Is a const for `logLevel`	
`WARNING`	const	Is a const for `logLevel`	
`INFO`	const	Is a const for `logLevel`	
`FINE`	const	Is a const for `logLevel`	
`FINER`	const	Is a const for `logLevel`	
`FINEST`	const	Is a const for `logLevel`	
`log`	function	Used to write a log, specifying the log level	`logLevel`, `messageString`
`severe`	function	Used to write a SEVERE log.	`messageString`
`warn`	function	Used to write a WARNING log.	`messageString`
`info`	function	Used to write an INFO log.	`messageString`
`debug`	function	Used to write a FINE log.	`messageString`

Attribute	Type	Description	Arguments
`makeSublogger`	function	Creates a new component logger with an ID appended by the string given	`componentNameSuffix`

Log Levels

An enum, `LogLevel`, exists for specifying the verbosity level of a logger. The mapping is:

Level	Number
SEVERE	0
WARNING	1
INFO	2
FINE	3
FINER	4
FINEST	5

Note: The default log level for a logger is **INFO**.

Logging verbosity

Using the component logger API, loggers can dictate at which level of verbosity a log message should be visible. You can configure the server or client to show more or less verbose messages by using the core logger's API objects.

Example: You want to set the verbosity of the org.zowe.foo application plug-in's dataservice, bar to show debugging information.

```
logger.setLogLevelForComponentName('org.zowe.foo.bar',LogLevel.DEBUG)
```

Configuring logging verbosity

The application plug-in framework provides ways to specify what component loggers you would like to set default verbosity for, such that you can easily turn logging on or off.

Server startup logging configuration

The server configuration file allows for specification of default log levels, as a top-level attribute `logLevel`, which takes key-value pairs where the key is a regex pattern for component IDs, and the value is an integer for the log levels.

For example:

```
"logLevel": {
    "com.rs.configjs.data.access": 2,
    //the string given is a regex pattern string, so .* at the end here will
 cover that service and its subloggers.
    "com.rs.myplugin.myservice.*": 4
    //
    // '_' char reserved, and '_' at beginning reserved for server. Just as
 we reserve
    // '_internal' for plugin config data for config service.
    // _unp = universal node proxy core logging
    //"_unp.dsauth": 2
  },
```

For more information about the server configuration file, see Zowe Application Framework configuration on page 108.

Zowe Conformance Program

Introduction

Administered by the Open Mainframe Project, the Zowe™ Conformance Program aims to give users the confidence that when they use a product, app, or distribution that leverages Zowe, they can expect a high level of common functionality, interoperability, and user experience.

Conformance provides Independent Software Vendors (ISVs), System Integrators (SIs), and end users greater confidence that their software will behave as expected. Just like Zowe, the Zowe Conformance Program will continue to evolve and is being developed by committers and contributors in the Zowe community.

As vendors, you are invited to submit conformance testing results for review and approval by the Open Mainframe Project. If your company provides software based on Zowe, you are encouraged to get certified today.

How to participate

To participate in the Zowe Conformance Program, follow the process on the Zowe Conformance Program website. You can also find a list of products that have earned Zowe Conformant status.

Chapter 4

Troubleshooting

Topics:

- Troubleshooting
- Zowe API Mediation Layer
- Zowe Application Framework
- Troubleshooting z/OS Services
- Zowe CLI
- Troubleshooting Zowe through Zowe Open Community

Troubleshooting

To isolate and resolve Zowe™ problems, you can use the troubleshooting and support information in this section.

Topics

- Troubleshooting API ML on page 282
- Troubleshooting Zowe Application Framework on page 301
- Troubleshooting z/OS Services on page 307
- Troubleshooting Zowe CLI on page 308
- Troubleshooting Zowe through Zowe Open Community on page 313

Zowe API Mediation Layer

Troubleshooting API ML

As an API Mediation Layer user, you may encounter problems with how the API ML functions. This article presents known API ML issues and their solutions.

Enable API ML Debug Mode

Use debug mode to activate the following functions:

- Display additional debug messages for API ML
- Enable changing log level for individual code components

Important: We highly recommend that you enable debug mode only when you want to troubleshoot issues. Disable debug mode when you are not troubleshooting. Running in debug mode while operating API ML can adversely affect its performance and create large log files that consume a large volume of disk space.

Follow these steps:

1. Open the file `<Zowe install directory>/components/api-mediation/bin/start.sh`.
2. Find the API Mediation Layer service, for which you want to enable the debug mode: discovery, catalog, or gateway.
3. Find the line that contains the `LOG_LEVEL=` parameter and set the value to `debug`:

   ```
   LOG_LEVEL=debug
   ```

4. Restart Zowe™.

   You have enabled the debug mode.
5. (Optional) Reproduce a bug that causes issues and review debug messages. If you are unable to resolve the issue, create an issue here.
6. Disable the debug mode. Find the `LOG_LEVEL` parameter, and change its current value to the default `LOG_LEVEL=` one:

   ```
   LOG_LEVEL=
   ```

7. Restart Zowe.

   You have disabled the debug mode.

Change the Log Level of Individual Code Components

You can change the log level of a particular code component of the API ML internal service at run time.

Follow these steps:

1. Enable API ML Debug Mode as described in Enable API ML Debug Mode. This activates the application/loggers endpoints in each API ML internal service (Gateway, Discovery Service, and Catalog).
2. List the available loggers of a service by issuing the GET request for the given service URL:

   ```
   GET scheme://hostname:port/application/loggers
   ```

   Where:

   - **scheme**

     API ML service scheme (http or https)
   - **hostname**

     API ML service hostname
   - **port**

     TCP port where API ML service listens on. The port is defined by the configuration parameter MFS_GW_PORT for the Gateway, MFS_DS_PORT for the Discovery Service (by default, set to gateway port + 1), and MFS_AC_PORT for the Catalog (by default, set to gateway port + 2).

   Exception: For the catalog you will able to get list the available loggers by issuing the GET request for the given service URL:

   ```
   GET [gateway-scheme]://[gateway-hostname]:[gateway-port]/api/v1/
   apicatalog/application/loggers
   ```

   Tip: One way to issue REST calls is to use the http command in the free HTTPie tool: https://httpie.org/.

   Example:

   ```
   HTTPie command:
   http GET https://lpar.ca.com:10000/application/loggers

   Output:
   {"levels":["OFF","ERROR","WARN","INFO","DEBUG","TRACE"],
    "loggers":{
      "ROOT":{"configuredLevel":"INFO","effectiveLevel":"INFO"},
      "com":{"configuredLevel":null,"effectiveLevel":"INFO"},
      "com.ca":{"configuredLevel":null,"effectiveLevel":"INFO"},
      ...
    }
   }
   ```
3. Alternatively, you extract the configuration of a specific logger using the extended **GET** request:

   ```
   GET scheme://hostname:port/application/loggers/{name}
   ```

   Where:

   - **{name}**

     is the logger name
4. Change the log level of the given component of the API ML internal service. Use the POST request for the given service URL:

   ```
   POST scheme://hostname:port/application/loggers/{name}
   ```

   The POST request requires a new log level parameter value that is provided in the request body:

   ```
   {

   "configuredLevel": "level"
   ```

```
}
```

Where:

- **level**

  is the new log level: **OFF**, **ERROR**, **WARN**, **INFO**, **DEBUG**, **TRACE**

Example:

```
http POST https://hostname:port/application/loggers/
org.zowe.apiml.enable.model configuredLevel=WARN
```

Known Issues

API ML stops accepting connections after z/OS TCP/IP stack is recycled

Symptom:

When z/OS TCP/IP stack is restarted, it is possible that the internal services of API Mediation Layer (Gateway, Catalog, and Discovery Service) stop accepting all incoming connections, go into a continuous loop, and write a numerous error messages in the log.

Sample message:

The following message is a typical error message displayed in STDOUT:

```
2018-Sep-12 12:17:22.850. org.apache.tomcat.util.net.NioEndpoint -- Socket
 accept failed java.io.IOException: EDC5122I Input/output error.

.at sun.nio.ch.ServerSocketChannelImpl.accept0(Native Method) ~.na:1.8.0.
.at
 sun.nio.ch.ServerSocketChannelImpl.accept(ServerSocketChannelImpl.java:478)
 ~.na:1.8.0.
.at
 sun.nio.ch.ServerSocketChannelImpl.accept(ServerSocketChannelImpl.java:287)
 ~.na:1.8.0.
.at org.apache.tomcat.util.net.NioEndpoint
$Acceptor.run(NioEndpoint.java:455) ~.tomcat-coyote-8.5.29.jar!/:8.5.29.
.at java.lang.Thread.run(Thread.java:811) .na:2.9 (12-15-2017).
```

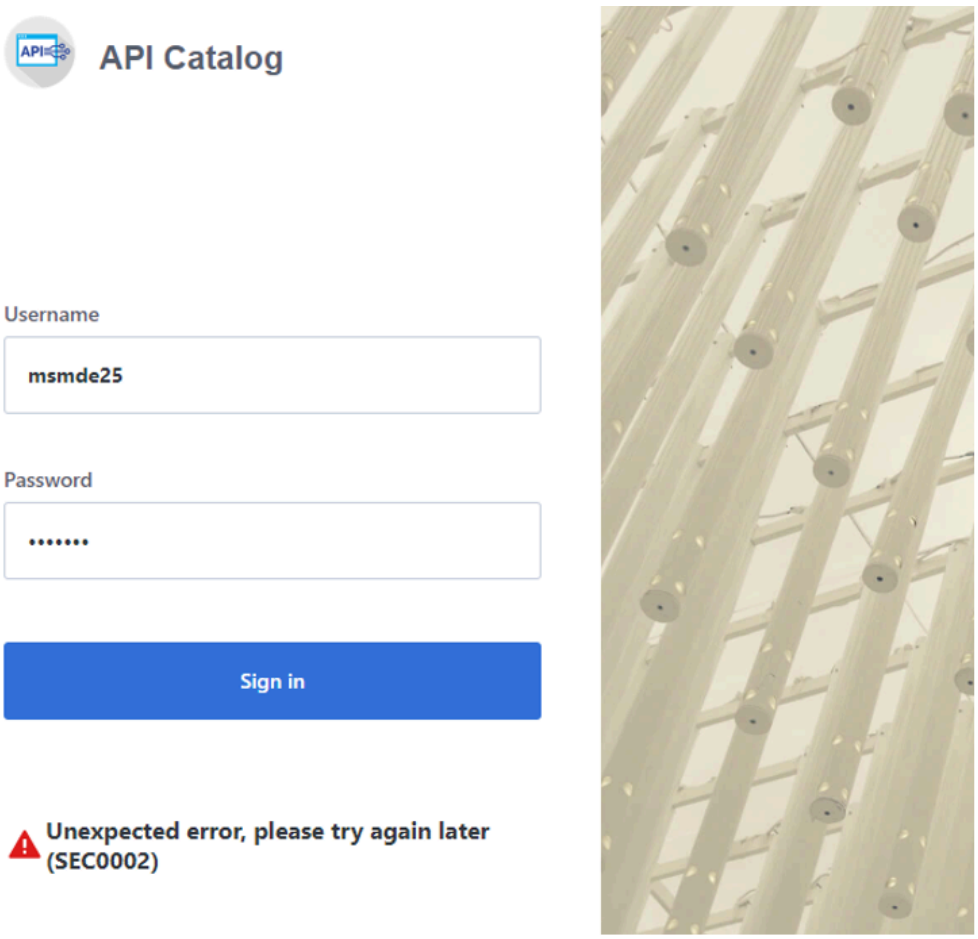
Solution:

Restart API Mediation Layer.

Tip: To prevent this issue from occurring, it is strongly recommended not to restart the TCP/IP stack while API ML is running.

SEC0002 error when logging in to API Catalog

SEC0002 error typically appears when users fail to log in to API Catalog. The following image shows the API Catalog login page with the SEC0002 error.



The error is caused by failed z/OSMF authentication. To determine the reason authentication failed, open the ZWESVSTC joblog and look for a message that contains `ZosmfAuthenticationProvider`. The following is an example of the message that contains `ZosmfAuthenticationProvider`:

```
2019-08-05 11:25:03.431 ERROR 5 --- .0.0-7552-exec-3.
 c.c.m.s.l.ZosmfAuthenticationProvider    : Can not access z/OSMF service.
 Uri 'https://ABC12.slv.broadcom.net:1443' returned: I/O error on GET
 request for "https://ABC12.slv.broadcom.net:1443/zosmf/info": ...
```

Check the rest of the message, and identify the cause of the problem. The following list provides the possible reasons and solutions for the z/OSMF authentication issue:

- Connection refused on page 285
- Missing z/OSMF host name in subject alternative names
- Invalid z/OSMF host name in subject alternative names

Connection refused

In the following message, failure to connect to API Catalog occurs when connection is refused:

```
Connect to ABC12.slv.broadcom.net:1443 .ABC12.slv.broadcom.net/127.0.0.1.
 failed: EDC8128I Connection refused.; nested exception is
 org.apache.http.conn.HttpHostConnectException:
```

The reason for the refused connection message is either invalid z/OSMF configuration or z/OSMF being unavailable. The preceding message indicates that z/OSMF is not on the 127.0.0.1:1443 interface.

Solution:

Configure z/OSMF

Make sure that z/OSMF is running and is on 127.0.0.1:1443 interface, and try to log in to API Catalog again. If you get the same error message, change z/OSMF configuration.

Follow these steps:

1. Locate the z/OSMF PARMLIB member IZUPRMxx.

   For example, locate IZUPRM00 member in SYS1.PARMLIB.
2. Change the current `HOSTNAME` configuration to `HOSTNAME('*')`.
3. Change the current `HTTP_SSL_PORT` configuration to `HTTP_SSL_PORT('1443')`.

   Important! If you change the port in the z/OSMF configuration file, all your applications lose connection to z/OSMF.

For more information, see Syntax rules for IZUPRMxx.

If changing the z/OSMF configuration does not fix the issue, reconfigure Zowe.

Follow these steps:

1. Open `.zowe_profile` in the home directory of the user who installed Zowe.
2. Modify the value of the `ZOWE_ZOSMF_PORT` variable.
3. Reinstall Zowe.

Missing z/OSMF host name in subject alternative names

In following message, failure to connect to API Catalog is caused by a missing z/OSMF host name in the subject alternative names:

```
Certificate for <ABC12.slv.broadcom.net> doesn't match any
 of the subject alternative names: ..; nested exception is
 javax.net.ssl.SSLPeerUnverifiedException: Certificate for
 <ABC12.slv.broadcom.net> doesn't match any of the subject alternative
 names: ..
```

Solutions:

Fix the missing z/OSMF host name in subject alternative names using the following methods:

Note: Apply the insecure fix only if you use API Catalog for testing purposes.

- Secure fix
- Insecure fix on page 287

Secure fix

Follow these steps:

1. Obtain a valid certificate for z/OSMF and place it in the z/OSMF keyring. For more information, see Configure the z/OSMF Keyring and Certificate.

2. Navigate to `$ZOWE_RUNTIME/components/api-mediation` and run the following command:

   ```
   scripts/apiml_cm.sh --action trust-zosmf
   ```

   2a. (Optional) If you do not use the default z/OSMF userid (IZUSVR) and keyring (IZUKeyring.IZUDFLT), issue the following command:

   ```
   scripts/apiml_cm.sh --action trust-zosmf--zosmf-userid **ZOSMF_USER** --
   zosmf-keyring **ZOSMF_KEYRING**
   ```

   where;

   - `--zosmf-keyring` and `--zosmf-userid` - options that override the default userid and keyring accordingly.

Insecure fix

Follow these steps:

1. Set the value of the `VERIFY_CERTIFICATES` property to `false` in `$ZOWE_RUNTIME/scripts/internal/run-zowe.sh` to disable verification of certificates in Zowe.
2. Reinstall Zowe.

Invalid z/OSMF host name in subject alternative names

In the following message, failure to connect to API Catalog is caused by an invalid z/OSMF host name in the subject alternative names:

```
Certificate for <ABC12.slv.broadcom.net> doesn't match any of the
 subject alternative names: [abc12.ca.com, abc12, localhost, abc12-slck,
 abc12-slck.ca.com, abc12-slck1, abc12-slck1.ca.com, abc12-slck2, abc12-
slck2.ca.com, usilabc12, usilabc12.ca.com];
nested exception is javax.net.ssl.SSLPeerUnverifiedException: Certificate
 for <ABC12.slv.broadcom.net> doesn't match any of the subject alternative
 names: [abc12.ca.com, abc12, localhost, abc12-slck, abc12-slck.ca.com,
 abc12-slck1, abc12-slck1.ca.com, abc12-slck2, abc12-slck2.ca.com,
 usilabc12, usilabc12.ca.com]
```

Solutions:

Fix the invalid z/OSMF host name in the subject alternative names using the following methods:

- Request a new certificate on page 287
- Change the ZOWE_EXPLORER_HOST variable

Request a new certificate

Request a new certificate that contains a valid z/OSMF host name in the subject alternative names.

Change the ZOWE_EXPLORER_HOST variable

Change `ZOWE_EXPLORER_HOST` variable to fix the issue.

Follow these steps:

1. Open .zowe_profile in the home directory of the user who installed Zowe.
2. Change `ZOWE_EXPLORER_HOST` to a host name from the subject alternative names of the z/OSMF certificate. For example, issue the following command:

   ```
   export ZOWE_EXPLORER_HOST=SAN (change this to the correct one > in the
    code block).
   ```

3. Reinstall Zowe.

Error Message Codes

The following error message codes may appear on logs or API responses. Use the following message code references and the corresponding reasons and actions to help troubleshoot issues.

API mediation common messages

ZWEAO102E

Gateway not found yet, transform service cannot perform the request

Reason:

The Transform service was requested to transform a url, but the Gateway instance was discovered.

Action:

Do not begin performing requests until the API Mediation Layer fully initializes after startup. Check that your Discovery service is running and that all services (especially the Gateway) are discovered correctly.

ZWEAO104W

GatewayInstanceInitializer has been stopped due to exception: %s

Reason:

An unexpected exception occurred while retrieving the Gateway service instance from the Discovery Service.

Action:

Check that both the service and the Gateway can register with Discovery. If the services are not registering, investigate the reason why. If no cause can be determined, create an issue.

ZWEAO401E

Unknown error in HTTPS configuration: '%s'

Reason:

An Unknown error occurred while setting up an HTTP client during service initialization, followed by a system exit.

Action:

Start the service again in debug mode to get a more descriptive message. This error indicates it is not a configuration issue.

Common service core messages

ZWEAM000I

%s has been started in %s seconds

Reason:

The service has been started

Action:

No action is needed

ZWEAM100E

Could not read properties from: '%s'

Reason:

The Build Info properties file is empty or null.

Action:

The jar file is not packaged correctly. Please submit an issue.

ZWEAM101E

I/O Error reading properties from: '%s' Details: '%s'

Reason:

I/O error reading `META-INF/build-info.properties` or `META-INF/git.properties`

Action:

The jar file is not packaged correctly. Please submit an issue.

ZWEAM102E

Internal error: Invalid message key '%s' is provided. Please create an issue with this message.

Reason:

Message service is requested to create message with an invalid key.

Action:

Create an issue with this message.

ZWEAM103E

Internal error: Invalid message text format. Please create an issue with this message.

Reason:

Message service is requested to create a message with an invalid text format.

Action:

Create an issue with this message.

ZWEAM104E

The endpoint you are looking for '%s' could not be located

Reason:

The endpoint you are looking for could not be located.

Action:

Verify that the URL of the endpoint you are trying to reach is correct.

ZWEAM400E

Error initializing SSL Context: '%s'

Reason:

An error occurred while initializing the SSL Context.

Action:

Refer to the specific message to identify the exact problem. Possible causes include:

- Incorrect security algorithm
- The keystore is invalid or corrupted
- The certificate is invalid or corrupted

ZWEAM500W

The service is not verifying the TLS/SSL certificates of the services

Reason:

This is a warning that the SSL Context will be created without verifying certificates.

Action:

Stop the service and set the verifySslCertificatesOfServices parameter to `true`. Then restart the service. Do not use this option in a production environment.

ZWEAM501W

Service is connecting to Discovery service using insecure HTTP protocol.

Reason:

The service is connecting to the Discovery Service using the non-secure HTTP protocol.

Action:

For production use, start the Discovery Service in HTTPS mode and configure the services accordingly.

ZWEAM502E

Error reading secret key: '%s'

Reason:

A key with the specified alias cannot be loaded from the keystore.

Action:

Ensure that the configured key is present, in the correct format, and not corrupt.

ZWEAM503E

Error reading secret key: '%s'

Reason:

Error reading secret key.

Action:

Refer to the specific message to identify the exact problem. Possible causes include:

- An incorrect security algorithm
- The keystore is invalid or corrupted
- The certificate is invalid or corrupted

ZWEAM504E

Error reading public key: '%s'

Reason:

Error reading secret key.

Action:

Refer to the specific message to identify the exact problem. Possible causes include:

- An incorrect security algorithm
- The keystore is invalid or corrupted
- The certificate is invalid or corrupted

ZWEAM505E

Error initializing SSL/TLS context: '%s'

Reason:

Error initializing SSL/TLS context.

Action:

Refer to the specific message to identify the exact problem. Possible causes include:

- An incorrect security algorithm

- The keystore is invalid or corrupted
- The certificate is invalid or corrupted

ZWEAM506E

Truststore Password configuration parameter is not defined

Reason:

Your truststore password was not set in the configuration.

Action:

Ensure that the parameter server.ssl.trustStorePassword contains the correct password for your truststore.

ZWEAM507E

Truststore configuration parameter is not defined but it is required

Reason:

The truststore usage is mandatory, but the truststore location is not provided.

Action:

If a truststore is required, define the truststore configuration parameter by editing the server.ssl.truststore, server.ssl.truststorePassword and server.ssl.truststoreType parameters with valid data. If you do not require a truststore, change the trustStoreRequired boolean parameter to `false`.

ZWEAM508E

Keystore not found, server.ssl.keyStore configuration parameter is not defined

Reason:

Your keystore path was not set in the configuration.

Action:

Ensure that the correct path to your keystore is contained in the parameter server.ssl.keyStore in the properties or yaml file of your service.

ZWEAM509E

Keystore password not found, server.ssl.keyStorePassword configuration parameter is not defined

Reason:

Your keystore password was not set in the configuration.

Action:

Ensure that the correct password to your keystore in the parameter server.ssl.keyStorePassword is contained in the properties or yaml file of your service.

ZWEAM510E

Invalid key alias '%s'

Reason:

The key alias was not found.

Action:

Ensure that the key alias provided for the key exists in the provided keystore.

ZWEAM511E

The certificate of the service accessed using URL '%s' is not trusted by the API Gateway: %s

Reason:

Thr Gateway does not trust the requested service and refuses to communicate with it. The certificate of the service is missing from the truststore of the API Mediation Layer.

Action:

Contact your administrator to verify API Mediation Layer truststore configuration.

ZWEAM600W

Invalid parameter in metadata: '%s'

Reason:

An invalid apiInfo parameter was found while parsing the service metadata.

Action:

Remove or fix the referenced metadata parameter.

ZWEAM700E

No response received within the allowed time: %s

Reason:

No response was received within the allowed time.

Action:

Verify that the URL you are trying to reach is correct and all services are running.

ZWEAM701E

The request to the URL '%s' has failed: %s caused by: %s

Reason:

Request failed because of internal error.

Action:

Refer to specific exception details for troubleshooting. Create an issue with this message.

Security common messages

ZWEAT103E

Could not write response: %s

Reason:

A message could not be written to the response

Action:

Please submit an issue with this message.

ZWEAT601E

z/OSMF service name not found. Set parameter apiml.security.auth.zosmfServiceId to your service ID.

Reason:

The parameter zosmfserviceId was not configured correctly and could not be validated.

Action:

Ensure that the parameter apiml.security.auth.zosmfServiceId is correctly entered with a valid zosmf service ID.

Security client messages

ZWEAS100E

Authentication exception: '%s' for URL '%s'

Reason:

A generic failure occurred while authenticating.

Action:

Refer to the specific message to troubleshoot.

ZWEAS101E

Authentication method '%s' is not supported for URL '%s'

Reason:

The HTTP request method is not supported for the URL.

Action:

Use the correct HTTP request method that is supported for the URL.

ZWEAS102E

Token is expired for URL '%s'

Reason:

The validity of the token is expired.

Action:

Obtain new token by performing an authentication request.

ZWEAS103E

API Gateway Service is not available by URL '%s' (API Gateway is required because it provides the authentication functionality)

Reason:

The security client cannot find a Gateway instance to perform authentication. The API Gateway is required because it provides the authentication functionality.

Action:

Check that both the service and Gateway are correctly registered in the Discovery service. Allow some time after the services are discovered for the information to propagate to individual services.

ZWEAS104E

Authentication service is not available by URL '%s'

Reason:

Authentication service is not available.

Action:

Make sure that authentication service is running and is accessible by the URL provided in the message.

ZWEAS105E

Authentication is required for URL '%s'

Reason:

Authentication is required.

Action:

Provide valid authentication.

ZWEAS120E

Invalid username or password for URL '%s'

Reason:

The username or password are invalid.

Action:

Provide a valid username and password.

ZWEAS121E

Authorization header is missing, or request body is missing or invalid for URL '%s'

Reason:

The authorization header is missing, or the request body is missing or invalid.

Action:

Provide valid authentication.

ZWEAS130E

Token is not valid for URL '%s'

Reason:

The token is not valid.

Action:

Provide a valid token.

ZWEAS131E

No authorization token provided for URL '%s'

Reason:

No authorization token is provided.

Action:

Provide a valid authorization token.

Discovery service messages

ZWEAD700W

Static API definition directory '%s' is not a directory or does not exist

Reason:

One of the specified static API definition directories does not exist or is not a directory.

Action:

Review the static API definition directories and their setup. The static definition directories are specified as a launch parameter to a Discovery service jar. The property key is: `apiml.discovery.staticApiDefinitionsDirectories`

ZWEAD701E

Error loading static API definition file '%s'

Reason:

A problem occurred while reading (IO operation) of a specific static API definition file.

Action:

Ensure that the file data is not corrupted or incorrectly encoded.

ZWEAD702W

Unable to process static API definition data: '%s'

Reason:

A problem occurred while parsing a static API definition file.

Action:

Review the mentioned static API definition file for errors. Refer to the specific log message to see what is the exact cause of the problem:

- ServiceId is not defined in the file '%s'. The instance will not be created. Make sure to specify the ServiceId.
- The `instanceBaseUrls` parameter of %s is not defined. The instance will not be created. Make sure to specify the `InstanceBaseUrl` property.
- The API Catalog UI tile ID %s is invalid. The service %s will not have an API Catalog UI tile. Specify the correct catalog title ID.
- One of the instanceBaseUrl of %s is not defined. The instance will not be created. Make sure to specify the InstanceBaseUrl property.
- The URL %s does not contain a hostname. The instance of %s will not be created. The specified URL is malformed. Make sure to specify valid URL.
- The URL %s does not contain a port number. The instance of %s will not be created.
- The specified URL is missing a port number. Make sure to specify a valid URL.
- The URL %s is malformed. The instance of %s will not be created: The Specified URL is malformed. Make sure to specify a valid URL.
- The hostname of URL %s is unknown. The instance of %s will not be created: The specified hostname of the URL is invalid. Make sure to specify valid hostname.
- Invalid protocol. The specified protocol of the URL is invalid. Make sure to specify valid protocol.

ZWEAD703E

A problem occurred during reading the static API definition directory: '%s'

Reason:

There are three possible causes of this error:

- The specified static API definition folder is empty
- The definition does not denote a directory
- An I/O error occurred while attempting to read the static API definition directory.

Action:

Review the static API definition directory definition and its contents on the storage. The static definition directories are specified as a parameter to launch a Discovery service jar. The property key is: `apiml.discovery.staticApiDefinitionsDirectories`

Gateway service messages

ZWEAG700E

No instance of the service '%s' found. Routing will not be available.

Reason:

The Gateway could not find an instance of the service from the Discovery Service.

Action:

Check that the service was successfully registered to the Discovery Service and wait for Spring Cloud to refresh the routes definitions

ZWEAG701D

Service '%s' does not allow encoded characters used in request path: '%s'.

Reason:

The request that was issued to the Gateway contains an encoded character in the URL path. The service that the request was addressing does not allow this pattern.

Action:

Contact the system administrator and ask to enable encoded characters in the service

ZWEAG704E

Configuration error '%s' when trying to read jwt secret: %s

Reason:

A problem occurred while trying to read the jwt secret key from the keystore.

Action:

Review the mandatory fields used in the configuration such as the keystore location path, the keystore and key password, and the keystore type.

ZWEAG705E

Failed to load public or private key from key with alias '%s' in the keystore '%s'.

Reason:

Failed to load a public or private key from the keystore during JWT Token initialization.

Action:

Check that the key alias is specified and correct. Verify that the keys are present in the keystore.

ZWEAG100E

Authentication exception: '%s' for URL '%s'

Reason:

A generic failure occurred during authentication.

Action:

Refer to specific authentication exception details for troubleshooting.

ZWEAG101E

Authentication method '%s' is not supported for URL '%s'

Reason:

The HTTP request method is not supported by the URL.

Action:

Use the correct HTTP request method supported by the URL.

ZWEAG102E

Token is not valid

Reason:

The JWT token is not valid

Action:

Provide a valid token.

ZWEAG103E

Token is expired

Reason:

The JWT token has expired

Action:

Obtain new token by performing an authentication request.

ZWEAG104E

Authentication service is not available at URL '%s'. Error returned: '%s'

Reason:

The authentication service is not available.

Action:

Make sure that the authentication service is running and is accessible by the URL provided in the message.

ZWEAG105E

Authentication is required for URL '%s'

Reason:

Authentication is required.

Action:

Provide valid authentication.

ZWEAG106W

Login endpoint is running in the dummy mode. Use credentials user/user to login. Do not use this option in the production environment.

Reason:

The authentication is running in dummy mode.

Action:

Do not use this option in the production environment.

ZWEAG107W

Incorrect value: apiml.security.auth.provider = '%s'. Authentication provider is not set correctly. Default 'zosmf' authentication provider is used.

Reason:

An incorrect value of the apiml.security.auth.provider parameter is set in the configuration.

Action:

Ensure that the value of apiml.security.auth.provider is set either to 'dummy' if you want to use dummy mode, or to 'zosmf' if you want to use the z/OSMF authentication provider.

ZWEAG108E

z/OSMF instance '%s' not found or incorrectly configured.

Reason:

The Gateway could not find the z/OSMF instance from the Discovery Service.

Action:

Ensure that the z/OSMF instance is configured correctly and that it is successfully registered to the Discovery Service.

ZWEAG109E

z/OSMF response does not contain field '%s'.

Reason:

The z/OSMF domain cannot be read.

Action:

Review the z/OSMF domain value contained in the response received from the 'zosmf/info' REST endpoint.

ZWEAG110E

Error parsing z/OSMF response. Error returned: '%s

Reason:

An error occurred while parsing the z/OSMF JSON response.

Action:

Check the JSON response received from the 'zosmf/info' REST endpoint.

ZWEAG120E

Invalid username or password for URL '%s'

Reason:

The username or password are invalid.

Action:

Provide a valid username and password.

ZWEAG121E

Authorization header is missing, or request body is missing or invalid for URL '%s'

Reason:

The authorization header is missing, or the request body is missing or invalid.

Action:

Provide valid authentication.

ZWEAG130E

Token is not valid for URL '%s'

Reason:

The token is not valid.

Action:

Provide a valid token.

ZWEAG131E

No authorization token provided for URL '%s'

Reason:

No authorization token is provided.

Action:

Provide a valid authorization token.

API Catalog messages

ZWEAC100W

Could not retrieve all service info from discovery -- %s -- %s -- %s

Reason:

The response from The Discovery Service about the registered instances returned an error or empty body.

Action:

Make sure the Discovery service is up and running. If the http response error code refers to a security issue, check that both the Discovery Service and Catalog are running with the https scheme and that security is configured properly.

ZWEAC101E

Could not parse service info from discovery -- %s

Reason:

The response from the Discovery Service about the registered instances could not be parsed to extract applications.

Action:

Run debug mode and look at the Discovery Service potential issues while creating a response. If the Discovery Service does not indicate any error, create an issue.

ZWEAC102E

Could not retrieve containers. Status: %s

Reason:

One or more containers could not be retrieved.

Action:

Check the status of the message for more information and the health of the Discovery Service.

ZWEAC103E

API Documentation not retrieved, %s

Reason:

API documentation was not found.

Action:

Make sure the service documentation is configured correctly.

ZWEAC104E

Could not retrieve container statuses, %s

Reason:

One or more containers statuses could not be retrieved.

Action:

Check the status of the message for more information and the health of Discovery Service.

ZWEAC700E

Failed to update cache with discovered services: '%s'

Reason:

Cache could not be updated.

Action:

Check the status of the Discovery Service.

ZWEAC701W

API Catalog Instance not retrieved from Discovery service

Reason:

An error occurred while fetching containers information.

Action:

The jar file is not packaged correctly. Please submit an issue.

ZWEAC702E

An unexpected exception occurred when trying to retrieve an API Catalog instance from the Discovery Service: %s

Reason:

An unexpected error occurred during API Catalog initialization. The API Catalog was trying to locate an instance of itself in the Discovery Service.

Action:

Review the specific message for more information. Verify if the Discovery Service and service registration work as expected.

ZWEAC703E

Failed to initialize API Catalog with discovered services

Reason:

The API Catalog could not initialize running services after several retries.

Action:

Ensure services are started and discovered properly.

ZWEAC704E

ApiDoc retrieval problem for service %s. %s

Reason:

ApiDoc for service could not be retrieved from cache.

Action:

Verify that the service provides a valid ApiDoc.

ZWEAC705W

The home page url for service %s was not transformed. %s

Reason:

The home page url for service was not transformed. The original url will be used.

Action:

Refer to the specific printed message. Possible causes include:

- The Gateway was not found. Transform service cannot perform the request. Wait for the Gateway to be discovered.
- The URI ... is not valid. Ensure the service is providing a valid url.
- Not able to select a route for url ... of the service ... Original url is used. If this is a problem, check the routing metadata of the service.
- The path ... of the service URL ... is not valid. Ensure the service is providing the correct path.

ZWEAC706E

Service not located, %s

Reason:

The service could not be found.

Action:

Check if the service is up and registered. If it is not registered, review the onboarding guide to ensure that all steps were completed.

Zowe Application Framework

Troubleshooting Zowe Application Framework

The following topics contain information that can help you troubleshoot problems when you encounter unexpected behavior installing and using Zowe™ Application Framework.

Before you reach out for support:

1. Is there already a GitHub issue (open or closed) that covers the problem? Check Application Framework issues.
2. Review the current list of Known Zowe Application Framework issues on page 303 in documentation. Also try searching using the Zowe Docs search bar.

Gathering information to troubleshoot Zowe Application Framework

Gather the following information to troubleshoot Zowe™ Application Framework issues:

- z/OS release level
- Zowe version and release level on page 301
- Zowe application configuration on page 302
- Zowe Application Server ports on page 302
- Log output from the Zowe Application Server on page 302
- Error message codes on page 303
- Javascript console output on page 303
- Screen captures on page 303
- Other relevant information on page 303

z/OS release level

To find the z/OS release level, issue the following command in SDSF:

```
/D IPLINFO
```

Check the output for the release level, for example:

```
RELEASE z/OS 02.02.00
```

Zowe version and release level

```
  cd <zowe-installation-directory>
  cat manifest.json
```

Output:

Displays zowe version

```
  {
    "name": "Zowe",
    "version": "1.2.0",
```

```
   "description": "Zowe is an open source project created to host
technologies that benefit the Z platform from all members of the Z
community (Integrated Software Vendors, System Integrators and z/OS
consumers). Zowe, like Mac or Windows, comes with a set of APIs and OS
capabilities that applications build on and also includes some applications
out of the box. Zowe offers modern interfaces to interact with z/OS
and allows you to work with z/OS in a way that is similar to what you
experience on cloud platforms today. You can use these interfaces as
delivered or through plug-ins and extensions that are created by clients or
third-party vendors.",
   "license": "EPL-2.0",
   "homepage": "https://zowe.org",
   "build": {
     "branch": "master",
     "number": 685,
     "commitHash": "63efa85df629db474197ec8481db50021e8fdd65",
     "timestamp": "1556733977010"
   }
 }
```

Zowe application configuration

Configuration file helps customize the Zowe app server, and is important to look at while you troubleshoot.

```
# navigate to zowe installation folder
cd <zowe-installation-folder>

# navigate to server configuration folder
cd zlux-app-server/deploy/instance/ZLUX/serverConfig

# display config
cat zluxserver.json
```

Read more about the Zowe app server Zowe Application Framework configuration on page 108 in the Zowe User Guide.

Zowe Application Server ports

```
# navigate to zowe installation folder
cd <zowe-installation-folder>

# navigate to install log directory
cd install_log

# list file by most recent first
ls -lt

# pick latest file
cat 2019-05-02-17-13-09.log | grep ZOWE_ZLUX_SERVER_HTTPS_PORT
cat 2019-05-02-17-13-09.log | grep ZOWE_ZSS_SERVER_PORT
```

Log output from the Zowe Application Server

There are two major components of Zowe application server: ZLUX and ZSS. They log to different files.

The default location for logs for both zlux and zss is folder `zlux-app-server/log`. You can customize the log location by using the environment variable.

```
env | grep ZLUX_NODE_LOG_DIR
env | grep ZSS_LOG_DIR
```

Read more about controlling the log location Controlling the logging location on page 120.

```
# navigate to zowe installation folder
cd <zowe-installation-folder>

# navigate to logs default location or custom location as described above
cd zlux-app-server/log

# custom log location can be found using environment variable

# list file by most recent first
ls -lt
```

Output:

List of files by most recent timestamp for both nodeServer as well ZSS.

```
nodeServer-<yyyy-mm-dd-hh-mm>.log
zssServer-<yyyy-mm-dd-hh-mm>.log
```

Error message codes

It is advisable to look into log files for capturing error codes.

Javascript console output

Web Developer toolkit is accessible by pressing F12.

Read more about it here.

Screen captures

If possible, add a screen capture of the issue.

Other relevant information

Node.js – v6.14.4 minimum for z/OS, elsewhere v6, v8, and v10 work well.

```
node -v
```

npm – v6.4 minimum

```
npm -v
```

Java – v8 minimum

```
java -version
```

Known Zowe Application Framework issues

The following topics contain information that can help you troubleshoot problems when you encounter unexpected behavior installing and using Zowe™ Application Framework.

Most of the solutions below identify issues by referring to the Zowe Gathering information to troubleshoot Zowe Application Framework on page 301. To identify and resolve issues, you should become familiar with their names and locations.

Desktop apps fail to load

Symptom:

When you open apps in the desktop they display a page with the message "The plugin failed to load."

Solution:

NodeJS v8.16.1 performs auto-encoding in a way that breaks Zowe apps. See https://github.com/ibmruntimes/node/issues/142 for details.

Use node v8.16.0 which is available at https://www.ibm.com/ca-en/marketplace/sdk-nodejs-compiler-zos. Download the `ibm-trial-node-v8.16.0-os390-s390x.pax.Z` file.

NODEJSAPP disables immediately

Symptom:

If you receive the message `CEE5207E The signal SIGABRT was received in stderr`, you might have reached the limit for shared message queues on your LPAR.

Solution:

When Node.js applications are terminated by a SIGKILL signal, shared message queues might not be deallocated. In the IBM "Troubleshooting Node.js applications" documentation, see the section titled **If the NODEJSAPP disables immediately**.

Cannot log in to the Zowe Desktop

Symptom:

When you attempt to log in to the Zowe Desktop, you receive the following error message, displayed beneath the **Username** and **Password** fields.

```
Authentication failed for 1 types:  Types: ["zss"]
```

Solution:

For the Zowe Desktop to work, the node server that runs under the ZWESVSTC started task must be able to make cross memory calls to the ZWESIS01 load module running under the ZWESISTC started task. If this communication fails, you see the authentication error.

To solve the problem, follow these steps:

1. Open the log file `/zlux-app-server/log/zssServer-yyyy-mm-dd-hh-ss.log`. This file is created each time ZWESVSTC is started and only the last five files are kept.

2. Look for the message that starts with `ZIS status`.

   If communication is working the message includes `Ok`. For example:

   ```
   ZIS status - Ok (name='ZWESIS_STD      ', cmsRC=0, description='Ok'
   ```

   If communication is not working the message includes `Failure`. For example:

   ```
   ZIS status - Failure (name='ZWESIS_STD      ', cmsRC=39,
    description='Cross-memory call ABENDed'
   ```

   If communication is not working, check that the ZWESISTC started task is running. If not, start it. Also, search the log for problems, for example statements saying that the server was unable to find the load module.

   If the problem is not easily-fixable (such as the ZWESISTC task not running), then it is likely that the cross memory server setup and configuration did not complete successfully. To set up and configure the cross memory server, follow steps as described in the topic Manually installing the Zowe Cross Memory Server.

   If there is an authorization problem, the message might include `Permission Denied`. For example:

   ```
   ZIS status - Failure (name='ZWESIS_STD      ', cmsRC=33,
    description='Permission denied'
   ```

   Check that the user ID of the ZWESVSTC started task is authorized to access the load module. Only authorized code can call ZWESIS01 because it is an APF-authorized load module. The setup for each security manager is different and is documented in the section "Security requirements for the cross memory server" in the topic Manually installing the Zowe Cross Memory Server.

   Note If you are using RACF security manager, a common reason for seeing `Permission Denied` is that the user running the started task ZWESVSTC (typically IZUSVR) does not have READ access to the FACILITY class ZWES.IS.

   If the message includes the following text, the configuration of the Application Framework server may be incomplete:

   ```
   ZIS status - Failure read failed ret code 1121 reason 0x76650446
   ```

   If you are using AT/TLS, then the `"attls" : true` statement might be missing from the `zluxserver.json` file. For more information, see Configuring ZSS for HTTPS on page 112

Server startup problem ret=1115

Symptom: When ZWESVSTC is restarted, the following message is returned in the output of the ZSS server log file, `/zlux-app-server/log/zssServer-yyyy-mm-dd-hh-ss.log`:

```
server startup problem ret=1115
```

Solution: This message means that some other process is already listening on port 7542, either at address 127.0.0.1 (localhost) or at 0.0.0.0 (all addresses). This prevents the ZSS server from starting.

One possibility is that a previously running ZSS server did not shut down correctly, and either the operating system has not released the socket after the ZSS server shut down, or the ZSS server is still running.

Application plug-in not in Zowe Desktop

Symptom: An application plug-in is not appearing in the Zowe Desktop.

Troubleshooting: To check if the plug-in loaded successfully, enter the following URL in a browser to display all successfully loaded Zowe plug-ins:

`https://my.mainframe.com:8544/plugins?type=application`

You can also search the Gathering information to troubleshoot Zowe Application Framework on page 301 for the plug-in identifier, for example `org.zowe.sample.app`. If the plug-in loaded successfully, you will find the following message:

```
[2019-08-06 13:54:21.341 _zsf.bootstrap INFO] - Plugin org.zowe.sampleapp at
 path=zlux\org.zowe.sampleapp loaded.
```

If the plug-in did not load successfully, you will find the following message:

```
[2019-08-06 13:54:21.208 _zsf.bootstrap WARNING] - Error:
 org.zowe.sampleapp
```

If the identifier is not in the logs, make sure the plug-in's locator file is in the `/zlux-app-server/deploy/instance/ZLUX/plugins/` directory. The plug-in locator is a `.json` file, usually with same name as the identifier, for example `org.zowe.sampleapp.json`. Open the file and make sure that the path defined with the `pluginLocation` attribute is correct. If the path is relative, make sure it is relative to the `zlux-app-server/bin` directory.

For more information on loading plug-ins to the Desktop, see Adding Your App to the Desktop.

Error: You must specify MVD_DESKTOP_DIR in your environment

Symptom:

A plug-in build in your local environment using `npm run start` or `npm run build` failed with an error message about a missing MVD_DESKTOP_DIR environment variable.

Solution: Add the Zowe Desktop directory path to the MVD_DESKTOP_DIR environment variable. To specify the path, run the following commands in your Windows console or Linux bash shell:

- Windows

  ```
  export MVD_DESKTOP_DIR=<zlux-root-dir>/zlux-app-manager/virtual-desktop
  ```

- Mac Os/Linux

  ```
  set MVD_DESKTOP_DIR=<zlux-root-dir>/zlux-app-manager/virtual-desktop
  ```

Raising a Zowe Application Framework issue on GitHub

When necessary, you can raise GitHub issues against the Zowe™ zlux core repository here. It is suggested that you use either the bug or enhancement template.

For issues with particular applications, such as Code Editor or JES Explorer, create the issue in the application's project.

Raising a bug report

Please provide as much of the information listed on Troubleshooting Zowe Application Framework on page 301 as possible. Anyone working on the issue might need to request this and other information if it is not supplied initially. A description of the error and how it can be reproduced is the most important information.

Raising an enhancement report

Enhancement reports are just as important to the Zowe project as bug reports. Enhancement reports should be clear and detailed requirements for a potential enhancement.

Troubleshooting z/OS Services

The following topics contain information that can help you troubleshoot problems when you encounter unexpected behavior installing and using Zowe™ z/OS Services.

z/OS Services are unavailable

Solution:

If the z/OS Services are unavailable, take the following corrective actions.

- Ensure that the z/OSMF REST API services are working. Check the z/OSMF IZUSVR1 task output for errors and confirm that the z/OSMF RESTFILES services are started successfully. If no errors occur, you can see the following message in the IZUSVR1 job output:

  ```
  CWWKZ0001I: Application IzuManagementFacilityRestFiles started in n.nnn
   seconds.
  ```

  To test z/OSMF REST APIs you can run curl scripts from your workstation.

  ```
  curl --user <username>:<password> -k -X GET --header 'Accept: application/
  json' --header 'X-CSRF-ZOSMF-HEADER: true' "https://<z/os host
   name>:<securezosmfport>/zosmf/restjobs/jobs?prefix=*&owner=*
  ```

  where the *securezosmfport* is 443 by default. You can verify the port number by checking the *izu.https.port* variable assignment in the z/OSMF `bootstrap.properties` file.

  If z/OSMF returns jobs correctly, you can test whether it is able to returns files by using the following curl scripts:

  ```
  curl --user <username>:<password> -k -X GET --header 'Accept: application/
  json' --header 'X-CSRF-ZOSMF-HEADER: true' "https://<z/os host
   name>:<securezosmfport>/zosmf/restfiles/ds?dslevel=SYS1"
  ```

  If the restfiles curl statement returns a TSO SERVLET EXCEPTION error, check that the the z/OSMF installation step of creating a valid IZUFPROC procedure in your system PROCLIB has been completed. For more information, see the z/OSMF Configuration Guide.

  The IZUFPROC member resides in your system PROCLIB, which is similar to the following sample:

  ```
  //IZUFPROC PROC ROOT='/usr/lpp/zosmf'  /* zOSMF INSTALL ROOT     */
  //IZUFPROC EXEC PGM=IKJEFT01,DYNAMNBR=200
  //SYSEXEC  DD DISP=SHR,DSN=ISP.SISPEXEC
  //         DD DISP=SHR,DSN=SYS1.SBPXEXEC
  //SYSPROC  DD DISP=SHR,DSN=ISP.SISPCLIB
  //         DD DISP=SHR,DSN=SYS1.SBPXEXEC
  //ISPLLIB  DD DISP=SHR,DSN=SYS1.SIEALNKE
  //ISPPLIB  DD DISP=SHR,DSN=ISP.SISPPENU
  //ISPTLIB  DD RECFM=FB,LRECL=80,SPACE=(TRK,(1,0,1))
  //         DD DISP=SHR,DSN=ISP.SISPTENU
  //ISPSLIB  DD DISP=SHR,DSN=ISP.SISPSENU
  //ISPMLIB  DD DISP=SHR,DSN=ISP.SISPMENU
  //ISPPROF  DD DISP=NEW,UNIT=SYSDA,SPACE=(TRK,(15,15,5)),
  //         DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
  //IZUSRVMP DD PATH='&ROOT./defaults/izurf.tsoservlet.mapping.json'
  //SYSOUT   DD SYSOUT=H
  //CEEDUMP  DD SYSOUT=H
  //SYSUDUMP DD SYSOUT=H
  ```

```
//
```

Note: You might need to change paths and data sets names to match your installation.

A known issue and workaround for RESTFILES API can be found at TSO SERVLET EXCEPTION ATTEMPTING TO USE RESTFILE INTERFACE.

- Check your system console log for related error messages and respond to them.

Zowe CLI

Troubleshooting Zowe CLI

Problem

Zowe™ CLI is experiencing a problem. You need to collect information that will help you resolve the issue.

Environment

These instructions apply to Zowe CLI installed on Windows, Mac OS X, and Linux systems as a standalone installation via a Zowe download or an NPM registry.

Before reaching out for support

1. Is there already a GitHub issue (open or closed) that covers the problem? Check CLI Issues.
2. Review the current list of Known Zowe CLI issues on page 311 in documentation. Also try searching using the Zowe Docs search bar.

Resolving the problem

Collect the following information to help diagnose the issue:

- Zowe CLI version installed.
- List of plug-ins installed and their version numbers.
- Node.js and NPM versions installed.
- List of environment variables in use.

For instructions on how to collect the information, see Gathering information to troubleshoot Zowe CLI on page 308.

The following information is also useful to collect:

- If you are experiencing HTTP errors, see z/OSMF troubleshooting on page 311 for information to collect.
- Is the CLI part of another Node application, such as VSCode, or is it a general installation?
- Which operating system version are you running on?
- What shell/terminal are you using (bash, cmd, powershell, etc...)?
- Which queue managers are you trying to administer, and on what systems are they located?
- Are the relevant API endpoints online and valid?

Gathering information to troubleshoot Zowe CLI

Follow these instructions to gather specific pieces of information to help troubleshoot Zowe™ CLI issues.

[]

Identify the currently installed CLI version

Issue the following command:

```
zowe -V
```

Zowe CLI versions may vary depending upon if the `@latest` or `@lts-incremental` version is installed.

For the `@latest` (forward-development) version:

```
npm list -g @zowe/cli
```

For the `@lts-incremental` version:

```
npm list -g @brightside/core
```

More information regarding versioning conventions for Zowe CLI and plug-ins is located in Versioning Guidelines.

Identify the currently installed versions of plug-ins

Issue the following command:

```
zowe plugins list
```

The output describes version and the registry information. Note that the offical downloads are located at https://api.bintray.com/npm

Environment variables

The following settings are configurable via environment variables:

Log levels

Environment variables are available to specify logging level and the CLI home directory.

Important\! Setting the log level to TRACE or ALL might result in "sensitive" data being logged. For example, command line arguments will be logged when TRACE is set.

Environment Variable	Description	Values	Default
`ZOWE_APP_LOG_LEVEL`	Zowe CLI logging level	Log4JS log levels (OFF, TRACE, DEBUG, INFO, WARN, ERROR, FATAL)	DEBUG
`ZOWE_IMPERATIVE_LOG_LEVEL`	Imperative CLI Framework logging level	Log4JS log levels (OFF, TRACE, DEBUG, INFO, WARN, ERROR, FATAL)	DEBUG

Home directory

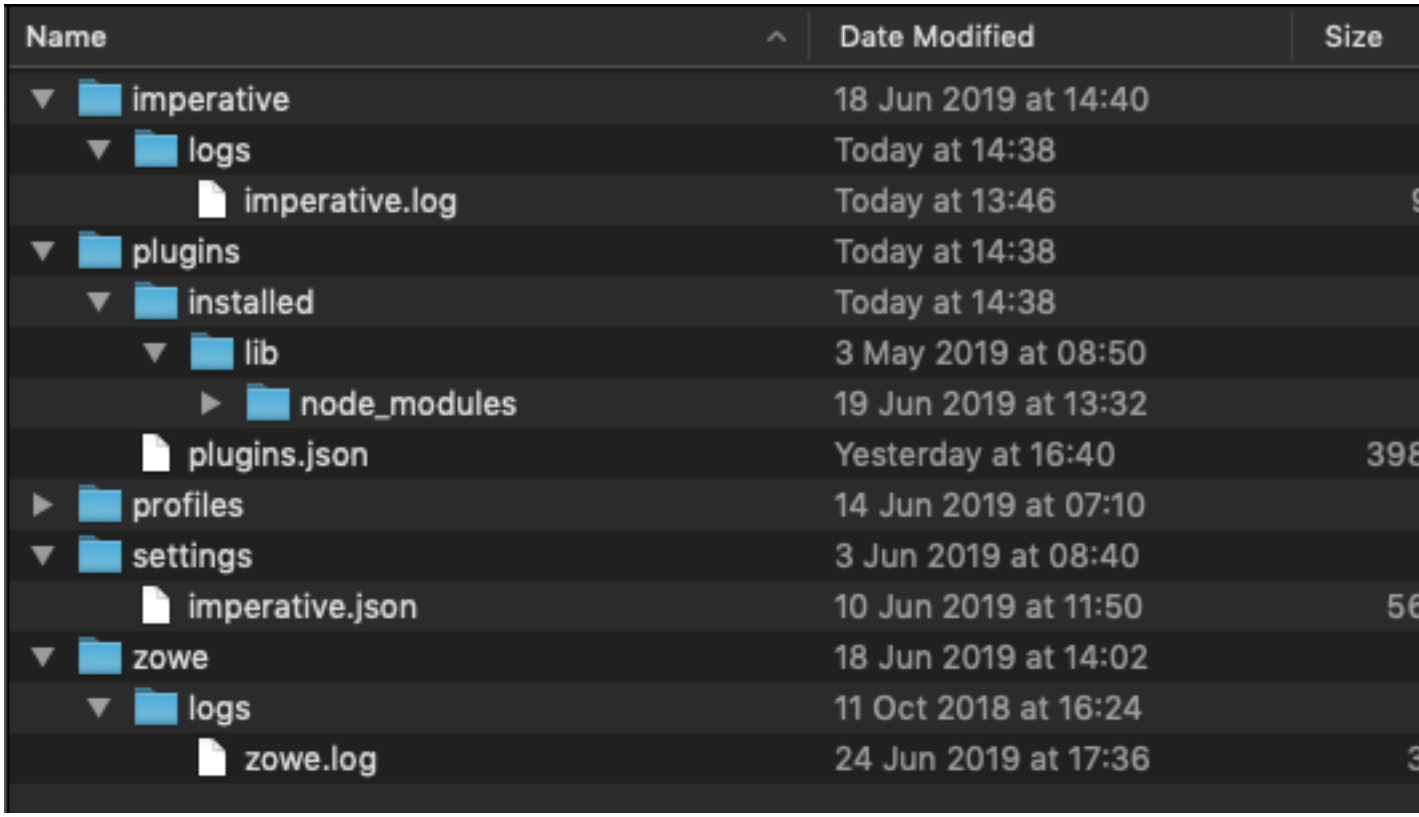
You can set the location on your computer for the Zowe CLI home directory, which contains log files, profiles, and plug-ins for the product.

Tip! The default `.zowe` folder is created when you issue your first Zowe CLI command. If you change the location of the folder, you must reinstall plug-ins and recreate or move profiles and log files that you want to retain. In some cases, you might want to maintain a different set of profiles in multiple folders, then switch between them using the environment variable.

Environment Variable	Description	Values	Default
`ZOWE_CLI_HOME`	Zowe CLI home directory location	Any valid path on your computer	C:\Users\\<username\>\\.zowe

The values for these variables can be **echo**ed.

Home directory structure



Location of logs

There are two sets of logs to be aware of:

- Imperative CLI Framework log, which generally contains installation and configuration information.
- Zowe CLI log, which contains information about interaction between CLI and the server endpoints.

Analyze these logs for any information relevant to your issue.

Profile configuration

The `profiles` folder stores connnection information.

Important\! The profile directory might contain "sensitive" information, such as your mainframe password. You should obfuscate any sensitive references before providing configuration files.

Node.js and npm

Zowe CLI should be compatible with Node.js v8 and greater.

To gather Node.js and npm versions, use the following:

```
node --version
npm --version
```

npm configuration

If you are having trouble installing Zowe CLI from an npm registry, gather your npm configuration to help identify issues with registry settings, global install paths, proxy settings, etc...

```
npm config ls -l
```

npm log files

In case of errors, npm creates log files in the `npm_cache\_logs` location. To get the `npm_cache` location for a specific OS, run the following command:

```
npm config get cache
```

By default, npm keeps only 10 log files, but sometimes more are needed. Increase the log count by issuing the following command:

```
npm config set logs-max 50
```

This command increases the log count to 50, so that more log files will be stored on the system. Now you can run tests multiple times and not lose the log files. The logs can be passed to Support for analysis.

As the log files are created only when an npm conmmand fails, but you are interested to see what is executed, you can increase the log level of npm. Issue the following command:

```
npm config set loglevel verbose
```

- With this change, you can see all actions taken by npm on the stdout. If the command is successful, it still does not generate a log file.
- The available log levels are: "silent", "error", "warn", "notice", "http", "timing", "info", "verbose", "silly", and "notice". "Notice" is the default.
- Alternatively, you can pass `--loglevel verbose` on the command line, but this only works with npm related commands. By setting log level in the config, it also works when you issue some `zowe` commands that use npm (for example, `zowe plugins install @zowe/cics`).

z/OSMF troubleshooting

The core command groups use the z/OSMF REST APIs which can experience any number of problems.

If you encounter HTTP 500 errors with the CLI, consider gathering the following information:

1. The IZU* (IZUSVR and IZUANG) joblogs (z/OSMF server)
2. z/OSMF USS logs (default location: /global/zosmf/data/logs - but may change depending on installation)

If you encounter HTTP 401 errors with the CLI, consider gathering the following information:

1. Any security violations for the TSO user in SYSLOG

Alternate methods

At times, it may be beneficial to test z/OSMF outside of the CLI. You can use the CLI tool `curl` or a REST tool such as "Postman" to isolate areas where the problem might be occurring (CLI configuration, server-side, etc.).

Example `curl` command to `GET /zosmf/info`:

```
curl -k -H "Accept: application/json" -H "X-CSRF-ZOSMF-HEADER: true"
 "https://zosmf.hostname.net:443/zosmf/info"
```

Known Zowe CLI issues

The following topics contain information that can help you troubleshoot problems when you encounter unexpected behavior installing and using Zowe™ CLI.

Command not found message displays when issuing `npm install` commands

Valid on all supported platforms

Symptom:

When you issue NPM commands to install the CLI, the message *command not found* displays. The message indicates that Node.js and NPM are not installed on your computer, or that PATH does not contain the correct path to the NodeJS folder.

Solution:

To correct this behavior, verify the following:

- Node.js and NPM are installed.
- PATH contains the correct path to the NodeJS folder.

More Information: System requirements on page 36

`npm install -g` Command Fails Due to an EPERM Error

Valid on Windows

Symptom:

This behavior is due to a problem with Node Package Manager (npm). There is an open issue on the npm GitHub repository to fix the defect.

Solution:

If you encounter this problem, some users report that repeatedly attempting to install Zowe CLI yields success. Some users also report success using the following workarounds:

- Issue the `npm cache clean` command.
- Uninstall and reinstall Zowe CLI. For more information, see Installing Zowe CLI on page 104.
- `Add the --no-optional` flag to the end of the `npm install` command.

Sudo syntax required to complete some installations

Valid on Linux and macOS

Symptom:

The installation fails on Linux or macOS.

Solution:

Depending on how you configured Node.js on Linux or macOS, you might need to add the prefix `sudo` before the `npm install -g` command or the `npm uninstall -g` command. This step gives Node.js write access to the installation directory.

`npm install -g` command fails due to `npm ERR! Cannot read property 'pause' of undefined` error

Valid on Windows or Linux

Symptom:

You receive the error message `npm ERR! Cannot read property 'pause' of undefined` when you attempt to install the product.

Solution:

This behavior is due to a problem with Node Package Manager (npm). If you encounter this problem, revert to a previous version of npm that does not contain this defect. To revert to a previous version of npm, issue the following command:

```
npm install npm@5.3.0 -g
```

Node.js commands do not respond as expected

Valid on Windows or Linux

Symptom:

You attempt to issue node.js commands and you do not receive the expected output.

Solution:

There might be a program that is named *node* on your path. The Node.js installer automatically adds a program that is named *node* to your path. When there are pre-existing programs that are named *node* on your computer, the program that appears first in the path is used. To correct this behavior, change the order of the programs in the path so that Node.js appears first.

Installation fails on Oracle Linux 6

Valid on Oracle Linux 6

Symptom:

You receive error messages when you attempt to install the product on an Oracle Linux 6 operating system.

Solution:

Install the product on Oracle Linux 7 or another Linux or Windows OS. Zowe CLI is not compatible with Oracle Linux 6.

Raising a CLI issue on GitHub

When necessary, you can raise GitHub issues against the Zowe™ CLI repository here. It is suggested that you use either the bug or enhancement template.

Raising a bug report

Please provide as much of the information listed on Troubleshooting Zowe CLI on page 308 as is reasonable. Anyone working on the issue might need to request this and other information if it is not supplied initially. A description of the error and how it can be reproduced is the most important information.

Raising an enhancement report

Enhancement reports are just as important to the Zowe project as bug reports. Enhancement reports should be clear and detailed requirements for a potential enhancement.

Troubleshooting Zowe through Zowe Open Community

To help Zowe™ Open Community effectively troubleshoot Zowe, we introduce a shell script that captures diagnostics data that is required for successful troubleshooting. By running the shell script on your z/OS environment, you receive a set of output files, which contain all relevant diagnostics data necessary to start a troubleshooting process. You can find the `zowe-support.sh` script in the `ZOWEDIR/scripts` folder with the rest of Zowe scripts. The script captures the following data:

- Started task output
  - Zowe server started task
  - Zowe Cross Memory started task (STC)
    - Zowe CLI or REXX (TSO output command, STATUS, capture all)
- Zowe Install log
- Scripts that are called from `run-zowe.sh`
- Versions:
  - manifest.json
  - z/OS version
  - Java version
  - Node version

- Additional logs
  - Zowe app server
  - zLUX app server
- Process list with CPU info with the following data points:
  - Running command and all arguments of the command
  - Real time that has elapsed since the process started
  - Job name
  - Process ID as a decimal number
  - Parent process ID as a decimal number
  - Processor time that the process used
  - Process user ID (in a form of user name if possible, or as a decimal user ID if not possible)

Contact Zowe Open Community to Troubleshoot Zowe

Contact Zowe Open Community to address and troubleshoot a Zowe issue.

Follow these steps:

1. Contact Open Zowe Community in Slack to address your issues.
2. Get instructions from the Community on whether you need to run the script that collects diagnostics data. If you do not need to run the script, the Community will proceed with troubleshooting.
3. If the Community instructs you to run the `zowe-support.sh` script, issue the following commands:

   ```
   cd $ZOWE_ROOT_DIR/scripts
   ./zowe-support.sh
   ```

4. Send the .pax.Z output file to Community members for further troubleshooting.

Community members will help you troubleshoot an issue.