

Zowe Documentation

Version 1.19.1 LTS

Contents

Chapter 1: Getting Started.....	7
Zowe overview.....	8
Zowe Demo Video.....	8
Component Overview.....	8
Zowe Third-Party Software Requirements and Bill of Materials.....	13
Zowe architecture.....	13
Zowe architecture when using Docker image.....	14
Release notes.....	20
Version 1.19.1 LTS (February 2021).....	21
Version 1.19.0 LTS (February 2021).....	22
Version 1.18.0 LTS (January 2021).....	24
Version 1.17.0 LTS (November 2020).....	25
Version 1.16.0 LTS (October 2020).....	27
Version 1.15.0 LTS (September 2020).....	30
Version 1.14.0 LTS (August 2020).....	33
Version 1.13.0 LTS (July 2020).....	36
Version 1.12.0 LTS (June 2020).....	38
Version 1.11.0 LTS (May 2020).....	41
Version 1.10.0 LTS (April 2020).....	43
Version 1.9.0 LTS (February 2020).....	44
Version 1.8.1 (February 2020).....	47
Version 1.8.0 (February 2020).....	47
Version 1.7.1 (December 2019).....	49
Version 1.7.0 (November 2019).....	49
Version 1.6.0 (October 2019).....	51
Version 1.5.0 (September 2019).....	51
Zowe SMP/E Alpha (August 2019).....	53
Version 1.4.0 (August 2019).....	53
Version 1.3.0 (June 2019).....	55
Version 1.2.0 (May 2019).....	56
Version 1.1.0 (April 2019).....	58
Version 1.0.1 (March 2019).....	59
Version 1.0.0 (February 2019).....	60
Zowe CLI quick start.....	61
Installing.....	62
Issuing your first commands.....	62
Using profiles.....	62
Writing scripts.....	63
Next Steps.....	63
Frequently Asked Questions.....	64
Zowe FAQ.....	64
Zowe CLI FAQ.....	66
Zowe Explorer FAQ.....	67
Zowe resources.....	68
Blogs.....	68
Videos, webinars.....	68
Community.....	68

Chapter 2: User Guide.....	69
Planning and preparing the installation.....	70
Introduction.....	70
System requirements.....	72
Installing Node.js on z/OS.....	76
Configuring z/OSMF.....	77
Configuring z/OSMF Lite (for non-production use).....	80
UNIX System Services considerations for Zowe.....	97
Installing Zowe z/OS components.....	99
z/OS Installation Roadmap.....	99
Installing Zowe runtime from a convenience build.....	102
Installing Zowe SMP/E.....	108
Installing Zowe SMP/E build with z/OSMF workflow.....	125
API Mediation Layer as a standalone component.....	126
Configuring the z/OS system for Zowe.....	126
Configuring Zowe certificates.....	136
Configuring Zowe certificates in UNIX files.....	139
Configuring Zowe certificates in a key ring.....	144
Installing and configuring the Zowe cross memory server (ZWESISTC).....	146
Creating and configuring the Zowe instance directory.....	150
Installing and starting the Zowe started task (ZWEVSTC).....	156
Configure Zowe with z/OSMF Workflows.....	157
Verifying Zowe installation on z/OS.....	161
Zowe Auxiliary Address space.....	162
Upgrading the z/OS system for Zowe.....	163
Stopping the ZWEVSTC PROC.....	165
Uninstalling Zowe from z/OS.....	165
Installing Zowe Docker Bundle.....	166
Docker Installation Roadmap (Technical Preview).....	166
Installing Zowe runtime Docker Image (Technical Preview).....	169
Configuring Zowe runtime Docker Container (Technical Preview).....	170
Installing Zowe CLI.....	173
Installing Zowe CLI.....	173
Install CLI from Online Registry Via Proxy.....	175
Updating Zowe CLI.....	177
Uninstalling Zowe CLI.....	178
Advanced Zowe configuration.....	179
Configuring Zowe Application Framework.....	179
Configuring Zowe CLI.....	195
Configuring the Zowe APIs.....	195
Advanced Gateway features configuration.....	196
API Mediation Layer as a standalone component.....	199
API Gateway configuration parameters.....	200
Using Zowe.....	203
Getting started tutorial.....	203
Using the Zowe Desktop.....	222
Using the Editor.....	228
Using API Catalog.....	229
Using Zowe SDKs.....	237
Zowe CLI extensions and plug-ins.....	239
Extending Zowe CLI.....	239
Software requirements for Zowe CLI plug-ins.....	239
Installing Zowe CLI plug-ins.....	240
IBM® CICS® Plug-in for Zowe CLI.....	244

IBM® Db2® Database Plug-in for Zowe CLI.....	245
IBM® z/OS FTP Plug-in for Zowe CLI.....	248
IBM® IMS™ Plug-in for Zowe CLI.....	249
IBM® MQ Plug-in for Zowe CLI.....	250
Secure Credential Store Plug-in for Zowe CLI.....	251
Zowe Explorer.....	253
Installing Zowe Explorer.....	253
Zowe Explorer Profiles.....	256
Using Zowe Explorer.....	260
Extending Zowe Explorer.....	278
Chapter 3: Extending.....	279
Overview.....	280
Extending Zowe.....	280
Packaging z/OS extensions.....	281
Install and configure Zowe server component.....	284
Zowe server component runtime lifecycle.....	287
Developing for Zowe CLI.....	288
Developing for Zowe CLI.....	288
Setting up your development environment.....	290
Installing the sample plug-in.....	291
Extending a plug-in.....	293
Developing a new plug-in.....	296
Implementing profiles in a plug-in.....	301
Developing for Zowe API Mediation Layer.....	302
Onboarding Overview.....	302
API Mediation Layer onboarding configuration.....	306
Onboarding a service with the Zowe API Meditation Layer without an onboarding enabler.....	311
Onboard a REST API without code changes required.....	317
API Mediation Layer Message Service Component.....	328
Zowe API Mediation Layer Security.....	331
API Mediation Layer routing.....	346
Enabling PassTicket creation for API Services that Accept PassTickets.....	348
Developing for Zowe Application Framework.....	351
Overview.....	351
Plug-ins definition and structure.....	352
Building plugin apps.....	356
Installing Plugins.....	357
Embedding plugins.....	358
Dataservices.....	360
Authentication API.....	366
Internationalizing applications.....	368
Zowe Desktop and window management.....	372
Configuration Dataservice.....	375
URI Broker.....	381
Application-to-application communication.....	382
Configuring IFrame communication.....	387
Error reporting UI.....	388
Logging utility.....	390
Using Conda to make and manage packages of Application Framework Plugins.....	393
Developing for Zowe SDKs.....	396
Zowe Conformance Program.....	397
Introduction.....	397
How to participate.....	397
How to suggest updates to the Zowe conformance program.....	397

Chapter 4: Troubleshooting..... 399

Overview.....	400
Troubleshooting.....	400
Understanding the Zowe release.....	400
Capturing diagnostics to assist problem determination.....	401
Verify Zowe runtime directory.....	403
Troubleshooting installation and startup of Zowe z/OS components.....	407
How to check if ZWESVSTC startup is successful.....	407
Unable to launch Zowe with.....	410
Unable to create BPXAS instances.....	410
Errors caused when running the Zowe desktop with node 8.16.1.....	411
Cannot start Zowe and UNIX commands not found with FSUM7351.....	411
Various warnings show when connecting Zowe with another domain.....	412
Zowe API Mediation Layer.....	413
Troubleshooting API ML.....	413
Error Message Codes.....	421
Zowe Application Framework.....	439
Troubleshooting Zowe Application Framework.....	439
Gathering information to troubleshoot Zowe Application Framework.....	445
Raising a Zowe Application Framework issue on GitHub.....	447
Troubleshooting z/OS Services.....	448
z/OSMF JVM cache corruption.....	448
Unable to generate unique CeaTso APPTAG.....	449
z/OS Services are unavailable.....	449
Zowe CLI.....	451
Troubleshooting Zowe CLI.....	451
Gathering information to troubleshoot Zowe CLI.....	451
z/OSMF troubleshooting.....	454
Known Zowe CLI issues.....	454
Raising a CLI issue on GitHub.....	456
Zowe Explorer.....	456
Troubleshooting Zowe Explorer.....	456
Known Zowe Explorer issues.....	457
Raising a Zowe Explorer issue on GitHub.....	457

Chapter

1

Getting Started

Topics:

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

Zowe overview

Zowe™ is an open source software framework that allows mainframe development and operation teams to securely manage, control, script, and develop on the mainframe. It was 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. Like Mac or Windows, Zowe 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 is a project within the Open Mainframe Project.

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 Terminal 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 applications to run within the desktop. See [Overview](#) on page 351. 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 223.

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 you 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 you use common tools such as Integrated Development Environments (IDEs), shell commands, bash scripts, and build tools for mainframe development. Though its ecosystem of plug-ins, you can automate actions on systems such as IBM Db2, IBM CICS, and more. It provides a set of utilities and services for users 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 [Installing Zowe Explorer](#) on page 253.
- 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 70.

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 239.

More Information:

- [System requirements](#) on page 72
- [Installing Zowe CLI](#) on page 173

Zowe Client Software Development Kits (SDKs)

The Zowe Client SDKs consist of programmatic APIs that you can use to build client applications or scripts that interact with z/OS. The following SDKs are available:

- Zowe Node.js Client SDK
- Zowe Python Client SDK

For more information, see [Using Zowe SDKs](#) on page 237.

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

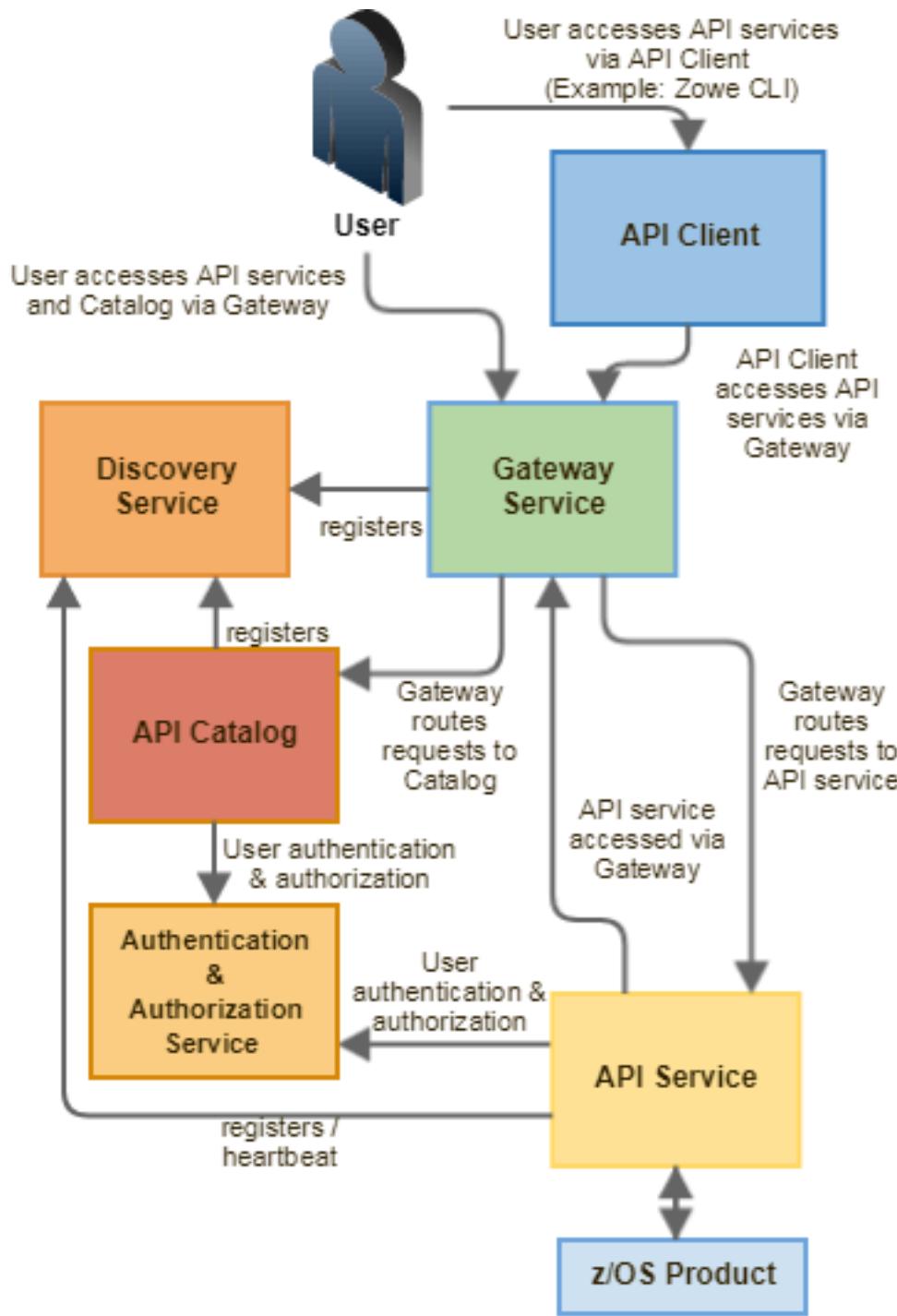
The API Mediation Layer provides a single point of access for mainframe service REST APIs. The layer offers enterprise, cloud-like features such as high-availability, scalability, dynamic API discovery, consistent security, a single sign-on experience, and documentation. The API Mediation Layer facilitates 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 heightened 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 302
- [Onboard an existing Spring Boot REST API service using Zowe API Mediation Layer](#)
- [Using API Catalog](#) on page 229

To learn more about the architecture of Zowe, see [Zowe architecture](#) on page 13.

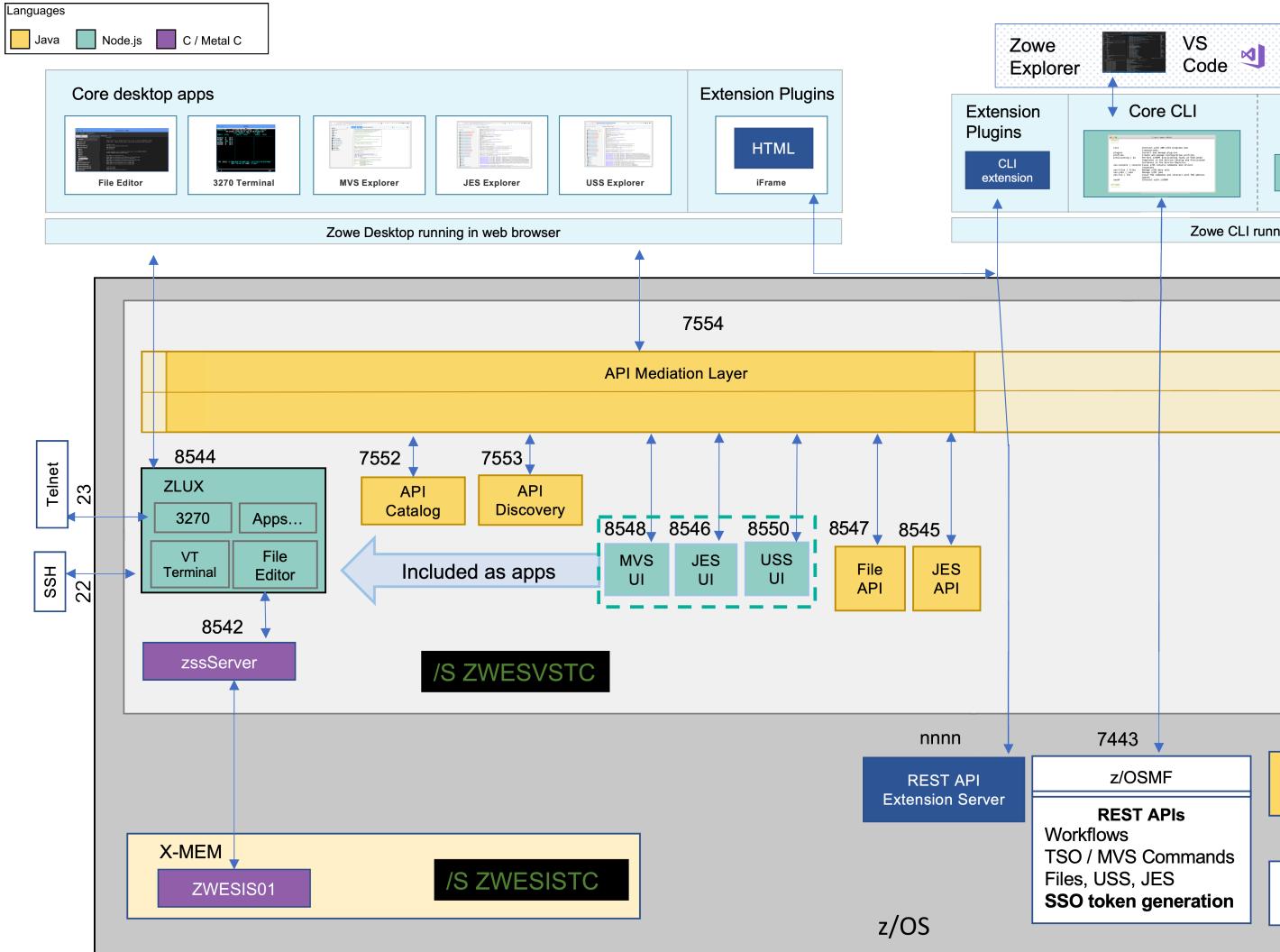
Zowe Third-Party Software Requirements and Bill of Materials

- [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.



The diagram shows the default port numbers that are used by Zowe. These are dependent on each instance of Zowe and are held in the Zowe instance directory configuration file `instance.env`. For more information, see [Creating and configuring the Zowe instance directory](#) on page 150.

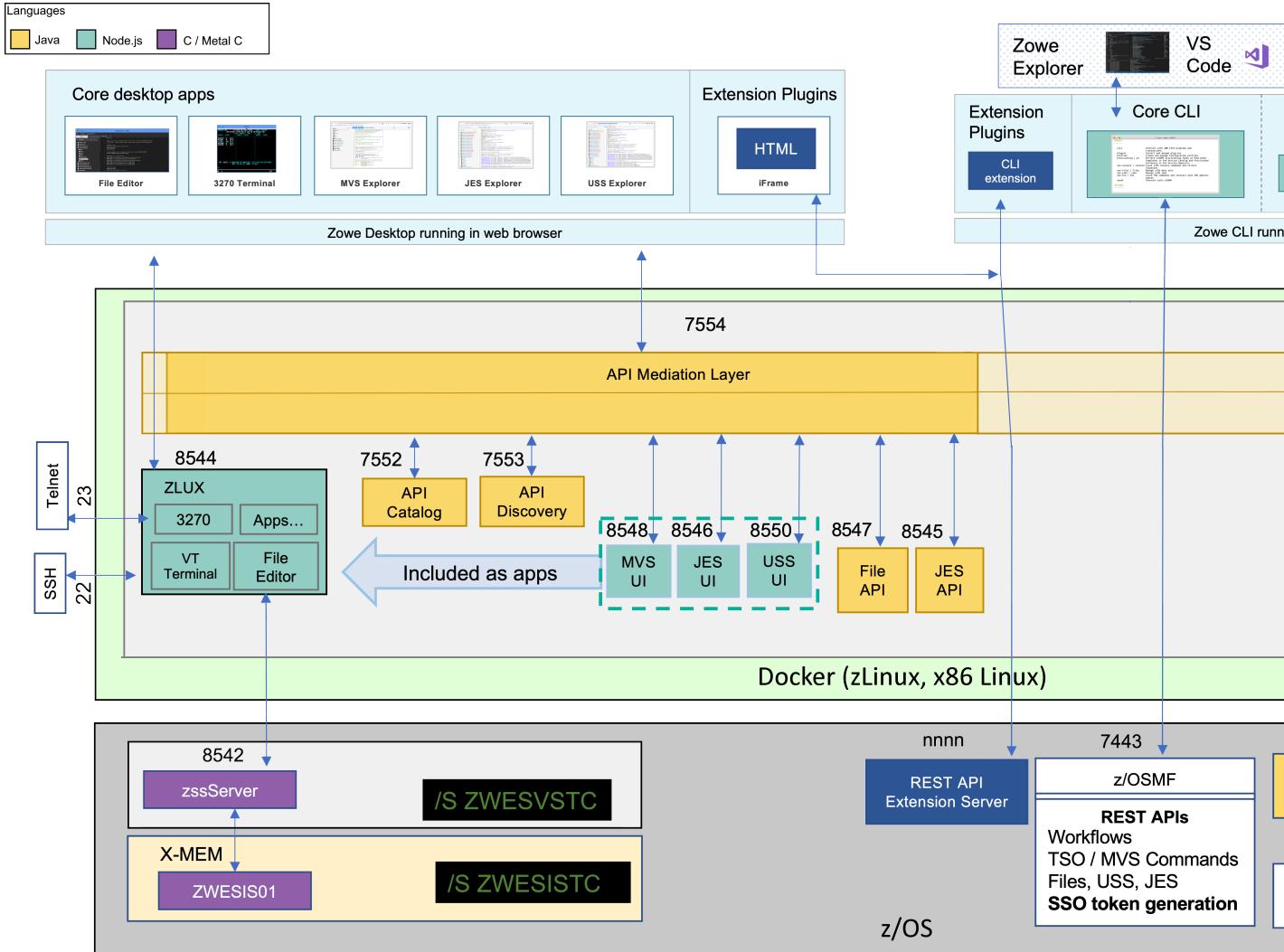
Zowe components can be categorized by location: server or client. While the client is always an end user tool such as a PC, browser or mobile device, the server components can be further categorized by what machine they run on.

Zowe server components can be installed and run entirely on z/OS, but a subset of the components can alternatively run on Linux or z/Linux via Docker. While on z/OS, many of these components run under Unix System Services (USS). The ones that do not run under USS must remain on z/OS when using Docker in order to provide connectivity to the mainframe.

Zowe architecture when using Docker image

The Zowe Docker build is a technical preview.

The following diagram depicts the difference in locations of Zowe components when using Docker as opposed to running all components on z/OS.

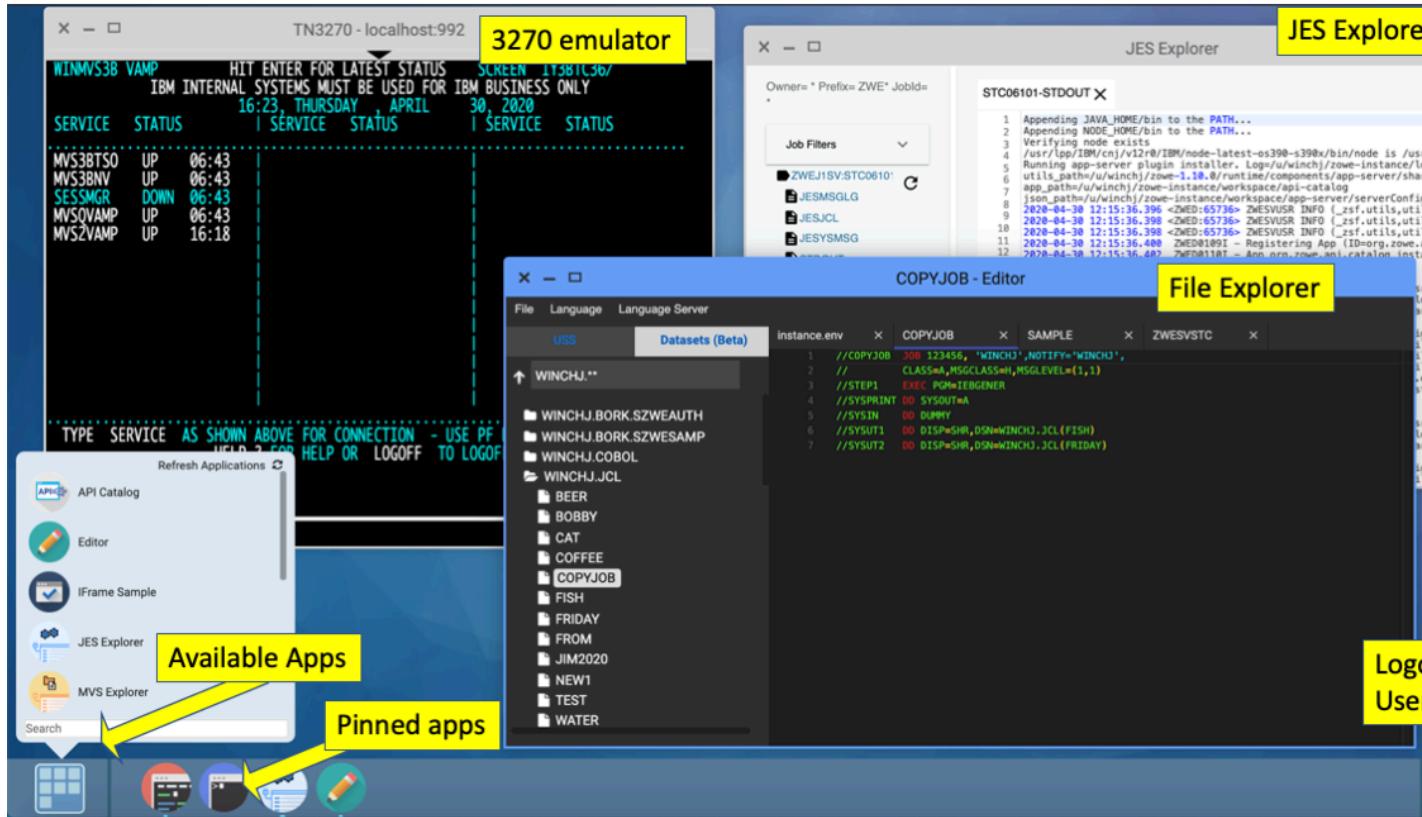


The components on z/OS run under the Zowe started task **ZWESVSTC**, which has its own user ID **ZWESVUSR** and includes a number of servers each with their own address space. The **ZWESVSTC** started task has a **STDOUT** file that includes log and trace information for its servers. Sever error messages are written to **STDERR**. For problem determination, see [Troubleshooting](#) on page 400.

When Docker is used, server components not running on z/OS instead run in a Linux environment provided via the Docker container technology. The servers run as processes within the container which log to **STDOUT** and **STDERR** of that container, with some components also write to the Zowe instance's log directory.

App Server

The App Server is a node.js server that is responsible for the Zowe Application Framework. It provides the Zowe desktop, which is accessible through a web browser via port 8544. The Zowe desktop includes a number of applications that run inside the Application Framework such as a 3270 emulator and a File Editor.



The App Server server logs are written to <INSTANCE_DIR>/logs/appServer-yyyy-mm-dd-hh-mm.log. The Application Framework provides REST APIs for its services that are included on the API catalog tile Zowe Application Framework that can be viewed at https://<ZOWE_HOST_IP>:7554/ui/v1/apicatalog/#/tile/ZLUX/zlux.

ZSS

The Zowe desktop delegates a number of its services to the ZSS server which it accesses through the http port 8542. ZSS is written in C and has native calls to z/OS to provide its services. ZSS logs are written STDOUT and STDERR for capture into job logs, but also as a file into <INSTANCE_DIR>/logs/zssServer-yyyy-mm-dd-hh-mm.log.

API Gateway

The API Gateway is a proxy server that routes requests from clients on its northbound edge, such as web browsers or the Zowe command line interface, to servers on its southbound edge that are able to provide data to serve the request. It is also responsible for generating the authentication token used to provide single sign-on (SSO) functionality. The API Gateway homepage is https://<ZOWE_HOST_IP>:7554, that after authentication allows you to navigate to the API Catalog.



API Mediation Layer

API Mediation Layer

- ✓ The API Catalog is running
- ✓ The Discovery Service is running
- ✓ The Authentication service is running

API Catalog

The API Catalog provides a list of the API services that have registered themselves as catalog tiles. These allow you to view the available APIs from Zowe's southbound servers as well as test REST API calls.

The screenshot shows the API Catalog interface. At the top, there's a blue header bar with the "API Catalog" logo and a search bar labeled "Search for APIs". Below the header, the title "Available API services" is displayed. There are five service cards arranged in two rows:

- API Mediation Layer API**: Described as the API Mediation Layer for z/OS internal API services. It provides a single point of access to mainframe REST APIs and offers enterprise cloud-like features. Status: All services are running.
- z/OS Datasets and Unix Files services**: IBM z/OS Datasets and Unix Files REST services. Status: All services are running.
- z/OS Jobs services**: IBM z/OS Jobs REST services. Status: All services are running.

- z/OSMF services**: IBM z/OS Management Facility REST services. Status: All services are running.
- Zowe Application Server**: The Proxy Server is an HTTP, HTTPS, and Websocket server built upon NodeJS and ExpressJS. This serves static content via "Plugins", and is extensible by REST and Websocket "Data...". Status: All services are running.

A decorative background graphic of a circuit board or network diagram is visible behind the service cards.

API Discovery

The API Discovery server acts as the registration service broker between the API Gateway and its southbound servers. It can be accessed through the URL `https://<ZOWE_HOST_IP>:7552`. You can view a list of registered API services on the API discovery homepage.

 **spring Eureka**

HOME LAST 1000 SINCE STARTUP

System Status

Environment	test	Current time	2020-04-30T14:50:51 +0000
Data center	default	Uptime	02:35
		Lease expiration enabled	true
		Renews threshold	18
		Renews (last min)	44

DS Replicas

winmvs3b.hursley.ibm.com

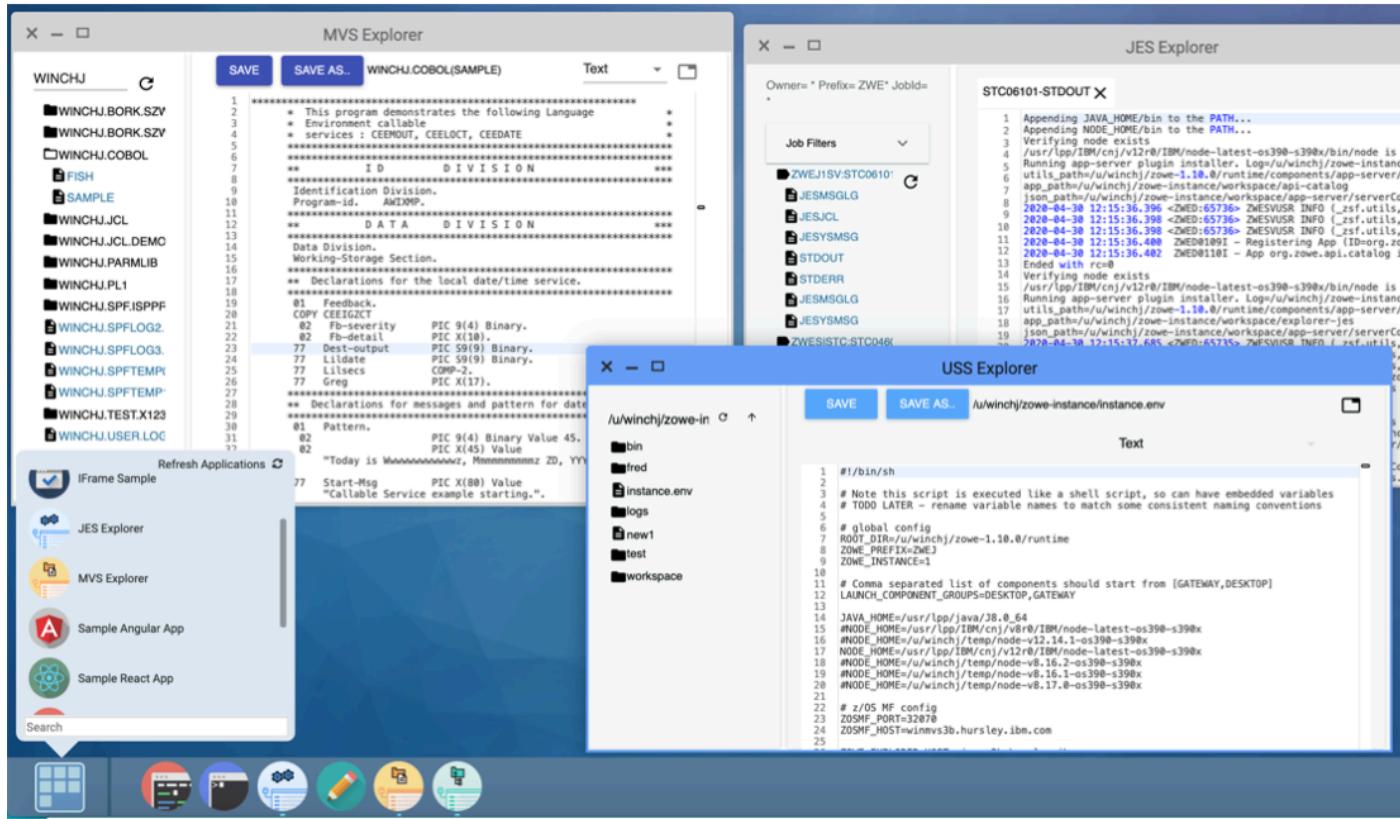
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
APICATALOG	n/a (1)	(1)	UP (1) - winmvs3b.hursley.ibm.com:apicatalog:26500
DATASETS	n/a (1)	(1)	UP (1) - STATIC-winmvs3b.hursley.ibm.com:datasets:26504
DISCOVERY	n/a (1)	(1)	UP (1) - winmvs3b.hursley.ibm.com:discovery:26501
EXPLORER-JES	n/a (1)	(1)	UP (1) - STATIC-winmvs3b.hursley.ibm.com:explorer-jes:26505
EXPLORER-MVS	n/a (1)	(1)	UP (1) - STATIC-winmvs3b.hursley.ibm.com:explorer-mvs:26506
EXPLORER-USS	n/a (1)	(1)	UP (1) - STATIC-winmvs3b.hursley.ibm.com:explorer-uss:26507
GATEWAY	n/a (1)	(1)	UP (1) - winmvs3b.hursley.ibm.com:gateway:26502
JOBSS	n/a (1)	(1)	UP (1) - STATIC-winmvs3b.hursley.ibm.com:jobs:26503
UNIXFILES	n/a (1)	(1)	UP (1) - STATIC-winmvs3b.hursley.ibm.com:unixfiles:26504
ZLUX	n/a (1)	(1)	UP (1) - localhost:zlux:26508
ZOSMF	n/a (1)	(1)	UP (1) - STATIC-winmvs3b.hursley.ibm.com:zosmf:32070

General Info

MVS, JES, and USS UI

Zowe provides a number of rich GUI web applications for working with z/OS. This includes the MVS Explorer for data sets, the JES Explorer for jobs, and the USS Explorer for the Unix File System. You can access them through the Zowe desktop.



File API and JES API

The File API server provides a set of REST APIs for working with z/OS data sets and Unix files. These APIs are used by the MVS and USS Explorer apps.

The JES API server provides a set of REST APIs for working with JES. These APIs are used by the JES Explorer application.

Both the File API and JES API servers are registered as tiles on the API catalog, so you can view the Swagger definition and test API requests and responses.

Cross memory server

The Cross memory server is a low-level privileged server for managing mainframe data securely. For security reasons, it is not an HTTP server. Instead, it has a trust relationship with ZSS. Other Zowe components can work through ZSS in order to handle z/OS data that would otherwise be unavailable or insecure to access from higher-level languages and software.

Unlike all of the servers described above which run under the ZWESVSTC started task as address spaces for USS processes, the cross memory server has its own separate started task ZWESISTC and its own user ID ZWESIUSR that runs the program ZWESIS01.

Release notes

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

Zowe Version 1.19 and earlier releases include the following enhancements, release by release.

- [Version 1.19.1 LTS \(February 2021\)](#)
- [Version 1.19.0 LTS \(February 2021\)](#)

- Version 1.18.0 LTS (January 2021)
- Version 1.17.0 LTS (November 2020)
- Version 1.16.0 LTS (October 2020)
- Version 1.15.0 LTS (September 2020)
- Version 1.14.0 LTS (August 2020)
- Version 1.13.0 LTS (July 2020)
- Version 1.12.0 LTS (June 2020)
- Version 1.11.0 LTS (May 2020)
- Version 1.10.0 LTS (April 2020)
- Version 1.9.0 LTS (February 2020)
- Version 1.8.1 (February 2020)
- 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 1.19.1 LTS (February 2021)

Notable changes

SMPE PTF

With the continuous optimization of the Zowe build size, now the Zowe release can fit in one PTF. This single PTF UO01969 will supersede PTF UO01967 and UO01968 from the v1.19.0 release.

New features and enhancements

Zowe CLI

The following enhancements were added to the **core CLI**:

- Updated Imperative version to support npm@7. This fixes an error when installing plugins.

The following enhancements were added in the **Imperative CLI Framework**:

- Fixed plugin install commands which were broken in npm@7. [#457](#)

Bug fixes

SMPE PTF

- Fixed the missing HOLD data about extra steps the system programmer should take after applying the PTF. It explains new configuration options that are introduced in `instance.env` and also a reconfiguration step where `zowe-configure-instance.sh` is mandatory.

Zowe CLI

The following bugs were fixed in the **Imperative CLI Framework**:

- Fixed incorrect formatting of code blocks in web help. [#535](#)

Version 1.19.0 LTS (February 2021)

Notable changes

Package manifest and component installer

Each Zowe extension and each core component can now use a manifest file to describe itself. The manifest file defines the name and purpose of the component. It also provides information about how this component should be installed, configured, started and tested. For more information, see [Packaging z/OS extensions](#) on page 281.

Two tools, `zowe-install-component.sh` and `zowe-configure-component.sh`, are introduced in this release for technical preview. The `zowe-install-component.sh` helps you install any Zowe server component (extension). Zowe core components are also installed with this tool. The `zowe-configure-component.sh` tool helps you configure an installed Zowe server component (extension) for a Zowe instance. Zowe core components are also configured with this tool. In order to be compatible with the tools, it is recommended that the components follow [Zowe server component package format](#) on page 281.

X.509 client certificate authentication support for API Mediation Layer (Technical Preview)

This feature is released for technical preview in Zowe 1.19. Previously, users were required to provide credentials (usually basic authentication) to make a login call against the API Gateway. From release 1.19 you can now use the x509 client certificate for calls to authenticate in the API ML, whereby information from the certificate verifies the user's identity through SAF and then returns a proper JWT.

If you would like to offer feedback about using client certificate authentication, please create an issue against the `Zowe api-layer` repository.

Standalone run of Zowe API Mediation Layer

You can now start the API Mediation Layer independently of other Zowe components. This allows you to minimize the resources used when you use Zowe as a Devops tool instead of a Virtual Desktop tool.

New features and enhancements

Zowe API Mediation Layer

- The connection limit of the Gateway has been configured to support multiple long-running requests by service. [#843](#)
- The size of API Mediation Layer has been reduced to fit within 150MB. [#909](#)
- You can now configure whether or not the Catalog appears on the Gateway homepage [#727](#)
- Connection limits have been enhanced to improve latency times when making requests through the API ML. This feature also enables concurrent requests. [#987](#)
- The connection limit log messages have been enhanced. New messages indicate when too many connections occur. [#987](#)
- The `/api/v1/gateway/services/{serviceId}` endpoint has been added which provides information about a service in API ML for API clients. You can now view information to choose the applicable available API service without having a trusted service certificate. Proper SAF authorization is required. [#873](#)
- The size limitation in the InMemory cache for proper handling is now supported when size limitations are reached. [#998](#)
- The 'Remove Oldest' eviction mechanism for Caching Service has been implemented to limit the volume of data in the cache. [#998](#)
- CORS origins per service has been configured so that onboarded services can request that CORS behavior for a route be delegated to the API Mediation Layer. [#997](#)
- The 'Reject eviction' strategy to the Caching Service has been implemented to limit the volume of data in the cache. [#998](#)
- Debug logging to x.509 Client certificate authentication classes has been added. This feature enables users to determine the cause of system problems during client certificate authentication setup.

Zowe App Server

- Dispatcher actions have been added to the iFrame adapter. [#302](#)

- Support has been added to a new destination property for iFrame pluginDefinition.json. iFrame with the new destination property will now make requests to /web/iFrame. [#296](#)
- The compression-webpack-plugin has been updated from 3.1.0 to 4.0.0 [#304](#)
- Support for a new destination property to iFrame pluginDefinition.json has been added, as well as a new double iFrame default template. [#257](#)
- Axios has been updated from 0.19.2 to 0.21.1 in /test/webapp/websocket [#259](#)
- The option to refresh file content has been added to the Editor. [#185](#)
- Refresh buttons have been added to USS and MVS. [#108](#)
- Additional keybindings and other improvements have been added to the Editor. [#182](#)
 - You can now move between open file tabs by using the following hotkeys: **Alt + (PAGEUP or <) and Alt + (PAGEDOWN or >)**
 - After closing a tab, or multiple tabs, you can now undo the close by using the following hotkeys: **CTRL + ALT + T**
 - The search function hotkey has been changed from **ALT + S** to **ALT + S.**
 - You can now hide/show the File Tree by using the following hotkeys: **ALT + B.**
- Existing code highlighters have been reorganized in order to improve their readability. Additionally, a new code highlighter for the REXX language has been added. This new code highlighter detects files and datasets wherein the files should end with the .rexx prefix, but the datasets may contain the rex or exec qualifiers. [#181](#)

Zowe Explorer

- Updated Keytar and Jest dev deps for Node 14.

Zowe JES/MVS/USS Explorers

The following features and enhancements were added to the **JES Explorer**:

- Introduced the menu shortcuts and confirmation dialog before canceling or purging the job for JES explorer. [#235](#)
- Refactored JES packaging and installation scripts, and folder renames to accommodate new iframe capability in ZLUX. [#236](#)
- Added manifest for API ML and App Framework installation using new plugin installation process. [#234](#)

The following features and enhancements were added to the **MVS Explorer**:

- Refactored MVS packaging and installation scripts, and folder renames, to accommodate new iframe capability in ZLUX. [#164](#)
- Added manifest for API ML and App Framework installation using new plugin installation process. [#164](#)

Bug Fixes

Zowe API Mediation Layer

- API ID is not sent to Eureka in metadata by the Java enabler [#991](#)
- Fixed tcp connections that are stuck open. [#1009](#)

Zowe App Server

- In previous versions, sso-auth URL encoding that used the % sign would always return with authorization:false when using RACF. This issue has been resolved in this version. [#258 #27](#)
- Fixes a bug in the Editor that prevented the unsaved changes symbol from being displayed. [#185](#)
- Stopped event propagation in the Editor in order to resolve a conflict with Firefox. [#183](#)
- Fixes a bug in the Editor that would cause the Languages menu to disappear when closing all tabs, then clicking undo. [#182](#)

Zowe CLI

The following bug was fixed in the **core CLI**:

- Updated the Imperative version to fix a vulnerability.

The following bugs were fixed in the **Imperative CLI Framework**:

- Fixed vulnerabilities by updating marked [#515](#)
- Fixed an issue where `TypeError` has been raised by `Logger.getCallerFileAndLineTag()` when there was not filename for a stack frame. [#449](#)

Version 1.18.0 LTS (January 2021)

Notable changes

Zowe Docker build Technical Preview is available

The Zowe Docker build enables you to run a subset of the Zowe server-side components outside z/OS. The Docker build runs in combination with the convenience or SMP/E build. You can download the build directly via the .tar file, or as a cloud download from Docker Hub. Separate downloads exist for z/Linux ("s390x") and other Linux ("amd64" for intel & amd systems).

For more information, see [Docker Installation Roadmap \(Technical Preview\)](#) on page 166. To download the Docker build technical preview, visit [Zowe.org](#).

New features and enhancements

Zowe API Mediation Layer

- Version 1.18.0 introduces a feature allowing users to run the Zowe API Mediation Layer as a standalone component. After downloading and installing the current Zowe SMPE package, you can then configure and deploy only the Zowe API Mediation Layer without the other Zowe components. [#856](#)
- You can now configure more detailed logging outside of Spool. [#709](#)
- High Availability: The start script per API ML service has been componentized. You can now launch and restart API Mediation Layer components individually. [#862](#)
- High Availability: It is now possible to distinguish between internal and external traffic through port separation, whereby each port uses a unique certificate; one presenting an internal certificate, and the other an external certificate. [#910](#)
- API version is now automatically set to the version tab selected in the API Catalog so users can easily grab the Base Path. [#943](#)
- API Catalog versioning has been improved with the addition of the API differences tab. This feature enables you to compare versions of two APIs. [#923](#)

Zowe App Server

- The Zowe App Framework's "single app mode" is now based on code shared with the Desktop, allowing it to support the Desktop's notification API and app2app communication. [#67](#) [#292](#)
 - This is backward compatible with apps that have previously used single app mode.
 - In the case where app2app communication is used and spawns a second app, that app will spawn in a window but will not be able to be minimized due to single app mode having no Desktop, and therefore no way to restore a minimized window.
- ZSS plug-ins can now issue HTTP requests as HTTP clients, provided by a new library in zowe-common-c. [#179](#)

Zowe CLI

The following enhancements were added to the **core CLI**:

- Added a `--replace` option to the `zowe zos-files copy data-set` command. Use this option if you want to replace like-named members in the target data set. [#808](#)
- Improved a cryptic error message that was shown if the TSO address space failed to start for the `zowe zos-tso issue` command command. [#28](#)

The following enhancements were added to the **Imperative CLI Framework**:

- Added an `arrayAllowDuplicate` option to the `ICommandOptionDefinition` interface. By default, the option value is set to `true` and duplicate values are allowed in an array. Specify `false` if you want Imperative to throw an error for duplicate array values. [#437](#)
- Expose `trim` parameter from `wrap-ansi` within `TextUtils.wordWrap()`

The following enhancement was added to the **IBM Db2 Plug-in**:

- Added a help example for how to pass output values when calling a Db2 stored procedure.

The following enhancement was added to the **FTP Plug-in**:

- Move the reusable code from handlers to api folder.

Zowe JES/MVS/USS Explorers

The following features and enhancements were added to the **JES Explorer**:

- Added webdevSever proxy setting in webpack.config.js to enable https for local development.

Bug Fixes

Zowe API Mediation Layer

- ZaaSJwtService enhancement on JWT parsing and error handling. [#897](#)
- Upgrade dependencies for the Enablers. [#933](#)

Zowe App Server

- The zss server log verbosity seen when using the TN3270 desktop app has been reduced. [#188](#)
- Keep-alive parsing has been temporarily disabled to patch a memory leak. A permanent fix that will allow the use of keep-alive parsing is scheduled to be implemented in the next release. [#186](#)
- The warning messages on the zss server startup have been removed due to a shell syntax problem. [#238](#)
- In previous versions, static app2app recognizers would not be loaded from storage because they were treated as actions instead of as recognizers. This issue has been resolved in this release. [#297](#)

Zowe CLI

The following bugs were fixed in the **core CLI**:

- Removed "" text that appeared in some error messages. The proper text "Imperative API Error" is now displayed. [#836](#)
- Improved performance of `zowe zos-files list` commands when long lists are printed to console. [#861](#)
- Updated Imperative dependency version to one that does not contain a vulnerable dependency.

The following bug was fixed in the **Imperative CLI Framework**:

- Updated opener dependency due to command injection vulnerability on Windows - [GHSL-2020-145](#)

Version 1.17.0 LTS (November 2020)

Notable changes

z/OSMF workflow for configuring Cross Memory Server

You can now use the z/OSMF workflow to install, configure, and launch the cross memory server if you want to use the Zowe desktop. The z/OSMF workflow also lets you create APF-authorized load libraries that are required to install and configure the cross memory server. For more information, see [Configure Zowe Cross Memory Server](#) on page 158.

Zowe Client SDKs

A new Zowe incubation project - the Zowe Client SDKs (Software Development Kits) is now available for Node.js, Python, and Swift programming languages. You can leverage these SDKs to rapidly develop off-platform applications and automation.

For more information about the Node.js and Python SDKs, see [Using Zowe SDKs](#) on page 237. For more information about the Swift SDK, see the [Swift SDK Readme](#).

New features and enhancements

The following features and enhancements were added.

Zowe installation

- You can now start ZSS independent of the Zowe Application Framework server by specifying the `LAUNCH_COMPONENT_GROUP "ZSS"`. If `DESKTOP` is specified instead of `ZSS`, ZSS will still be included as a prerequisite to the Application Framework server. [#1632](#)
- Zowe instance configuration script (`zowe-configure-instance.sh`) can now skip checking for Node.js by passing in the `-s` flag since Node.js may not be needed if the components to be launched don't require it. [#1677](#)
- The `run-zowe.sh` script can also skip the checking for Node.js by setting the environment variable `SKIP_NODE=1` for the cases where the components to be launched don't require Node.js.
- Exported the `EXTERNAL_CERTIFICATE_AUTHORITIES` variable to the `zowe-certificates.env` file such that it may be used by the Application Framework server. [#1742](#)
- A new documentation chapter [Upgrading the z/OS system for Zowe](#) on page 163 has been included, that describes the steps to take when upgrading an existing Zowe installation.

Zowe API Mediation Layer

- Multiple versions of one API are now presented in the Catalog if configured to do so. Users can now switch between different versions within the Catalog to see differences in API documentation between versions. [#844](#)
- Setting `APIML_DEBUG_MODE_ENABLED` in `instance.env` is properly passed on to the all API ML services. [#901](#)

Zowe App Server

- ZSS no longer requires NodeJS for its `configure.sh` script.
- Added support for DER-encoded X.509 certificates.
- You are now able to change tags for all files in the directory excluding subdirectories. For example, `POST / unixfile/ctag/u/user/tmp?codeset=1047&type=text&recursive=false` should change tags only for files in `u/user/tmp` without changing tags for files in subdirectories. [#176](#)
- Multiple enhancements in the Editor for USS file and directory actions, including: [#84](#) [#102](#) [#93](#)
 - The ability to cut, copy, & paste files into a directory, such as the currently active directory.
 - Re-ordered context menu options.
 - Improved error messages by including more detail.
 - Added support to see a file's current tag and change it.
 - `chown` & `chmod` enhancement that pre-populates the owner and group fields when opening the ownership and properties dialogs. It also adds owner and group information to the file properties dialog.

Zowe CLI

The following enhancements were added to the **core CLI**:

- Zowe CLI was tested and confirmed to be compatible with Node.js v14.
- Published the programmatic interfaces in Zowe CLI as separate Software Development Kits (SDKs). [#750](#)
- The "`@zowe/cli`" package still includes both API and CLI methods. In addition, the following SDK packages are now available:
 - `@zowe/provisioning-for-zowe-sdk`
 - `@zowe/zos-console-for-zowe-sdk`
 - `@zowe/zos-files-for-zowe-sdk`
 - `@zowe/zos-jobs-for-zowe-sdk`
 - `@zowe/zos-tso-for-zowe-sdk`
 - `@zowe/zos-uss-for-zowe-sdk`
 - `@zowe/zos-workflows-for-zowe-sdk`
 - `@zowe/zosmf-for-zowe-sdk`
 - `@zowe/core-for-zowe-sdk`

The following enhancement was added to the **Imperative CLI Framework**:

- Exposed the `trim` parameter from `wrap-ansi` within `TextUtils.wordWrap()`. [#458](#)

The following enhancement was made to enable support for Node.js v14 for the **Secure Credential Store Plug-in**:

- Enabled support on Node.js v14 by updating Keytar dependency to v6. [#28](#)

The following enhancement was made to enable support for Node.js v14 for the **IBM Db2 Plug-in**:

- Enabled support for Node.js v14 by updating dependencies. [#60](#)

Zowe Explorer

- Added login and logout functions for base profiles. You can now log in to API Mediation Layer and generate a token for your base profile. [#914](#)

Zowe JES/MVS/USS Explorers

The following features and enhancements were added to the **JES Explorer**:

- Added ability to refresh content of an open job output file via context menu entry on the job file. [#549](#)
- Major material ui update from v1.x to 4.x, and minor react update. Accordian and snackbar changes as required by latest material-ui version.

Bug fixes

The following bugs were fixed.

Zowe API Mediation Layer

- Improved returned information while logging out via logout on Gateway. [#831](#)
- Updated API paths for the API ML in the API Catalog to use the service id in front. [#853](#)

Zowe App Server

- Make use of external certificate authorities referenced during keystore setup time.
- ZSS startup would issue warnings about failure to write yml files for APIML in the case APIML was not also being used.
- Bugfix: In previous versions, external certificate authorities were not registered with the app server properly and would sometimes contribute to a SELF_SIGNED_CERT_IN_CHAIN error when using the mediation layer. This issue has been resolved by adding external CA certs to the app-server CA array. [#138](#)

Zowe CLI

The following bug was fixed in the **core CLI**:

- Fixed incorrect syntax of example for `zowe files create data-set-vsam` in the help. [#823](#)

The following bug was fixed in the **Imperative CLI Framework**:

- Updated opener dependency due to command injection vulnerability on Windows. For more information, see [GHSL-2020-145](#).

Zowe Explorer

- Fixed the empty profile folders in Favorites issue. [#1026](#)
- Fixed the initialization error that occurred when base profiles were used while being logged out from API ML. [#1063](#)
- Fixed the issue preventing the tree refresh function from updating extender profiles. [#1078](#)

Version 1.16.0 LTS (October 2020)

Notable changes

Certificate management and keyring support

- In V1.15, the JCL member ZWEKRING was added to the sample PDS library SZWESAMP. This member contains commands to create a keyring that can contain the Zowe certificate(s) and a local certificate authority. In this

release, the JCL member ZWENOKYR was added to SZWESAMP that contains the inverse commands, so it can be used to remove the keyring, the Zowe certificate(s), and the certificate authority.

- In V1.15, the JCL member ZWEKRING and the supporting code in the Zowe runtimes for working with certificates held in keyrings were provided in beta format for early technical preview for RACF only. In this release, the commands in ZWEKRING, ZWENOKYR and the supporting code in the Zowe runtimes for working with keyrings and certificates in RACF, TopSecret, and ACF/2 are now a supported piece of functionality.
- A new documentation section is added to help you understand the configuration scenarios around Zowe certificates, and the relationship to a Zowe instance directory and Zowe runtime. See [Topology for the Zowe z/OS launch process](#).

Additional TN3270 terminal configuration options

Additional TN3270 terminal configuration options can now be specified within the `instance.env` configuration file. These choices, such as codepage and terminal dimensions, affect server defaults but do not change the pre-existing ability for you to set your own preferences within the Desktop at runtime. A list of the available options can be found [here](#).

New features and enhancements

The following features and enhancements were added.

Zowe installation

- Moved `explorer-ui-server` out of `explorers` into new shared folder under Zowe Runtime Directory. [#1545](#), [#207](#), [#37](#)
- Created `zowe-setup-keyring-certificates.env` and removed the overloaded properties from `zowe-setup-certificates.env` to try to simplify the user experience when setting up certificates in the keyring and USS keystore modes. [#1603](#)

Zowe API Mediation Layer

- ZAAS Client can now use HTTP so that the Application Transparent Transport Layer Security (AT-TLS) can be used for communication to ZAAS. [#813](#)
- Implemented the logout functionality in ZAAS Client. [#808](#)
- Added a more helpful and actionable description to message ZWEAM511E, which occurs when API ML does not trust the certificate provided by the service. [#818](#)

Zowe App Server

- The `install-app.sh` script used to install App Server plugins can now be used without Node.js. If Node.js is not detected when the script is executed, this behavior will be automated. You can also force this behavior with the environment variable `INSTALL_NO_NODE=1`, such as in the following example:
 - `INSTALL_NO_NODE=1 ./install-app.sh ~/zlux-editor` [#137](#)
- ZSS is now automatically registered to the API Mediation Layer when both are present, using a static registration file. [#208](#)
- Additional environment variables are now supported, which provides more options for TN3270 for the `instance.env` configuration file [#1176](#) while also allowing TN3270 host to be specified during installation configuration. [#1125](#). The following new environment variables are now supported in `instance.env` [#108](#):
 - `ZOWE_ZLUX_TELNET_HOST = string`
 - `ZOWE_ZLUX_SSH_HOST = string`
 - `ZOWE_ZLUX_TN3270_ROW = number`
 - `ZOWE_ZLUX_TN3270_COL = number`
 - `ZOWE_ZLUX_TN3270_MOD = numbers 2-5 as well as "dynamic" or other variations of the word`
 - `ZOWE_ZLUX_TN3270_CODEPAGE = ccsid number or string as seen in the ui`
- The Agent API now provides limited information without the need for authentication. Non-admins are able to view a subset of the information available to admins, specifically regarding the functionality of Zowe. Examples

of the information available to non-admins are: OS architecture and environment variables for Zowe configuration such as the components used and the ports they are accessible on. [#211](#)

- `/server/agent/environment` (limited info)
- `/server/agent/services`
- The ZSS `/unixfile` API has been updated to include an option to force file content to be sent or received as a specific encoding. If not specified, the pre-existing behavior of automatically choosing encoding based on tagging and file extensions will be used. [#160](#)
- The app server can now read and use keys, certificates, and certificate authorities contained with PKCS12 files. This is in addition to existing support for PEM-encoded files as well as z/OS keyrings. [#244](#)

Zowe CLI

The following enhancements were added to the **core CLI**:

- Added a `--pattern` option to the `zowe files list all-members` command. The option lets you restrict returned member names to only names that match a given pattern. The argument syntax is the same as the "pattern" parameter of the ISPF LMMLIST service. [#810](#)
- Added new options `--lrecl` and `--recfm` to the `zos-files create` command. Use these options to specify a logical record length and record format for data sets that you create. [#788](#)

Zowe Explorer

- Added the Allocate Like feature. [#904](#)
- Added the ability to disable/enable profile validation. [#922](#)
- Added the ability to access other profiles during profile validation. [#953](#)
- Grouped Favorites by profile for Datasets, USS, and Jobs. [#168](#)
- Once entered, datasets and members are displayed in uppercase. [#962](#)
- Updated the environment check for Theia compatibility. [#1009](#)

Zowe JES/MVS/USS Explorers

The following enhancement was added to the **Explorer UI Server**:

- Explorer UI Server is now published as separate pax and decoupled out of explorers. [#37](#)

The following features and enhancements were added to the **JES Explorer**:

- Moved `explorer-ui-server` out of explorers into new `shared` folder under Zowe Runtime Directory. Changed JES lifecycle start script to use new shared location. [#207](#)
- Added context menu entry for download JCL used to submit a job. [#335](#)
- Updated webpack to latest version, added `.npmrc` to specify npm registry as config. [#222](#)

The following features and enhancements were added to the **MVS Explorer**:

- Extracted out `explorer-ui-server`, changed MVS lifecycle start script and packaging script accordingly. [#151](#)
- Added loading icon to editor menu bar when opening a dataset's contents. [#291](#)
- Added `.npmrc` to specify npm registry as config.

The following features and enhancements were added to the **USS Explorer**:

- Extracted out `explorer-ui-server`, changed USS lifecycle start script and packaging script accordingly. [#100](#)
- Added `.npmrc` to specify npm registry as config.

Bug fixes

The following bugs were fixed.

Zowe API Mediation Layer

- Changed the default expiration time value for JWT token to 8h for consistency with the z/OSMF default. [#615](#)
- Reduced excessive and unhelpful log messages. [#672](#)

- Added the Base Path field in the API Catalog if one is available, which can override the Swagger Base Path. This causes the proper Base Path to be displayed in the event that the api doc is not populated properly. [#810](#)
- Removed overwriting of the Swagger Base Path, which resulted in malformed API routes when the base URL is shared among multiple services. [#852](#)
- API ML was previously not reporting SSL certificate errors when servers were unable to communicate. Now, if a SSLEException occurs, SSL certificate errors are reported. [#698](#)
- Fixed language in log messages for consistency. [#830](#)

Zowe App Server

In previous versions, the environment `arch` and `os` fields were incorrect. This has been fixed, and the updated response from `/server/agent/environment` service is [#213](#):

```
{
  "agentName": "zss",
  "agentVersion": "1.15.0+20200903",
  "arch": "s390x",
  "os": "zos",
  "osRelease": "04.00",
  "osVersion": "02",
  "hardwareIdentifier": "8561"
}
```

Zowe CLI

The following bug was fixed in the **FTP plug-in for Zowe CLI**:

- Fixed an issue where the `view spool-file-by-id` command retrieved incorrect contents. [#61](#)

Zowe Explorer

- Fixed USS renaming issues. [#911](#)
- Fixed the deletion of datasets issue. [#963](#).
- Removed errors in Favorites items caused by profiles that are created by other extensions. [#968](#)

Version 1.15.0 LTS (September 2020)

Notable changes

Keyring support

Prior to v1.15, the Zowe z/OS components were only able to use a certificate held in a USS Java KeyStore. In v1.15, the Zowe z/OS components can now use a certificate that is held in a z/OS keyring as described in [Configuring Zowe certificates in a key ring](#) on page 144.

For more information about Zowe certificates, certificate authorities, trust stores, and how they are used by Zowe, see [Configuring Zowe certificates](#) on page 136.

Auto-Save plug-in data

Plug-in developers can now make use of the new autosave feature, which can automatically save state data based on what the developer intends to retain, at regular time intervals. This is to protect against client crashes, and in the case of a crash, the apps are reopened upon desktop login and restored with the saved state. This new capability furthers the larger goal of high availability and fault tolerance for all Zowe components.

Support for starting Zowe API ML without z/OSMF on your system

By default, the API Gateway uses z/OSMF as an authentication provider. With the release of Zowe 1.15 it is now possible to switch to SAF as the authentication provider instead of z/OSMF. So, if you want to securely run the Zowe API ML but your system does not have z/OSMF, simply select SAF as your authentication provider. For more information on how to switch to SAF, see [API Gateway configuration parameters](#).

New features and enhancements

The following features and enhancements were added:

Zowe API Mediation Layer

- The API Path Pattern now supports `serviceId` as the first element. This improves the consistency of the URL when processing through the Gateway or outside of the Gateway. [#688](#)
- The SAF Provider can now be used as a possible authentication provider. This removes the API ML dependency on z/OSMF for authentication enabling SAF to obtain the JWT. [#472](#)
- The Swagger URL is now provided for z/OSMF. This URL provides full documentation containing the Try It Out functionality if the z/OSMF version supports the Swagger endpoint. Alternatively, the URL provides the info endpoint to directly enable access to Zowe endpoints. [#665](#)
- The default configuration of API ML now supports character encoding. [#777](#)

ZSS

A new endpoint has been added to the Agent API. This new endpoint will return a list of services to the user. [#209](#)

- Sample request: GET /server/agent/services
- Sample response:

```
{
  "services": [
    {
      "name": "plugin definitions service",
      "urlMask": "/plugins",
      "type": "REST"
    },
    {
      "name": "UnixFileContents",
      "urlMask": "/unixfile/contents/**",
      "type": "REST"
    },
    {
      "name": "UnixFileRename",
      "urlMask": "/unixfile/rename/**",
      "type": "REST"
    }
  ]
}
```

Zowe App Server

- Added a feature that allows users to auto save plug-in data by subscribing to the event. By default, the feature will auto save every 5 minutes, but this interval can be customized. [#250](#)
 - This feature is enabled via the Plugin Definition. `"autosave": true`
- You are now able to select multiple jobs in the job tree, which allows for functions such as purging multiple jobs at once. [#274](#), [#204](#)

Zowe CLI

The following features and enhancements were added to the **core CLI**:

- Added a `--responseTimeout` option to the z/OS Files APIs, CLI commands, and z/OSMF profiles. Specify `--responseTimeout <##>` to set the number of seconds that the TSO servlet request runs before a timeout occurs. The default is 30 seconds. You can set the option to 5 - 600 seconds (inclusive). [#760](#)
- Added the `--encoding` option for the `zowe zos-files upload dir-to-pds` command. This option lets you upload multiple members with a single command. [#764](#)

The following features and enhancements were added to the **Imperative CLI Framework**:

- Added support for dynamically generated cookie names. Updated `AbstractSession.storeCookie()` to process cookie names that are not fully known at build-time. [#431](#)

- Added the SSO Callback function, which allows applications to call their own functions while validating session properties (that is, host, port, user, password, token, and so on). The callback option is named `getValuesBack`. [#422](#)

The following features and enhancements were added to the **Secure Credential Store Plug-in**:

- Added the `scs revert` command. Use the command to revert securely stored credentials in your user profiles to be stored in plain text. [#22](#)
- Changed the `scs update` and `scs revert` commands so that they fail if Secure Credential Manager is not enabled. [#23](#)

Zowe JES/MVS/USS Explorers

The following features and enhancements were added to the **JES Explorer**:

- Changed the packaging and lifecycle `start.sh` script to add explorer-ui-server keyring support. [#1177](#)
- Added app bar, along with settings, and local storage to store user preferences and remember the last search filter. [#487](#)
- Notifications preference can set duration for snack bar notification. [#273](#)

The following features and enhancements were added to the **MVS Explorer** and **USS Explorer**:

- Changed the packaging and lifecycle `start.sh` script to add explorer-ui-server keyring support. [#1177](#)
- Added ability to collapse and resize jobs tree. [#259](#)

Bug fixes

The following bugs were fixed.

Zowe API Mediation Layer

- Fixed SSL validation when Eureka is running in HTTP mode. When the scheme is HTTP, SSL configuration is not verified since it is not used. [#792](#)
- Fixed a problem in error handling when no api-doc is available. Now a specific return code and message is generated when a problem occurs when obtaining or transforming the api-doc. [#571](#)

ZSS

- When RBAC is disabled, only the following services will be available. [#210](#)
 - `/server/agent/environment` (with limited information)
 - `/server/agent/services`

Zowe App Server

- External CA certificates to the Zowe `ZWED_node_https_certificateAuthorities` array only after checking to see if the certificates exist, which prevents it from pointing to nothing, resulting in it breaking. [#136](#)
- In previous versions, the `component.json` file was only being created when users upgraded their Zowe system to a more recent version. Performing an initial installation would not result in the `component.json` file being created. In this version, this bug has been resolved, and the `component.json` file is created both when upgrading and performing an initial installation. [#135](#)

Zowe CLI

The following bugs were fixed in the **core CLI**:

- Renamed the z/OS Files API option from `storeclass` to `storclass`. This fixed an issue where the CLI could define the wrong storage class on `create dataset` commands. [#503](#)
- Fixed an issue where the output of the `zowe zos-uss issue ssh` command would sometimes omit the last line. [#795](#)

The following bug was fixed in the **Imperative CLI Framework**:

- Fixed an issue with `ConnectionPropsForSessCfg` where the user would be prompted for user/password even if a token was present. [#436](#)

Zowe JES/MVS/USS Explorers

The following bugs were fixed in the **JES Explorer**:

- Fixed a bug where no jobs would show after auth token expired and user logs back in. [#408](#)
- Added default value for ZOWE_EXPLORER_FRAME_ANCESTORS at lifecycle start script. It resolves [#44](#).
- Fixed an issue where job tree height is greater than app container which makes the page scrollable. [#484](#)

The following bugs were fixed in the **MVS Explorer**:

- Fixed an issue where the dataset tree and the content viewer were not aligned. [#484](#)
- Added default value for ZOWE_EXPLORER_FRAME_ANCESTORS at lifecycle start script. It resolves [#44](#).

The following bugs were fixed in the **USS Explorer**:

- Added default value for ZOWE_EXPLORER_FRAME_ANCESTORS at lifecycle start script. It resolves [#44](#).

Version 1.14.0 LTS (August 2020)

Notable changes

Zowe Node APIs

Did you know that you can leverage the Zowe Node APIs directly? The Zowe Node APIs are the programmatic APIs that enable Zowe CLI to interface with the mainframe. You can use the APIs to build your own applications or automation scripts, independent of Zowe CLI. For more information and usage examples, see the [Zowe CLI readme file](#).

Support for verifying Zowe release integrity

Zowe now provides a new tool to verify that the code in the Zowe runtime directory installed on your z/OS® system is identical to the released code. The tool comprises a script file `zowe-verify-authenticity.sh`, plus the files it needs to check the release contents.

If the contents of the Zowe runtime directory have been modified, then it may result in unpredictable behavior. For more information about the tool, see [Verify Zowe runtime directory](#) on page 403.

New features and enhancements

The following features and enhancements were added.

Zowe installation

- If you are upgrading to Zowe v1.14 from a previous release, and the value of ZOWE_EXPLORER_HOST does not match the host and domain that you put into your browser to access Zowe, you must update your configuration due to updated referrer-based security. See [System Requirements](#) for information on updating your configuration.
- Allow the user to verify the authenticity of a Zowe driver. The script `zowe-verify-authenticity.sh` will check that a Zowe ROOT_DIR for an installed release matches the contents for when that release was created, which assists with support and troubleshooting. To verify pre-1.14 releases, the script and its associated code are available [separately](#) (see [#1552](#)). For more information, see the new topic [Verify Zowe runtime directory](#) on page 403 that describes the operation of the script.
- Allow multiple domains (names/IP Addresses) when generating certificates. This also includes SMP/E HOLDDATA for the affected function Zowe Configuration. [#1511](#)
- Included z/OSMF workflows for Zowe z/OS configuration. [#1527](#)
- Added warning if ZWESVSTC runs under user ID IZUSVR. [#1534](#)
- Changed the documentation so that SZWEAAUTH PDSE load library members should not be copied elsewhere, but instead the original installation target SZWEAAUTH PDSE should be APF-authorized and used as the runtime load library. This also includes SMP/E HOLDDATA for the affected function STC_JCL as well as changes to topics [Installing and configuring the Zowe cross memory server \(ZWESISTC\)](#) on page 146 and [Installing and starting the Zowe started task \(ZWESVSTC\)](#) on page 156.
- Added a new topic [Installing and configuring Zowe z/OS components using scripts](#).

API Mediation Layer

- Prevented crashing of API ML when null routes are set. [#767](#)
- Added support to the X-Forwarded-* Headers. [#769](#)
- Improved the configuration validator for the enablers to improve message specificity when one or more parameters required for setup are missing. [#760](#)

Zowe App Server

- Using a cross-memory server without REUSASID=YES may result in an ASID shortage. This pull-request adds a check that will print a warning if REUSASID=YES is not detected. [#145](#)
- In previous versions, the server used the property InstanceID instead of ZOWE_INSTANCE. In order to maintain backwards compatibility, these properties are now unified when the value of ZOWE_INSTANCE is non-default. Additionally, the server uses these values whenever an instance number is needed, such as in the case of determining profile names for RBAC use [#130](#)
- The packaged size of the Editor has been significantly reduced by removing uncompressed versions of files that have compressed variants and .map files which were used for development debugging. [#160](#)
- The ZSS /unixfile REST API now supports the changing of permissions on a file or folder, similar to chmod, by calling /unixfile/chmod. The behavior is documented [in swagger](#). [#195](#) [#132](#)
- A notification will be displayed when users attempt to upload a wallpaper image that is too large. [#254](#)
- The desktop personalization panel's color selection UI now has an extra highlight around the selected color to make the selection more apparent. [#236](#)
- Users can now recall migrated datasets in the Editor (via the File Tree) by clicking on them. [#78](#)

Zowe CLI

The following features and enhancements were added to the **core CLI**:

- Added the command zowe zos-files delete migrated-data-sets to delete migrated data sets. [#716](#)
- Added a new --fail-fast option to the zowe zos-files download all-members command. The option defaults to true, which preserves existing behavior. Set the option to false to continue downloading members if one or more of the downloads fails. [#759](#)
- Updated the Imperative CLI Framework version. [#744](#)

z/OS FTP Plug-in for Zowe CLI:

The following enhancement was added to the z/OS FTP Plug-in:

- The following flags were added to the zowe zos-ftp submit data-set command: [#55](#)
 - wait - Specify a query interval and max times to query as comma-separated, numeric values. For example, specify 5,12 to query the job status every 5 seconds up to 12 times.
 - wait-for-output - Wait for the job to enter OUTPUT status.
 - wait-for-active - Wait for the job to enter ACTIVE status.

Zowe Explorer

The following features and enhancements were added to the **Zowe Explorer**:

- Added a webpack that works with localization and logging.
- Allowed extenders to load the saved profile sessions upon activation.
- Added an automatic re-validation for invalid profiles.

Also, check out [the Zowe Explorer FAQ](#) to learn more about the purpose and function of the VS Code extension.

Bug fixes

The following bugs were fixed.

Zowe App Server

- Bugfix: ZSS will now maintain the connection if users respond to the 404 message with the request Connection: Keep-Alive [#147](#)
 - **NOTE:** The code only recognizes Connection: Keep-Alive. Other "Keep-Alive" properties will be ignored.
- Bugfix: If a load module is incorrectly copied to STEPLIB, the z/OS loader will fail to load it. In these cases, an available copy in LPA will be used instead, if one is available. The problem with LPA is that any IDENTIFY calls to a module with an incorrect version number may cause serious issues. This pull-request ensures that ZWESIS01 comes from private storage. [#146](#)
- Bugfix: Fixes various issues that would occur when the number in the Content-length response header was different from the actual content length. [#150](#)
- Bugfixes for default plugin config and terminal handler location. This change was made in order to include the _internal folder. storageDefaults other than _internal are already supported. For more information, see the [wiki](#). [#229](#)
 - This fix allows the server-side plugin config to exist within its own folder, rather than in the instance directory. As a result, plugins no longer have to perform a copy operation during installation.
 - You can now specify terminal proxy handler overrides within \$INSTANCE_DIR, which was previously only possible within \$ROOT_DIR. \$ROOT_DIR modification is not recommended and not conformant for Zowe plugins.
- Bugfix: The process of auto-converting untagged USS ebcdic files when using the ZSS /unixfile REST API has been improved by determining if the files are text or binary based on a list of file extensions. The API behavior towards unknown extensions has been changed from assuming text to now assuming binary. This fixed some cases where text files were not readable through the REST API. [#148](#) [#152](#)
- Bugfix: When using ZSS's /unixfile/contents REST API, large files would occasionally cause an incorrect HTTP message to be sent because the content-length header did not match the actual content length. This could result when there is a conversion error. This issue has been solved by updating the API, allowing it to use the transfer encoding type "chunked" instead, which allows these previously broken files to be sent successfully. [#150](#)
- Bugfix: Some file actions in the Editor would generate URLs that included multiple slashes in a row, which may cause errors on servers that receive such requests. In this update, the URI Broker now removes multiple slashes when they are encountered, which may additionally improve behavior in other apps that use the URI Broker for the ZSS REST API /unixfile. [#251](#)
- Bugfix: During ZSS initialization, certain warning log messages were not displayed, such as the warning about lack of permission to use ICSF to generate a random number. This issue has been resolved by initializing the logger responsible for issuing the messages. [#143](#)
- Bugfix: In order to conserve log space, ZSS no longer prints debug information regarding HTTP dispatch. [#156](#)
- Bugfix: In previous versions, the app framework build process referenced webpack incorrectly, leading to an unnecessary build-time error if webpack was not installed globally. This issue has been resolved. [#248](#)
- Bugfix: In previous versions, developing with the app framework would show linting warnings in VSCode. This issue has been resolved by updating tsconfig.json [#240](#)
- Bugfix: Some app server configuration values could not be specified via environment variables due to the limited characters allowed in variables. A new syntax has been made to allow these edge-case configuration values to be specified, and this new syntax is seen here: [#230](#)
 - Overall behavior is described [in the wiki](#).

Zowe CLI

The following bug was fixed in Imperative CLI Framework:

- Fix update profile API storing secure fields incorrectly when called without CLI args.
- Fixed a compilation error when building the CLI from source. [#770](#)

Zowe Explorer

- Fixed the bug related to saving USS files.
- Fixed the bug related to the deletion of datasets.

Version 1.13.0 LTS (July 2020)

Notable changes

Zowe CLI added the ability to access mainframe services through API Mediation Layer using single-sign on (SSO) and multi-factor authentication (MFA). Use Zowe CLI to log in to API Mediation Layer and receive a token that is used for secure authentication to one or more services. For more information, see [Integrating CLI with API Mediation Layer](#).

The CLI also supports a type of profile named "base profile" that lets you store configuration information for multiple services. For more information, see [Using Profiles](#).

New features and enhancements

The following features and enhancements were added.

Zowe installation

- Updated zowe-configure-instance upgrade to update ROOT_DIR. This allows you to move the Zowe runtime to a different place when you install a new version of Zowe. [#1414](#)
- Updated the port validation logic to reduce false negatives. [#1399](#)
- Updated the Zowe installation and configuration to tolerate ZERT Network Analyzer better. [#1124](#)

API Mediation Layer

- Added Cross-origin resource sharing (CORS) Headers Support.
- Introduced an option to set connection timeout for a service.
- Provided SAF Keyrings support for a ZAAS Client.
- Introduced Spring Boot enabler configuration validation.

Zowe App Server

- The app server is now able to use more than one certificate authority (CA). This allows the server to validate other server's authenticity by recognizing the CA that another server may have used [#128](#)
- The dispatcher.invokeAction method now returns promise, which provides the ability to wait until dispatcher.invokeAction finishes and handles errors [#59](#)
- The ngx-color picker has been replaced by a custom hue selection bar, lightness swatches bar, and color palette, allowing for a more customizable personalization experience [#235](#)
- In this version, cross-launch via URL has been implemented, allowing for integration between the Application Framework and applications. This feature enables users to bookmark a set of app2app communication actions (in the form of a URL) that will be executed when opening the webpage [#234](#)
- Bookmarking features have been added to the TN3270 emulator [#30](#)
 - Users can now save connection preferences on a per-user level. Clicking the floppy disk icon saves user settings to that user's scope.
 - Codepages have been reorganized so that the numbers are shown first, making it easier for users to navigate to their favorites
 - The buttons found in this feature have been realigned
- Several features have been added to the Zowe Editor [#153](#)
 - Globally increased the shortest duration of snackbar notifications from 2 seconds to 3 seconds
 - Added a "Close All" button in the menu (hot key is Alt + W + Shift)
 - A snackbar notification will be displayed when users attempt to open a file that they do not have permission to open
 - Added an "Undo" option to the Close All feature to reopen tabs & files
- Login activity and session activity is now synchronized across multiple desktop tabs [#242](#)
 - When a user logs out of a desktop tab, all other active tabs will also log out
 - When a user performs an action on a desktop tab, the other tabs register this activity, which stops them from timing out

Zowe CLI

The following features and enhancements were added to the **core Zowe CLI**:

- Added the ability to log into and out of API ML using a token. [#718](#)
- Added the `--base-profile` option to all commands that use profiles to let them make use of base profiles that contain shared values. [#718](#)
- CLI commands now prompt for any of the following option values if the option is missing: host, port, user, and password. [#718](#)
- Added character encoding/code page support for download and upload data set operations in the API library and the CLI. [#632](#)
- Added the `--encoding` option to the `zosmf` profile type. [#632](#)
- Introduced an API to delete migrated data sets. [#715](#).

The following features and enhancements were added to the **Imperative CLI Framework**:

- Added the `ConnectionPropsForSessCfg.addPropsOrPrompt` function to store credentials, such as a token, in a session configuration object. [#718](#)
 - CLI plug-ins must implement this function to create sessions in order to consume automatic token-handling and prompt for mission options features.
 - Connection information is obtained from the command line in the following order: Environment variables, service profiles, base profiles, or a default option value.
 - If connection information is not supplied to any core CLI command, the user is prompted for:
 - host
 - port
 - user
 - password

The prompt times out after 30 seconds so that automated scripts will not fail.

- Added base profiles, a type of profile that can store values and provide them to other profile types, such as `zosmf` profiles. [#402](#)

The following properties can be stored in a base profile:

- host
- port
- user
- password
- `rejectUnauthorized`
- `tokenType`
- `tokenValue`
- Added `login` and `logout` commands to retrieve and delete tokens. [#405](#)
 - Added a `showToken` flag to display the token and not save it to the user profile.
 - Added the ability to create a user profile upon login, if no profile of that type existed previously.
- Added the `--dd` flag, which lets users create a profile without using the default values specified for that profile. [#718](#)
- If a token is present in the underlying REST session object, Imperative uses the token for authentication.
- CLI help text includes new options such as `tokenValue`. Plug-in developers might need to update mismatched snapshots in automated tests.
- Updated the version of TypeScript from v3.7.4 to v3.8.0.
- Updated the version of TSLint from v5.x to v6.1.2.
- Update `log4js` to improve Webpack compatibility for extenders.

Zowe Explorer

The following features and enhancements were added to **Zowe Explorer**:

- Added a credentials check feature that allows users to update their credentials if they receive an authorization error.
- Added a star icon that clearly denotes data sets, USS files, and jobs as favorites.
- Added a profile validation feature that checks whether a profile is valid. The feature is triggered when any action is performed with the profile. Validated profiles are indicated by a green mark.
- Disallowed case sensitivity for profiles with same names.
- Enabled editing of search filters.
- Enabled editing of ASCII files in USS.
- Improved text in confirmation dialogs.
- Reorganized the Data Sets context menu to match the order of commands recommended by VSCode.

Bug fixes

The following bugs were fixed.

ZSS

- Bugfix: ICFS error message is not printed. In this version, the issue has been resolved [#143](#)

Zowe App Server

- Bugfix: Changing editor syntax in the MVS explorer caused a callstack limit exception. This was due to a trap focus conflict between the Orion editor and the modal part within the ui Select component on syntax change. In this version, the issue has been resolved by disabling `disableEnforceFocus` for the syntax selector [#129](#)
- Bugfix: An Infinite Auth loop would occur on explorer apps due to APIML and z/OSMF auth timeouts mismatch. In this version, the issue has been resolved by adding a force login flag if a datasets request comes back as 401 [#124](#)
- Bugfix: When using the JES Explorer to view Spool files of a job, users cannot open a spool file that has the same name as one already open. This issue has been resolved by adding a unique id to content tabs to allow opening of overlapping names [#188](#)
- Bugfix: The Env var for TERM gets set to "linux", which is not recognized by USS. This issue has been resolved through the removal of rxjs-compat [#29](#)
- Bugfix: NGX-monaco-editor library has been removed in order to fix a bug. This now allows the Editor to open and view files after the second instance of opening them [#155](#)
 - Removed use of node-sass, so that native compilation is not required
 - Updated to typescript 3.7 from version 2.7.2
 - Updated to monaco 0.20 from version 0.13. The monaco changelog can be found [here](#)

Zowe CLI

- Fixed an issue where CLI web help failed to load in Internet Explorer 11. [#393](#).
- Fixed an issue where the `--help-web` option did not function on macOS when the `DISPLAY` environment variable was undefined. [#322](#).
- Updated Imperative version to include security fixes.
- Updated Imperative version to fix a problem where users could not use a service profile after storing a token in a base profile.
- Fixed an issue where optional secure fields were not deleted when overwriting a profile.

Version 1.12.0 LTS (June 2020)

New features and enhancements

The following features and enhancements were added.

Zowe installation

- Keystore directory generation updated to add new parameters. If you wish to enable SSO for the desktop you need to rerun the `zowe-setup-certificates.sh` script during the upgrade process, with new values in the `zowe-setup-certificates.env` file. [#1347](#) / Doc: [#1162](#)

- Added a `-l` optional parameter to the `zowe-support.sh` script. This parameter allows you to specify the custom log directory used in installation and configuration when collecting support data. [#1322](#) / Doc: [#1165](#)
- Added the validate only mode of Zowe. This allows you to check whether all the component validation checks of the Zowe installation pass without starting any of the components. [#1335](#) / Doc: [#1181](#)
- Separated ZSS component from the Zowe App Server component. [#1320](#)
- Introduced z/OSMF Workflows that accomplish the following Zowe installation and configuration tasks:
 - Install Zowe runtime using z/OSMF Workflows.
 - Configure z/OS Security Manager.
 - Configure Zowe certificates.
 - Create and configure the Zowe instance directory, and run the Zowe started task.

API Mediation Layer

- Provided Zowe Authentication and Authorization Service (ZAAS) client.
- Refreshed the static client definitions from the API Catalog UI.
- Switched to `sso-auth` instead of `apiml-auth`.
- Added logout endpoint API documentation.
- Made `j jwt` only a test dependency.
- Fixed the order of fetching the JWT from a request.
- Implemented request retrying for service instances.

ZSS

- ZSS now follows the Zowe Component scheme, as part of the DESKTOP component group [#177](#)
- Read JWT token information from environment variables, if they exist, to further support SSO during a standard installation. [#178](#)
- In previous versions, ZIS did not use the version information provided in `zss/version.txt`. In this version, the ZIS build uses `version.txt` the same way that ZSS uses it. [#184](#)

Zowe App Server

- Added SSO token name and label to `convert-env.sh` for use with ZSS. [#118](#)
- Script has been updated to allow ZSS to be a separate component. [#117](#)
- The app-server will favor and use a SAF keyring if defined for use in Zowe, rather than a unix file for keys, certificates, and certificate authorities. [#116](#)
- The process for making bundled plugins using `ROOT_DIR` has been upgraded [#123](#)
- Updates have been implemented for modal keyboard accessibility. [#148](#):
 - Editor now has keyboard navigation in the browsing tree and pop-up modals.
 - Pop-ups can be traversed with Tab/Tab + Shift.
- Desktop redesign suite and personalization settings have been implemented. [#221](#)
- Right-click context menus have been implemented for the new desktop style. [#216](#)
- A new attribute has been implemented to load plugins from different relative paths. [#212](#)

Zowe CLI

The following features and enhancements were added to the **core Zowe CLI**:

- Added the `zowe files hrec ds` command to recall data sets. [#556](#)
- Made the `account` option optional in TSO profiles. [#709](#)
- Made `user` and `host` options optional in SSH profiles. [#709](#)

The following features and enhancements were added to the **z/OS FTP Plug-in for Zowe CLI**:

- Added the `zowe zos-ftp list data-set-members` command to find members in a PDS. [#45](#)
- Added the `zowe zos-ftp make uss-directory` command. [#47](#)

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](#).

Bug fixes

The following bugs were fixed.

Zowe installation

- Minor enhancements to add log directory validation and remove unnecessary log file splitting. [#1334](#), [#1300](#)
- When the automatically detected hostname that Zowe is installed on cannot be resolved, use the IP address instead. This covers the scenario when the USS `hostname` command returned a system name that wasn't externally addressable. [#1279](#)
- Fixed an issue that could cause an upgraded version of Zowe to try and use an old version of plug-ins, by switching the desktop to use a relative reference to find plugins. [#1326](#)

ZSS

- Bugfix: Fixed a segfault when no config file is provided by moving all the zowelog invocations to a location where the logging environment is ready. Additionally, cleanup logic has been introduced to ensure that we free the STC base resources before leaving main. [#187](#)
- Bugfix: In previous versions, if a warning message is produced by the compiler, the build process is considered successful. This is often dangerous as warnings can indicate passing the wrong type or redefinition of a `#define`, which should be considered bugs. The following changes have been implemented to make the build process more strict [#188](#):
 - Make sure there are no warning messages in the current build.
 - Update deps to remove the `httpfileservice.c` warning message, and pick up a minor type fix.
 - Ensure side-deck `file/SYSDEFSD DD` by adding the `d11` option to the linker.
 - Adjust the compiler env variable that controls the severity.
 - Ensure no ZSS binary is created if `RC != 0`.

Zowe App Server

- Bugfix: Logout of sso-auth was not working because it was expecting apiml parameters that should have been there but were controlled by the env var `APIML_ENABLE_SSO`. In this version, the issue has been resolved. [#126](#)
- Bugfix: In this release, many bugs picked up by the Sonar scan for core Zowe repositories have been resolved [#214](#)
- Bugfix: Plugin api would not respond if a plugin could not load due to a dependency not being met. That plugin would not be placed in the array that checks when the processing has finished, so a response would never be generated. [#208](#)
- Bugfix: Fixed a logout cookie bug and sso-auth behavior bug in order to fully support SSO. Additionally, `tokenInjector` was removed as it is no longer required with the introduction of SSO. [#209](#)
- Bugfix: Fixed lease information for API ML [#218](#)
- Bugfix: In previous versions, the user was never shown the logout screen when the plugin would detect zss, but not apiml. In this version, this issue has been resolved. [#221](#)
- Bugfix: Fixed issue where `localhost & 127.0.0.1` were always used even when not true. Additionally, each worker in the cluster attempted registration even though, from an outside perspective, it is 1 server. In this version, the server uses a real hostname and tries to find the ip that best matches what apiml would be able to use [#203](#)

Zowe CLI

Updated Yargs in Zowe Imperative CLI Framework to fix vulnerabilities.

Version 1.11.0 LTS (May 2020)

New features and enhancements

The following features and enhancements were added:

API Mediation Layer

The following new feature was added to the Zowe API Mediation Layer in this version:

- The 'Try it out' functionality has been added to test for public and private endpoints. [#258](#)

[API ML Changelog](#)

ZSS

- A new query parameter (?addQualifiers) which can be appended to /datasetMetadata/ allows for searching that more closely represents the search behavior of 3.4 [#108](#)
- Added support for changing log levels via REST API [#173](#)

Zowe App Server

- Updated the JES Explorer, MVS Explorer, and USS Explorer apps to support single sign-on from the Zowe API Mediation Layer. [#344](#) [#345](#) [#346](#)
- Modals in the Editor now have an "X" icon to close the modal. [#130](#)
- An event emitter for session changes, login, logout, and sessionExpire for Angular, React, and iFrame applications has been added [#210](#)
- Session events have been added to mvdhosting [#53](#)
- Updates made to generate_zlux_certificates.sh because apiml_cm.sh has been moved into the zowe-install-packaging repo [#110](#)
- Zowe Web browser plugin, which can be used to view webpages that are not Zowe apps, has been added. [#194](#)
- Translations have been added for labels and buttons for password reset forms [#215](#), [#218](#)
- Browser-based apiml token, auth simplification [#196](#):
 1. API mediation layer token is now held in the browser upon login via the Desktop. This also allows for the Desktop to do single-sign-on login with the token if it is already present in the browser.
 2. Auth plugins no longer need to be specified explicitly within the server configuration file, the capability remains for backwards compatibility. The server will now auto-detect the auth plugins that are available
 3. Auth plugins can now be of more than one type, to satisfy environments that have plugins that need access to APIs of similar but different types
- New shortcuts have been added to navigate the start menu with a keyboard [#213](#)
- Sessions are now maintained based on most recent activity across tabs [#219](#)
- Support for password changing, including expired password changing, has been implemented [#193](#)

Zowe APIs

Zowe Jobs APIs

- Version 2 APIs now support single sign-on from the Zowe API Mediation Layer [#21](#)
- Updated embedded Spring Boot version [#89](#)

Zowe Data Set and Unix Files APIs

- Version 2 APIs now support single sign-on from the Zowe API Mediation Layer [#18](#)
- Updated embedded Spring Boot version [#151](#)
- Added incomplete connect timeout parameter to prevent Slowloris DOS attacks [#158](#)

Zowe CLI

Reference the appropriate version in each of the following changelogs to learn about CLI features, enhancements, and fixes:

Core CLI Changelogs:

- [Zowe CLI - v6.11.0](#)
- [Imperative CLI Framework - v4.6.0](#)
- [Secure Credential Store Plug-in - v4.0.4](#)

CLI Plug-in Changelogs:

- [IBM CICS Plug-in - v4.0.2](#)
- [IBM DB2 Plug-in - v4.0.6](#)
- [IBM FTP Plug-in - v1.0.2](#)
- [IBM IMS Plug-in - v2.0.1](#)
- [IBM MQ Plug-in - v2.0.1](#)

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](#).

Zowe installer

- Added a `-l` parameter to the [Installing Zowe runtime from a convenience build](#) on page 102, `zowe-setup-certificates.sh`, `zowe-install-xmem.sh`, and [Step 1: Copy the PROCLIB member ZWESVSTC](#) on page 156 scripts. This parameter allows you to specify where the setup scripts write trace logs.
- Improved port validation to assist determining whether Zowe's ports are available.

Zowe troubleshooting

- Improved the troubleshooting script `zowe-support.sh` to assist with offline problem determination. See [Capturing diagnostics to assist problem determination](#) on page 401.

Zowe documentation

- Added a topic [Zowe runtime lifecycle](#) on page 287 that describes the use of the `EXTENDER_COMPONENTS` value in the `instance.env` file. See [Extensions](#) on page 154.
- Improved the [Zowe architecture](#) on page 13 information to include a more current architecture topology diagram and more details on the individual Zowe services, where they log their data, and how to perform high-level problem determination.
- Added new problem determination scenarios and resolution. See [Troubleshooting Zowe Application Framework](#) on page 439
- Added information on how to determine which release of Zowe is installed. See [Zowe releases](#).
- Added a [Zowe resources](#) on page 68 topic, which provides a list of resources that supplement the documentation on this site.

Bug fixes

The following bugs were fixed:

ZSS

- Bugfix: Fixed a below-the-line leak in the QSAM code [#138](#)

Zowe App Server

- Bugfix: Material dialogs no longer overlap over the login screen [#145](#)
- Bugfix: Re-login to same desktop session would duplicate items in the launch menu. In this version, the session is cleared on logout, fixing the duplication issue [#208](#)
- Bugfix: Bugfix for websockets to prevent server throwing exception on malformed message [#189](#)
- Bugfix: Fixed app server configuration bug where min worker count was ignored when max worker count was not defined [#187](#)
- Bugfix: Added missing pluginID argument for `setStorageAll` method. [#191](#)
- Bugfix: app-server agent information was not available to plugins if it was specified via command line arguments [#111](#)

Version 1.10.0 LTS (April 2020)

New features and enhancements

The following features and enhancements were added:

API Mediation Layer

The following new feature was added to the Zowe API Mediation Layer in this version:

- Zowe API ML can now use z/OSMF to provide JSON Web Tokens (JWT). [#433](#)

ZSS

- Fast EBCDIC to UTF8 character translation is now supported by using the TROO instruction with a "EBCDIC 1047 to ISO/IEC 8859-1" translation table. [#127](#)
- Performance improvements in character conversion, JSON and collections code. [#162](#)
- The code now prints fewer warnings when AT-TLS is not set up. [#130](#)
- ZSS logs belonging in the ZSS repo have been refactored so that they now use the Zowe logger and message IDs. [#163](#)
- Config variable names have been updated to stay consistent with IBM terminology. [#165](#)

Zowe App Server

- The sample-react-app README has been updated to state prerequisites. [#20](#)
- An example of how to use the Zowe Desktop's built-in context menu has been added. [#31](#)
- Sample angular app has been updated for angular 6 best practices use of HttpClient, RxJS [#33](#)
- Simple conda build scripts have been added. [#46](#)
- App server logs now have IDs prefixed, for easy lookup in future documentation. [#49](#)
- Enhancements for plugin adding. [#51](#)
- App server logs now have IDs prefixed, for easy lookup in future documentation [#102](#).
- App server now defaults to prevent apps from being embedded in an iframe that does not come from the same origin. [#104](#)
- The jes-explorer has been updated to support Single Sign On functionality offered by the api-layer. [#160](#)
- Desktop now has key bindings to minimize (ctrl-alt-down), maximize windows (ctrl-alt-up), and show launchbar menu (ctrl-alt-m). [#176](#)
- App server “router”-type dataservices now have a new Storage API within their context object, for standardized in-server state persistence. [#178](#)
- App server can now add plugins on-demand without a restart, by re-scanning plugins directory via REST API / plugins. [#179](#)
- App server can now be configured to set HTTP headers that will default and possibly override those of the plugins. [#180](#)
- App server /auth API now returns which handler is the default. [#183](#)
- Events and actions for viewports and windows are now accessible to iframe via the standardized window.ZoweZLUXiframe object. [#184](#)
- Focus on app2app, as well as some package updates. [#188](#)
- 3 features:
 1. Desktop can now filter the list of apps by search query.
 2. Desktop cleanup has reduced the bootstrapping server requests by half.
 3. Desktop now can load new apps added to the server without a page reload. [#189](#)
- Desktop's DOM now has lang attribute as soon as the language preference is known. [#190](#)
- Desktop login screen updated with new Zowe logo. [#204](#)
- JES, MVS Explorers now have support for APIML's Single Sign On feature [#344](#)

Zowe CLI

The Secure Credential Store plug-in is now packaged with tools that build dependencies locally. This fixes an issue where the installation could fail at sites with firewall restrictions. [#9](#)

Tip: Zowe CLI release notes are now aggregated in changelogs. Reference the appropriate version in each changelog to learn about features, enhancements, and fixes.

Core CLI Changelogs:

- [Zowe CLI - v6.10.1](#)
- [Secure Credential Store Plug-in - v4.0.3](#)

CLI Plug-in Changelogs:

- [IBM CICS Plug-in - v4.0.2](#)
- [IBM DB2 Plug-in - v4.0.5](#)
- [IBM FTP Plug-in: - v1.0.1](#)
- [IBM IMS Plug-in: - v2.0.1](#)
- [IBM MQ Plug-in: - v2.0.1](#)

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](#).

Bug fixes

The following bugs were fixed:

Zowe z/OS Installation

Bugfix: `zowe-configure-instance.sh` does not allow the `-c` instance directory location to be an existing Zowe runtime. This caused a deadlock and running out of BPXAS instances. See [Unable to create BPXAS instances](#) on page 410. [#1123](#)

Zowe App Server

- Bugfix: subloggers would not inherit message translation maps from parent loggers. [#24](#)
- Bugfix: `sample-angular-app` could not be run from a folder outside of `$ROOT_DIR`. [#34](#)
- Bugfix: Menu locations were wrong when multiple apps opened because the numbers used partially came from the previous instance. [#36](#)
- Bugfix: Apps that were the target of app2app communication were not put into focus. [#50](#)
- Bugfix: Developers could not run app-server without a certificate authority. [#98](#)
- Bugfix: App server could not work with self-signed/invalid TLS certificates sometimes used in test/development, because the configuration option broke. The option has been restored. [#103](#)
- Bugfix: App server instance settings initialization had inconsistent write permissions. [#105](#)
- Bugfix: App server no longer issues warning about failure to load undefined log file. [#182](#)
- Bugfix: Fixes unformatted messages when a language is not specified. [#186](#)
- Bugfix: Editor would not work for unix files when used through api mediation layer due to encoded slash. [#187](#)
- Bugfix: App framework's right click menu could go off screen vertically at the bottom. [#200](#)
- Bugfix: `zosmf-auth` no longer issues configuration warning during startup. [#398](#)
- Doc Bugfix: Sample react app did not state its dependence on the sample angular app. [#405](#)
- Bugfix: Substitute `zosmf-auth` for `apiml-auth` to remove warning. [#1232](#)

Version 1.9.0 LTS (February 2020)

Zowe v1.9.x is designated as the current Zowe Long-term Support (LTS) version.

New features and enhancements

The following features and enhancements were added:

API Mediation Layer

The following new features and enhancements have been made to the Zowe API Mediation Layer in this version:

- Support of special characters has been added to API Mediation Layer core services. In addition, all onboarding enablers now support special characters as well.
- Custom metadata support has been added to the onboarding enablers. Additional parameters can now be easily added to an expandable parameter array. This feature may be used for security configuration in the future.
- Passticket support has been added to API ML Core Services and onboarding enablers. This makes it easier to authenticate existing mainframe applications with the API Mediation Layer.
- New versions of Spring Boot based onboarding enablers (V1 and V2) have been released. These enablers support the new version of the metadata required by the Discovery Service. The new versions of the enablers consume significantly less disk space.

The following bug fixes have been introduced:

- A fix of a critical authentication issue with some versions of z/OSMF has been applied.
- A fix has been applied to support multipart requests.
- A fix has been applied to the z/OSMF authorization header.

Zowe App Server

- Added support for Node.js - z/OS V12. See [Installing Node.js on z/OS](#) on page 76 for details.
- A new endpoint for removing dataservices has been added [#62](#)
- Functionality for removing data sets has been added [#65](#)
- Deletion of data sets and their members is now supported [#88](#)
- Deletion of data sets and their members is now supported [#85](#)
- The following helper functions have been added to test caller's environment [#115](#):
 - A function to test whether the caller is running in SRB
 - A function to test whether the caller is in cross-memory mode
 - A function to test whether the caller is holding a CPU, CMS, CML or local lock
- The logout endpoint has been re-added for zss [#100](#)
- Added support of SRB and locked callers to the Cross-Memory server's PC space switch routine [#153](#)
- This pull request add the following features [#120](#): Ability to use the lock-free queue intrusively which allows a more flexible storage management on the user's side Functions to copy to/from foreign address space using destination/source keys and ALETS
- Reformatted the save as modal in zowe editor [#129](#)
- Added Snackbar notification for directory error [#131](#)
- Removed language server tab in editor [#134](#)
- Explicitly call zss for logout to make sure cookies are known to be invalid [#28](#)
- The following changes have been made to Zlux server framework logging [#174](#):
 - Added English resource files for messages
 - Added code to all error, warning, debug and informational logged outputs
 - Replaced most console.log calls with logger calls
- Support for HTTP-Strict-Transport-Security. Custom headers for static content are now available [#173](#)
- Functionality for controlling application access for individual users has been added [#216](#)
- Out-of-band multi-factor authentication is now supported [#225](#)

Zowe CLI

To leverage the new features and plug-ins available in this version, you must follow the steps in [Migrating to Long-term Support \(LTS\) version](#) on page 177.

The following new CLI plug-ins are added:

- IBM® z/OS FTP Plug-in for Zowe CLI on page 248
- IBM® IMS™ Plug-in for Zowe CLI on page 249
- IBM® MQ Plug-in for Zowe CLI on page 250
- Secure Credential Store Plug-in for Zowe CLI on page 251

The following new features and enhancements are added in this version:

- **Notable Change:** The `zowe zos-files download ds` and `zowe zos-files download uf` commands no longer put the full content in the response format json (`--rfj`) output. [More information](#).
- **Notable Change:** The `--pass` option is changed to `--password` for all commands and profiles (zosmf, cics, etc...). The aliases `--pw` and `--pass` still function. To update a profile, issue the `zowe profiles update` command and use the new option name `--password`.
- **Notable Change:** You can enter `PROMPT*` as a value for any CLI option to enable interactive prompting. If you wrote scripts in which any option is defined with the exact value `PROMPT*`, the script will not execute properly in this version. For more information, see [Using the prompt for sensitive options](#).
- Zowe CLI was tested and confirmed to run on Unix System Services (USS) on z/OS. For more information, refer to blog [Installing Node.js on the Mainframe](#).

(The IBM Db2 and Secure Credential Store plug-ins for Zowe CLI will *not* run on z/OS due to native code requirements.)

- The `zowe files copy` command was added for copying the contents of a data set or member to another data set or member. [#580](#)
- Zowe CLI now exploits Node.js stream APIs for download and upload of spool files, data sets, and USS files. [\(#331\)](#)
- The following new commands were added for interacting with file systems:
 - `zowe zos-files list fs` [#429](#)
 - `zowe zos-files mount fs` [#431](#)
 - `zowe zos-files unmount fs` [#432](#)
- The following new commands were added for creating USS files and directories:
 - `zowe zos-files create file` [#368](#)
 - `zowe zos-files create dir` [#368](#)

The IBM® CICS® Plug-in is updated with the following functionality:

- **Notable Change:** The plug-in now uses HTTPS by default when connecting to CMCI. The option `--protocol http` was added to let you override the default as needed. [#77](#)
- Define, enable, install, discard, disable, and delete CICS URIMaps. [#53 #49 #48 #51 #50 #52](#)
- Define and delete CICS web services. [#58 #59](#)
- Add and remove CSD Groups to/from CSD Lists [#60](#).

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](#).

Watch a video on how to [Installing](#) on page 253.

Bug fixes

The following bugs were fixed:

Zowe App Server

- URL encoding with % sign were always returning with authorization:false with RACF [#27](#)
- Users are no longer able to delete the initial "/" in the address bar for selected files [#379](#)

- The search bar text for datasets has been changed from "Enter a dataset" to "Enter a dataset query". The Address bar text for files has been changed from "Enter a directory" to "Enter an absolute path" #60

Version 1.8.1 (February 2020)

Bug fixes for Zowe CLI

A bug was fixed where Zowe CLI installation could fail and users could receive the following error message when installing Zowe CLI v1.8.0:

```
981 verbose stack Error: EPERM: operation not permitted
```

To install the fix, download the new v1.8.1 package from [Zowe.org](#) and retry the installation process.

Version 1.8.0 (February 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 `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 `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 [z/OS Installation Roadmap](#) on page 99.

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 [Installing](#) on page 253.

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` script 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.

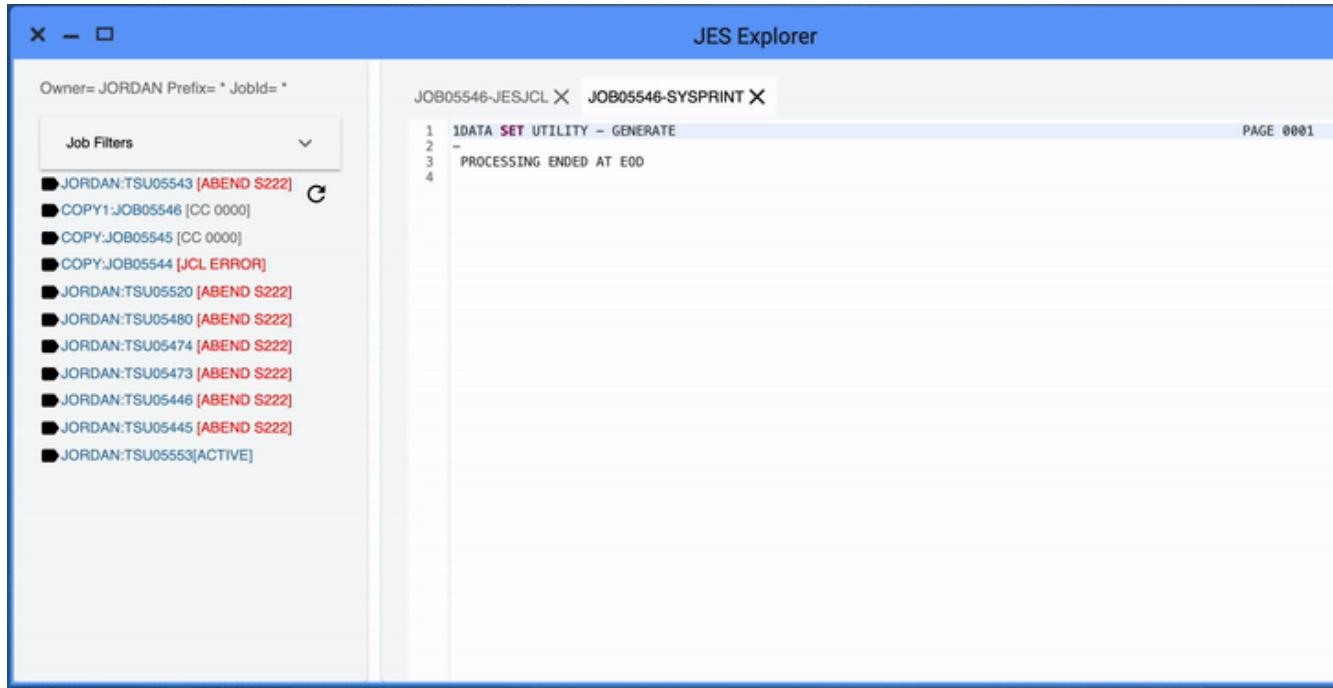
API Mediation Layer

- 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 injectables 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 Extension for Zowe contains various changes in 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](#) on page 108.

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(<${logbackService:-${logbackServiceName}}:>${thread:$PID:-}>){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_294](#).
- 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/EBDCIC 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 80.

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 61 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 sing 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 173.

- [Installing](#) on page 62
- [Issuing your first commands](#) on page 62
- [Using profiles](#) on page 62

- [Writing scripts](#) on page 63
- [Next Steps](#) on page 63

Installing

Software Requirements

Before you install Zowe CLI, download and install Node.js and npm. Use an LTS version of Node.js that is compatible with your version of npm. For a list of compatible versions, see [Node.js Previous Releases](#).

(Linux only): On graphical Linux, install `gnome-keyring` and `libsecret` on your computer before you install the Secure Credential Store. On headless Linux, follow the procedure documented in the [SCS plug-in Readme](#).

Installing Zowe CLI core from public npm

Issue the following commands in sequence to install the core CLI.

The "core" includes Zowe CLI and Secure Credential Store, which enhances security by encrypting your username and password.

```
npm install @zowe/cli@zowe-v1-lts -g
zowe plugins install @zowe/secure-credential-store-for-zowe-cli@zowe-v1-lts
```

Installing CLI plug-ins

```
zowe plugins install @zowe/cics-for-zowe-cli@zowe-v1-lts @zowe/db2-for-zowe-cli@zowe-v1-lts @zowe/ims-for-zowe-cli@zowe-v1-lts @zowe/mq-for-zowe-cli@zowe-v1-lts @zowe/zos-ftp-for-zowe-cli@zowe-v1-lts
```

The command installs most open-source plug-ins, but the IBM Db2 plug-in requires [Installing](#) on page 245.

For more information, see [Installing Zowe CLI plug-ins](#) on page 240.

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 more, see [Using CLI](#).

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](#).

Next Steps

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

- Issue the `zowe --help` command to explore the product functionality, or review the online [web help](#).
- Learn about [using environment variables](#) to store configuration options.
- Learn about [integrating with API Mediation Layer](#).
- Write scripts and integrate them with automation server, such as Jenkins.
- See what [Extending Zowe CLI](#) on page 239 for the CLI.

- Learn about [Developing a new plug-in](#) on page 296 (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 64
- [Zowe CLI FAQ](#) on page 66
- [Zowe Explorer FAQ](#) on page 67

Zowe FAQ

What is Zowe?

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 scriptable command-line interface, extensions to existing REST APIs, and new REST APIs on z/OS.

Who is the target audience for using Zowe?

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?

Zowe consists of several components. The primary languages are Java and JavaScript. Zowe CLI and Desktop are written in TypeScript. ZSS is written in C, while the cross memory server is written in metal C.

What is the licensing for Zowe?

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?

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?

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?

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 70.

To get up and running with the Zowe CLI component quickly, see [Zowe CLI quick start](#) on page 61.

What are the prerequisites for Zowe?

Prerequisites vary by component used, but in most cases the primary prerequisites are Java and NodeJS 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 72.

What's the difference between using Zowe with or without Docker?

Docker is a download option for Zowe that allows you to run certain Zowe server components outside of z/OS. The Docker image contains the Zowe components that do not have the requirement of having to run on z/OS: The App server, API Mediation Layer, and the USS/MVS/JES Explorers.

Configuring components with Docker is similar to the procedures you would follow without Docker, however tasks such as installation and running with Docker are a bit different, as these tasks become Linux oriented, rather than utilizing Jobs and STCs.

NOTE: z/OS is still required when using the Docker image. Depending on which components of Zowe you use, you'll still need to set up z/OS Management Facility as well as Zowe's ZSS and Cross memory servers.

Is the Zowe CLI packaged within the Zowe Docker download?

At this time, the Docker image referred to in this documentation contains only Zowe server components. It is possible to make a Docker image that contains the Zowe CLI, so additional Zowe content, such as the CLI, may have Docker as a distribution option later.

If you are interested in improvements such as this one, please be sure to express that interest to the Zowe community!

How is access security managed on z/OS?

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

How is access to the Zowe open source managed?

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?

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 302 on Zowe Docs.

When will Zowe be completed?

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?

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 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 80 on Zowe Docs.

Zowe CLI FAQ

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

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?

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?

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?

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?

- 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?

- 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?

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 288.

Zowe Explorer FAQ

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

The Zowe Explorer VSCode extension provides developers new to the mainframe with a modern UI, allowing you to access and work with the data set, USS, and job functionalities in a fast and streamlined manner. In addition, Zowe Explorer enables you to work with Zowe CLI profiles and issue TSO/MVS commands.

How can I get started with Zowe Explorer?

First of all, make sure you fulfill the following Zowe Explorer software requirements:

- Get access to z/OSMF.
- Install [Node.js](#) v8.0 or later.
- Install [VSCode](#).
- Configure TSO/E address space services, z/OS data set, file REST interface, and z/OS jobs REST interface. For more information, see [z/OS Requirements](#).

Once the software requirements are fulfilled, create a Zowe Explorer profile.

Follow these steps:

1. Navigate to the explorer tree.
2. Click the + button next to the **DATA SETS**, **USS**, or **JOBS** bar.
3. Select the **Create a New Connection to z/OS** option.
4. Follow the instructions, and enter all required information to complete the profile creation.

You can also watch [Getting Started with Zowe Explorer](#) to understand how to use the basic features of the extension.

Where can I use Zowe Explorer?

You can use Zowe Explorer either in [VSCode](#) or in Theia. For more information about Zowe Explorer in Theia, see [the Theia Readme](#).

How do I get help with using Zowe Explorer?

- Use [the Zowe Explorer channel](#) in Slack to ask the Zowe Explorer community for help.
- Open a question or issue directly in [the Zowe Explorer GitHub repository](#).

How can I use Secure Credential Store with Zowe Explorer?

Activate the Secure Credential Store plug-in in Zowe Explorer.

Follow these steps:

1. Open Zowe Explorer.
2. Navigate to the VSCode settings.
3. Open Zowe Explorer Settings.
4. Add the **Zowe-Plugin** value to the **Zowe Security: Credential Key** entry field.
5. Restart VSCode.
6. Create a profile.

Your Zowe Explorer credentials are now stored securely.

For more information, see [the Enabling Secure Credential Store page](#).

How can I use FTP as my back-end service for Zowe Explorer?

Check out the GitHub article about [the FTP extension](#) with the information on how to build, install, and use FTP as your back-end service for working with Unix files.

How can I contribute to Zowe Explorer?

As a developer, you may contribute to Zowe Explorer in the following ways:

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

Note: For more information, see [Extending Zowe Explorer](#).

Zowe resources

Learn more about Zowe from these blog posts, videos, and other resources.

Blogs

- [Zowe blogs on Medium](#)
- [Zowe blogs on Open Mainframe Project website](#)

Details for how to contribute to the [Zowe blogs on Medium](#) site are at [Zowe Blog Guidelines](#).

Videos, webinars

As well as Zowe videos owned and managed by the community, there are a number of external youtubers who host Zowe related content.

- [Zowe Demos playlist from Bill Pereira](#)
- [Mainframe Bytes channel from Jessielaine Punongbayan](#)

Community

Join us on Slack

- [Slack invite link](#)
- [Zowe Slack channels](#)

Learn more about the community

Find out information about Zowe sub-projects, GitHub repos, mailing lists, community meeting minutes, contribution guidelines, and so on.

- [Zowe community GitHub repo](#)

Chapter

2

User Guide

Topics:

- Planning and preparing the installation
- Installing Zowe z/OS components
- Installing Zowe Docker Bundle
- Installing Zowe CLI
- Advanced Zowe configuration
- Using Zowe
- Zowe CLI extensions and plugins
- Zowe Explorer

Planning and preparing the installation

Introduction

The installation of Zowe™ consists of two independent processes: installing the Zowe server components either entirely on z/OS or a combination of z/OS and Docker, and then installing Zowe CLI on a desktop computer.

The Zowe server components provide a web desktop that runs in a web browser providing a number of applications for z/OS users, together with an API mediation layer provides single-sign on (SSO), organization of the multiple zowe servers under a single website, and other 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.

Because Zowe is a set of components, before installing Zowe, first determine which components you want to install and where you want to install them. This guide provides documentation for all of the components and it is split into sections so you can install as much as you need.

Here are some scenarios to consider:

- If you will only be accessing the Zowe server components through a web browser or REST API client, then you do not need to install the Zowe CLI.
- If you will only be using the Zowe CLI, depending on the plugins used you may not need to install the Zowe server components.
- If you intend to use Docker for the server components, less components need to be installed on z/OS. If you are not using the Desktop or ZSS, then it's possible run the other Zowe components without installing any of Zowe onto z/OS.

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

Planning the installation of Zowe server components

All Zowe server components can be installed on z/OS, but some have the alternative option of being run inside of a Docker image on a Linux host. Which option you choose effects the prerequisites, where they are installed, and the installation steps needed.

Planning z/OS installation

If you are installing one or more server components onto z/OS, the following information is required during the installation process. Software and hardware prerequisites are covered in the next section.

- The zFS directory where you will install the Zowe runtime files and folders. For more details of setting up and configuring the UNIX Systems Services (USS) environment, see [UNIX System Services considerations for Zowe](#) on page 97.
- 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.
- (If not using Docker) 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.

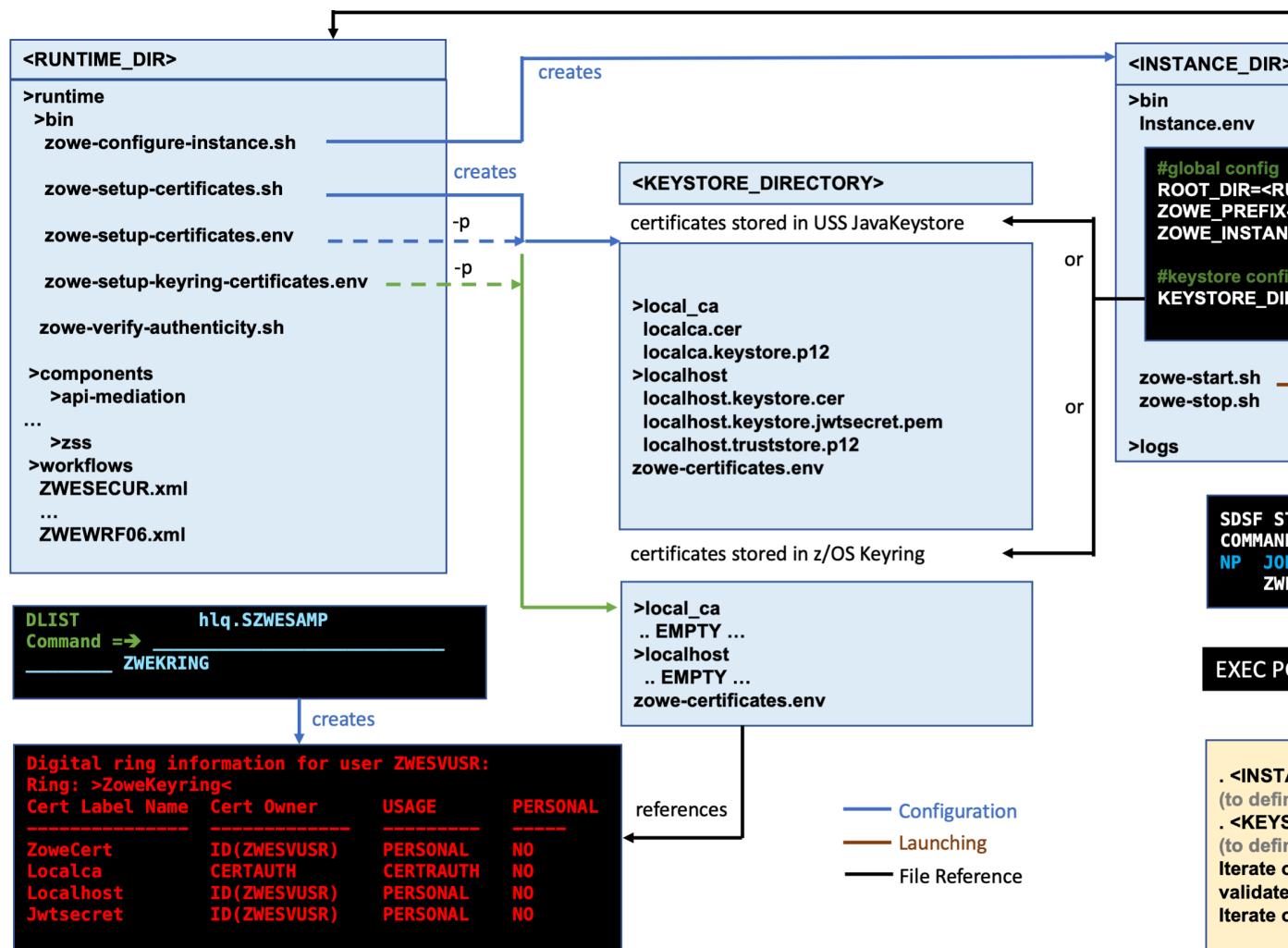
- Zowe has two started tasks.
 - ZWESISTC is a cross memory server that the Zowe desktop uses to perform APF-authorized code. More details on the cross memory server are described in [Installing and configuring the Zowe cross memory server \(ZWESISTC\) on page 146](#).
 - ZWEVSTC brings up the every other other part of the Zowe runtime on z/OS that was requested. This may include Desktop, API mediation layer, ZSS, and more, but when using Docker likely only ZSS will be used here.

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 126 and a sample JCL member ZWESECUR is shipped with Zowe that contains commands for RACF, TopSecret, and ACF2 security managers.

Note: To start the API Mediation Layer as a standalone component, see [API Mediation Layer as a standalone component](#) on page 126

Topology of the Zowe z/OS launch process

The following diagram depicts the high-level structure of a Zowe installation and runtime.



RUNTIME_DIR

The runtime directory **<RUNTIME_DIR>** contains the binaries and executable files. You can create a runtime directory in one of the following ways:

- Executing the `zowe-install.sh` script contained within the `install` directory of a Zowe convenience build.
- Installing the Zowe SMP/E FMID AZWE001 using the JCL members in the REL4 member.
- Executing the z/OSMF workflow script `ZWERF01` contained in the SMP/E FMID AZWE001.

During execution of Zowe, the runtime directory contents are not modified. Maintenance or APAR release for Zowe replaces the contents of the runtime directory and are rollup PTFs.

INSTANCE_DIR

The instance directory `<INSTANCE_DIR>` is required to launch Zowe. It is created with the script `<RUNTIME_DIR>/bin/zowe-configure-instance.sh`. More than one instance directory can be created and used to launch multiple instances of Zowe sharing the same runtime directory `<RUNTIME_DIR>`.

Zowe instances are started by running the script `<INSTANCE_DIR>/bin/zowe-start.sh`. This creates a started task with the PROCLIB member `ZWESVSTC` that is provided with the samplib `SZWESAMP` created during the installation of Zowe. The JCL member `ZWESVSTC` starts a USS shell under which it launches its address spaces.

The instance directory file `instance.env` is used to configure a Zowe launchable. The file is executed during the launch of Zowe and specifies shell variables such as ports and location of dependent directories and services on z/OS.

The `instance.env` file sets the location of the `<RUNTIME_DIRECTORY>` as well as the `<KEYSTORE_DIRECTORY>`

KEYSTORE_DIRECTORY

Zowe uses certificates to encrypt data as well as a truststore. The keystore directory `<KEYSTORE_DIRECTORY>` controls where the certificates are located, either in a JavaKeystore or a z/OS keyring. A `<KEYSTORE_DIRECTORY>` is created by using the script `<RUNTIME_DIR>/bin/zowe-setup-certificates.sh`.

System requirements

Before installing ZoweTM, ensure that your environment meets the prerequisites.

- [z/OS system requirements \(host\)](#) on page 73
 - [Zowe API Mediation Layer on z/OS requirements \(host\)](#) on page 73
 - [Zowe Web Explorers and APIs on z/OS requirements \(host\)](#) on page 73
 - [Zowe Application Framework on z/OS requirements \(host\)](#) on page 73
 - [Important note for users upgrading to v1.14](#) on page 74
- [Docker requirements \(host\)](#) on page 74
- [Zowe Desktop requirements \(client\)](#) on page 74
- [Zowe CLI requirements](#) on page 75
 - [Client-side](#) on page 75
 - [Host-side](#) on page 75
 - [Free disk space](#) on page 75
- [Feature requirements](#) on page 75
 - [Multi-Factor Authentication \(MFA\)](#) on page 75
 - [Single Sign-On \(SSO\)](#) on page 76

Please note: Not all of the prerequisites below are needed. The prerequisites needed depends on what components you want to use.

Zowe CLI operates independently of the Zowe z/OS component and is installed on a client PC that runs Windows, Linux, or Mac operating systems. It can access z/OS endpoints such as z/OSMF, or can access FTP, CICS, DB2, and other z/OS services through plug-ins. Unless required by plug-ins, the Zowe CLI does not require the Zowe server components to be installed.

All Zowe server components can be installed on a z/OS environment, while some can alternatively be installed on Linux or zLinux via Docker. The components provide a number of services that are accessed through a web browser such as an API catalog and a web desktop. The client PC that accesses the Zowe server components does not need to have the Zowe CLI installed.

Even though the Zowe server components do not require the CLI and vice-versa, there is an advantage to having the server components if using the CLI. When installed, the API Mediation Layer of Zowe can provide benefits to the CLI such as single-sign-on and only needing to trust a single certificate when using multiple Zowe-related endpoints.

For more information on the relationship between the Zowe server components and Zowe CLI, see [Zowe overview](#) on page 8.

z/OS system requirements (host)

- z/OS version in active support, such as Version 2.3 and Version 2.4

Note: z/OS V2.2 reaches end of support on 30 September 2020. For more information, see the z/OS v2.2 lifecycle details <https://www.ibm.com/support/lifecycle/details?q45=Z497063S01245B61>. Zowe Version 1.15 and earlier can continue to work with z/OS V2.2 but you are advised to upgrade your z/OS to more recent versions.

- zFS volume with at least 833mb of free space for Zowe server components, their keystore, instance configuration files and logs, and third party plugins. **Requirement for:** Zowe Server Components (API Mediation Layer, Application Framework, ZSS)
- (Optional, recommended) IBM z/OS Management Facility (z/OSMF) Version 2.2, Version 2.3 or Version 2.4. While z/OSMF is optional for Zowe, many components utilize it and therefore it is recommended that z/OSMF is present to fully exploit Zowe's capabilities.

When utilizing z/OSMF with Zowe ensure the [z/OSMF JWT Support is available via APAR and associated PTFs](#).

- (Optional, recommended) z/OS OpenSSH V2.2.0 or higher. Some features of Zowe require SSH, such as the Desktop's SSH terminal. Some users may also find it convenient to install & manage Zowe via SSH, as an alternative to OMVS over TN3270.

::: 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 startup 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 80.
- For production use of Zowe, see [Configuring z/OSMF](#) on page 77. :::

Zowe API Mediation Layer on z/OS requirements (host)

- IBM SDK for Java Technology Edition V8 or later

Zowe Web Explorers and APIs on z/OS requirements (host)

- Node.js v8.x (except v8.16.1), v12.x, or v14

Note: When using Node.js v12.x or later, it is highly recommended that plug-ins used are tagged. For more information, see [Tagging on z/OS](#).

To install Node.js on z/OS, follow the instructions in [Installing Node.js on z/OS](#) on page 76.

- IBM SDK for Java Technology Edition V8 or later

Zowe Application Framework on z/OS requirements (host)

The Zowe Application Framework server provides the Zowe Desktop that contains an extensible GUI with a number of applications allowing access to z/OS functions, such as the File Editor, TN3270 emulator, JES Explorer, and more. For more information, see [Zowe Architecture](#).

- Node.js v8.x (except v8.16.1), v12.x, or v14

Note: When using Node.js v12.x or later, it is highly recommended that plug-ins used are tagged. For more information, see [Tagging on z/OS](#).

To install Node.js on z/OS, follow the instructions in [Installing Node.js on z/OS](#) on page 76.

- IBM SDK for Java Technology Edition V8 or later

Important note for users upgrading to v1.14

If you are upgrading to Zowe v1.14 from a previous release, and the value of `ZOWE_EXPLORER_HOST` does not match the host and domain that you put into your browser to access Zowe, you must update your configuration due to updated referrer-based security.

To configure your system for the version 1.14 update, perform **one** of the following tasks:

- Define `ZWE_EXTERNAL_HOSTS` as a comma-separated list of hosts from which you would access Zowe from your browser.
 - `ZWE_EXTERNAL_HOSTS=host1,host2,...`
- Define `ZWE_REFERRER_HOSTS` as a comma-separated list for the value to be applied specifically for referrer purposes.
 - `ZWE_REFERRER_HOSTS=host1,host2,...`

See [Reviewing the instance.env file](#) on page 151 for additional information on the use of `instance.env` files.

See [Configuring a Zowe instance via instance.env file](#) for additional information on configuring `instance.env` files.

Docker requirements (host)

The Zowe Docker build is a technical preview.

Docker is a technology for delivering a set of software and all its prerequisites and run them in an isolated manner to reduce installation steps and to eliminate troubleshooting environmental differences. Docker can run on many operating systems, but currently the Zowe Docker image is for x86 Linux (Intel, AMD) and zLinux ("s390x"). Support for platforms such as zCX, Windows, and more will be added over time.

To get Docker for Linux, you should check your Linux software repository. Whether using Ubuntu, Red Hat, SuSE, and many other types of Linux, you can install Docker the same way you install other software on Linux through the package manager.

Once you have Docker, the Zowe Docker image has the following requirements

- 4 GB free RAM, 8 GB recommended
- 4 GB free disk space
- Network access to the z/OS host. The Linux host must be able to communicate with the z/OS host.

When using Docker, z/OS is still required and if using the Application Framework or ZSS, installing ZSS on the z/OS host is still required. See these requirements:

- [z/OS system requirements \(host\)](#) on page 73

Note: The subsections of z/OS requirements such as for API Mediation Layer, Web Explorers, and Application Framework are not required because they are included in the Docker install.

Zowe Desktop requirements (client)

The Zowe Desktop is powered by the Application Framework which has server prereqs depending on where it is installed

- [Application Framework on z/OS prerequisites](#)
- [Docker requirements \(host\)](#) on page 74

The Zowe Desktop runs inside of a browser. No browser extensions or plugins are required. The Zowe Desktop supports Google Chrome, Mozilla Firefox, Apple Safari and Microsoft Edge releases that are at most 1 year old,

except when the newest release is older. For Firefox, both the regular and Extended Support Release (ESR) versions are supported under this rule.

Currently, the following browsers are supported: - Google Chrome V79 or later - Mozilla Firefox V68 or later - Safari V13 or later - Microsoft Edge 79

If you do not see your browser listed here, please contact the Zowe community so that it can be validated and included.

Zowe CLI requirements

Client-side

Zowe CLI is supported on Windows, Linux, and Mac operating systems. Meet the following requirements before you install the CLI:

- **Node.js:** Install a currently supported Node.js LTS version. For an up-to-date list of supported LTS versions, see [Nodejs.org](#).
 ::: 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. :::
- **npm:** Install a version of Node Package Manager (npm) that is compatible with your version of Node.js.
 ::: tip Npm is included with most Node.js installations. Issue the command `npm --version` to check your current version. You can reference the [Node.js release matrix](#) to verify that the versions are compatible. :::
- **Plug-in client requirements:** If you plan to install plug-ins, review the [Software requirements for Zowe CLI plug-ins](#) on page 239. You *must* meet the client requirements for the Secure Credential Store and IBM Db2 plug-ins prior to installing them.

Host-side

Zowe CLI requires the following mainframe configuration:

- **IBM z/OSMF configured and running:** You do not need to install the full Zowe solution to install and use Zowe CLI. Minimally, an instance of IBM z/OSMF must be running on the mainframe before you can issue Zowe CLI commands successfully. z/OSMF enables the core capabilities such as retrieving data sets, executing TSO commands, submitting jobs, and more. If Zowe API Mediation Layer (API ML) is configured and running, CLI users can choose to connect to API ML rather than to every separate service.
- **Plug-in services configured and running:** Plug-ins communicate with various mainframe services, which must be configured and running on the mainframe prior to issuing plug-in commands. For example, the IMS plug-in requires an instance of IBM IMS on the mainframe with IMS Connect (REST services) running. For more information, see [Software requirements for Zowe CLI plug-ins](#) on page 239
- **Zowe CLI on z/OS is not supported:** Zowe CLI can be installed on an IBM z/OS environment and run under Unix System Services (USS). However, the IBM Db2 and Secure Credentials Store plug-ins will *not* run on z/OS due to native code requirements. As such, Zowe CLI is *not supported on z/OS* and is currently experimental.

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 your operating system, the plug-ins that you install, and user profiles saved to disk.

Feature requirements

Zowe has several optional features that have additional prerequisites as follows.

Multi-Factor Authentication (MFA)

Multi-factor authentication is supported for several components, such as the Desktop and API Mediation Layer. Multi-factor authentication is provided by third party products which Zowe is compatible with. The following are known to work:

- [IBM Z Multi-Factor Authentication](#).

For information on using MFA in Zowe, see [Multi-factor authentication configuration](#) on page 191.

Single Sign-On (SSO)

Zowe has an SSO scheme with the goal that each time you use multiple Zowe components you should only be prompted to login once.

Requirements:

- IBM z/OS Management Facility (z/OSMF)
- (Optional, recommended) PKCS#11 token setup is required when using ZSS, the Desktop, and Application Framework with SSO. See [Creating a PKCS#11 Token](#) for more information.

Installing Node.js on z/OS

Note: This section is not required if using Docker or only using the CLI.

Before you install Zowe™ on z/OS, you must install IBM SDK for Node.js on the same z/OS server that hosts the Zowe Application Server and z/OS Explorer Services. Review the information in this topic to obtain and install Node.js.

- [Supported Node.js versions](#) on page 76
- [How to obtain IBM SDK for Node.js - z/OS](#) on page 76
- [Hardware and software prerequisites](#) on page 77
- [Installing the PAX edition of Node.js - z/OS](#) on page 77
- [Installing the SMP/E edition of Node.js - z/OS](#) on page 77

Supported Node.js versions

The following Node.js versions are supported to run Zowe. See the [Hardware and software prerequisites](#) on page 77 section for the prerequisites that are required by Zowe.

The corresponding [IBM Knowledge Center for Node.js - z/OS](#) lists all the prerequisites for Node.js. Some software packages, which might be listed as prerequisites there, are **NOT** required by Zowe. Specifically, you do **NOT** need to install Python, Make, Perl, or C/C++ runtime or compiler. If you can run `node --version` successfully, you have installed the prerequisites required by Zowe.

- v8.x (except v8.16.1)
 - z/OS V2R2: PTFs UI62788, UI46658, UI62416, UI62415 (APARs [PH10606](#), [PI79959](#), [PH10740](#), [PH10741](#))
 - z/OS V2R3: PTFs UI61308, UI61375, UI61747 (APARs [PH0710](#), [PH08352](#), [PH09543](#))
 - z/OS V2R4: PTFs UI64839, UI64940, UI64837, UI64830 (APARs [PH14559](#), [PH16038](#), [PH15674](#), [PH14560](#))

Known issue: There is a known issue with node.js v8.16.1 and Zowe desktop encoding. See the [GitHub issue](#) for details.

Workaround: Use node.js v8.16.2 or later, which is available at <https://www.ibm.com/ca-en/marketplace/sdk-nodejs-compiler-zos>. Download the `pax.z` file.

- v12.x
 - z/OS V2R2: PTFs UI62788, UI46658, UI62416, UI62415 (APARs [PH10606](#), [PI79959](#), [PH10740](#), [PH10741](#))
 - z/OS V2R3: PTFs UI61308, UI61375, UI61747 (APARs [PH0710](#), [PH08352](#), [PH09543](#))
 - z/OS V2R4: PTFs UI64839, UI64940, UI64837, UI64830, UI65567 (APARs [PH14559](#), [PH16038](#), [PH15674](#), [PH14560](#), [PH17481](#))
- v14
 - z/OS V2R3: PTFs UI61308, UI61375, UI61747 (APARs [PH07107](#), [PH08352](#), [PH09543](#))
 - z/OS V2R4: PTFs UI64830, UI64837, UI64839, UI64940, UI65567 (APARs [PH14560](#), [PH15674](#), [PH14559](#), [PH16038](#), [PH17481](#))

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 edition through your IBM representative for production use

- Use the PAX edition for non-production deployments

For more information, see the blog "[How to obtain IBM SDK for Node.js - z/OS, at no charge](#)".

Hardware and software prerequisites

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

The corresponding [IBM Knowledge Center for Node.js - z/OS](#) lists all the prerequisites for Node.js. Some software packages, which might be listed as prerequisites there, are **NOT** required by Zowe. Specifically, you do **NOT** need to install Python, Make, Perl, or C/C++ runtime or compiler.

If you can run `node --version` successfully, you have installed the Node.js prerequisites required by Zowe.

Hardware:

IBM zEnterprise® 196 (z196) or newer

Software:

- z/OS UNIX System Services enabled
- Integrated Cryptographic Service Facility (ICSF) configured and started

ICSF is required for Node.js to operate successfully on z/OS. If you have not configured your z/OS environment for ICSF, see [Cryptographic Services ICSF: System Programmer's Guide](#). To see whether ICSF has been started, check whether the started task ICSF or CSF is active.

Installing the PAX edition of Node.js - z/OS

Follow these steps to install the PAX edition of Node.js - z/OS to run Zowe.

1. Download the pax.Z file 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/install/node-v6.14.4-os390-s390x`.

Installing the SMP/E edition of Node.js - z/OS

To install the SMP/E edition 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 for z/OSMF configuration

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

Requirements	Description	Resources in IBM Knowledge Center
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:

- 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 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 `zosmf.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 77](#).

1. [Introduction on page 81](#)
 2. [Assumptions on page 81](#)
 3. [Software Requirements on page 82](#)
 - a. [Minimum Java level on page 82](#)
 - b. [WebSphere® Liberty profile \(z/OSMF V2R3 and later\)](#)
 - c. [System settings on page 83](#)
 - d. [Web browser on page 83](#)
 4. [Creating a z/OSMF nucleus on your system](#)
 - a. [Running job IZUNUSEC to create security on page 83](#)
 - b. [Running job IZUMKFS to create the z/OSMF user file system](#)
 - c. [Copying the IBM procedures into JES PROCLIB on page 87](#)
 - 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 90](#)
 - a. [Enabling the z/OSMF JOB REST services](#)
 - b. [Enabling the TSO REST services on page 91](#)
 - 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 94](#)
 - a. [Common problems and scenarios on page 94](#)
 - b. [Tools and techniques for troubleshooting on page 94](#)
- [Appendix A. Creating an IZUPRMxx parmlib member on page 95](#)
 - [Appendix B. Modifying IZUSVR1 settings on page 96](#)
 - [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 95.

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 87.

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 82
- [WebSphere® Liberty profile \(z/OSMF V2R3 and later\)](#)
- [System settings](#) on page 83
- [Web browser](#) on page 83

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 95.

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 96.

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 94 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 83
- [Running job IZUMKFS to create the z/OSMF user file system](#)
- [Copying the IBM procedures into JES PROCLIB](#) on page 87
- [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 94.
ADDDGROUP 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 AUTOUID.

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.

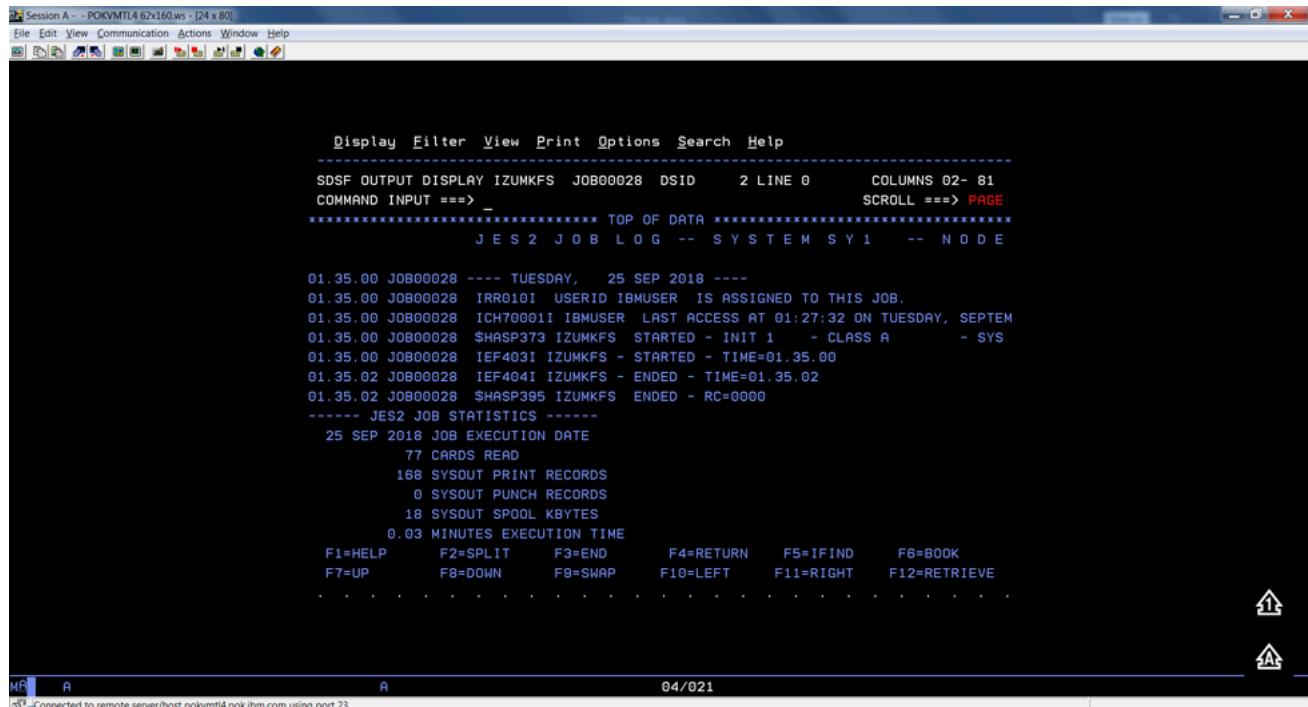
Results

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:



The image contains two side-by-side screenshots of a z/OS SDSF session window. Both windows have a title bar 'Session A - POKVMTI4 62x160.ws [24x80]' and a menu bar 'File Edit View Communication Actions Window Help'.

Screenshot 1 (Top):

```

Display Filter View Print Options Search Help
-----
SDSF OUTPUT DISPLAY IZUMKFS JOB000028 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
10E200077I HFS-compatibility aggregate IZU.SIZUUSRD has been successfully created
F1=HELP F2=SPLIT F3=END F4=RETURN F5=IFIND F6=BOOK
F7=UP F8=DOWN F9=SWAP F10=LEFT F11=RIGHT F12=RETRIEVE

```

Screenshot 2 (Bottom):

```

Display Filter View Print Options Search Help
-----
SDSF OUTPUT DISPLAY IZUMKFS JOB000028 DSID 107 LINE 1 COLUMNS 02- 81
COMMAND INPUT ===> _ SCROLL ==> PAGE
READY
BPXBATCH SH mkdir -p /global/zosmf
READY
MOUNT FILESYSTEM('IZU.SIZUUSRD') TYPE(ZFS) MOUNTPOINT('/global/zosmf') MODE(
READY
BPXBATCH SH mkdir -p /global/zosmf/data/home/izusvr
READY
BPXBATCH SH mkdir -p /global/zosmf/configuration/workflow
READY
BPXBATCH SH chown -R IZUSVR:IZUADMIN /global/zosmf
READY
BPXBATCH SH chmod -R 755 /global/zosmf
READY
END
***** BOTTOM OF DATA *****

F1=HELP F2=SPLIT F3=END F4=RETURN F5=IFIND F6=BOOK
F7=UP F8=DOWN F9=SWAP F10=LEFT F11=RIGHT F12=RETRIEVE

```

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 .

Symptom	Cause	Resolution
Job fails with an authorization error.	Job statement errors.	See Troubleshooting problems on page 94.

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:

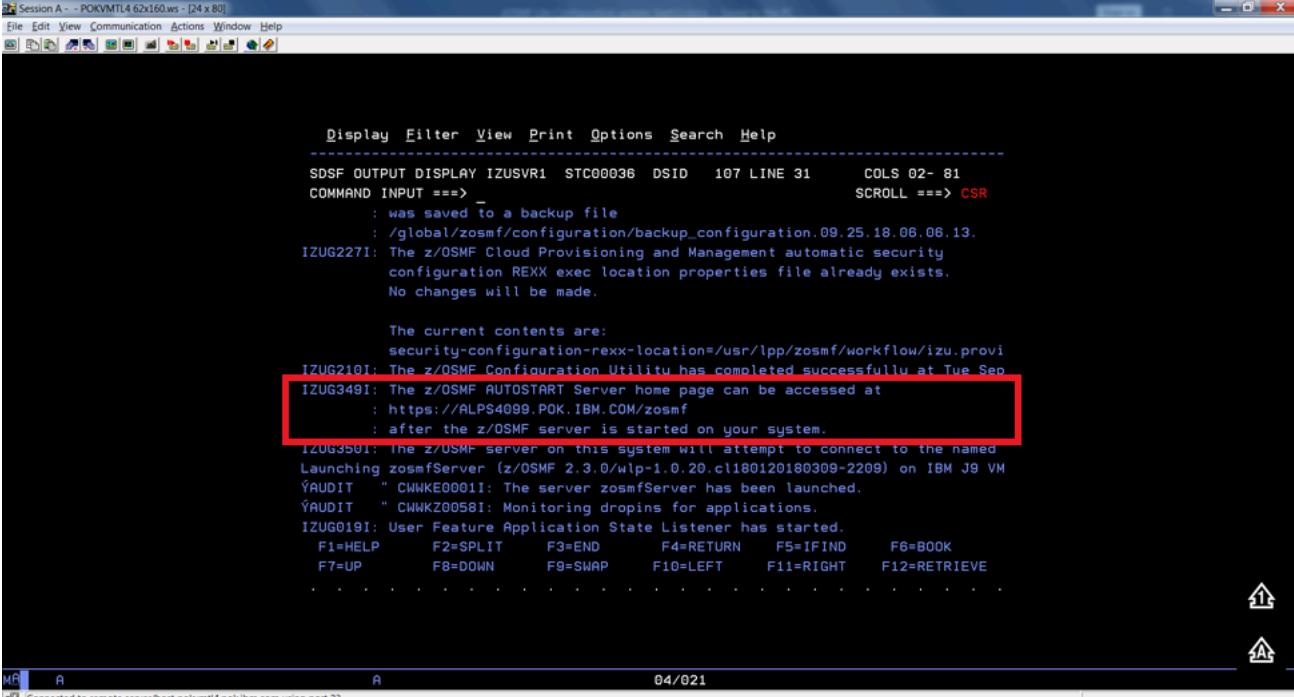
```
CWWKB0069I: INITIALIZATION IS COMPLETE FOR THE IZUANG1 ANGEL PROCESS.
IZUG400I: The z/OSMF Web application services are initialized.
CWWKF0011I: The server zosmfServer is ready to run a smarter planet.
```

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.



```
Session A - - POKVMTL4 62x160.ws - [24 x 80]
File Edit View Communication Actions Window Help
File Explorer Task List Application Icons

Display Filter View Print Options Search Help
-----
SDSF OUTPUT DISPLAY IZUSVR1 STC00036 DSID 107 LINE 31      COLS 02- 81
COMMAND INPUT ===>                                     SCROLL ==> CSR
: was saved to a backup file
: /global/zosmf/configuration/backup_configuration.09.25.18.06.06.13.
IZUG227I: The z/OSMF Cloud Provisioning and Management automatic security
configuration REXX exec location properties file already exists.
No changes will be made.

The current contents are:
security-configuration-rexx-location=/usr/lpp/zosmf/workflow/izu.provi
IZUG210I: The z/OSMF Configuration Utilty has completed successfully at Tue Sep
IZUG349I: The z/OSMF AUTOSTART Server home page can be accessed at
: https://ALPS4099.POK.IBM.COM/zosmf
: after the z/OSMF server is started on your system.

IZUG350I: The z/OSMF server on this system will attempt to connect to the named
Launching zosmfServer (z/OSMF 2.3.0/wlp-1.0.20.c1180120180309-2209) on IBM J9 VM
SYAUDIT " CWNKE0001I: The server zosmfServer has been launched.
SYAUDIT " CWNKZ0058I: Monitoring dropins for applications.
IZUG019I: User Feature Application State Listener has started.
F1=HELP   F2=SPLIT   F3=END      F4=RETURN   F5=IFIND   F6=BOOK
F7=UP     F8=DOWN    F9=SWAP     F10=LEFT    F11=RIGHT  F12=RETRIEVE
A          A          A          A          A          A
04/021
Connected to remote server/host pokvmtl4.pok.ibm.com using port 23
```

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/`

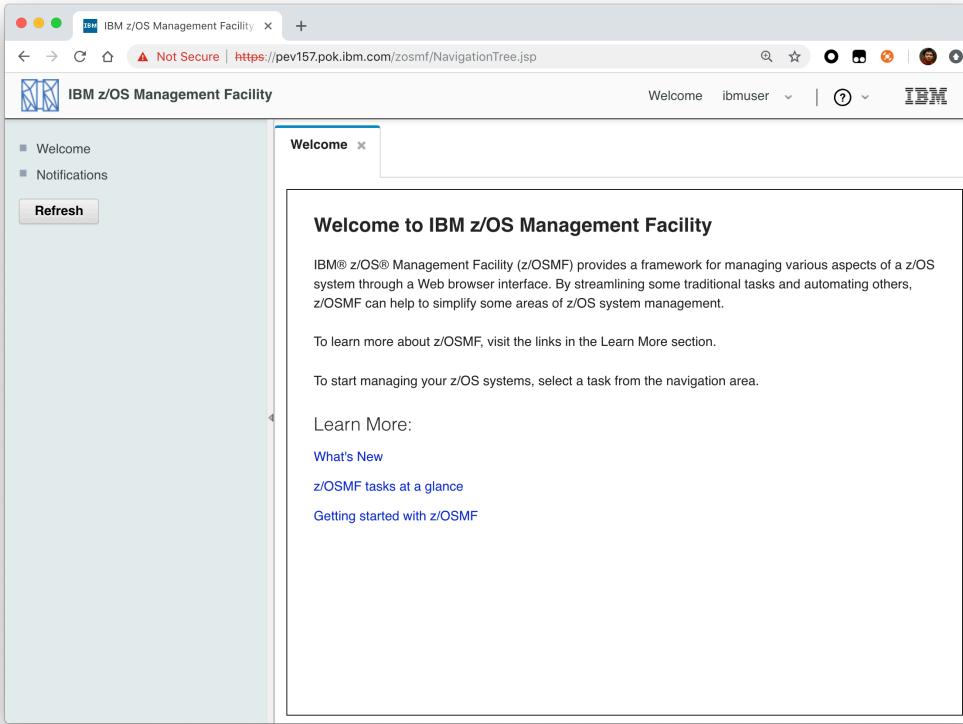
Where:

- *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 91
- [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/rest/jobs/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.isuzuUsers 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 95.

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.
```

```
BXP014I 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 96.

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(<i>hostname</i>)	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(nn)	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(<i>account-number</i>) REGION(<i>region-size</i>) PROC(<i>proc-name</i>)	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 <i>z/OSMF Configuration Guide</i> .	Default: 443 ACCT(IZUACCT) REGION(50000) PROC(IZUFPROC)
USER_DIR= <i>filepath</i>	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= <i>filepath</i>	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.

UNIX System Services considerations for Zowe

The Zowe z/OS component runtime requires USS to be configured. As shown in the [Zowe architecture](#) on page 13, a number of servers run under UNIX System Services (USS) on z/OS. Review this topic for knowledge and considerations about USS when you install and configure Zowe.

- [Introduction](#)
- [Setting up USS for the first time](#) on page 98
- [Language environment](#) on page 98
- [OMVS segment](#) on page 98
- [Address space region size](#) on page 98

What is USS?

The UNIX System Services element of z/OS® is a UNIX operating environment, which is implemented within the z/OS operating system. It is also known as z/OS UNIX. z/OS UNIX files are organized in a hierarchy, as in a UNIX system. All files are members of a directory, and each directory in turn is a member of another directory at a higher level in the hierarchy. The highest level of the hierarchy is the *root* directory. The z/OS UNIX file system is also known as zFS.

For more information on USS, see the following resources:

- [Introduction to z/OS UNIX for z/OS 2.2](#)
- [Introduction to z/OS UNIX for z/OS 2.3](#)
- [Introduction to z/OS UNIX for z/OS 2.4](#)

Setting up USS for the first time

If you have not enabled USS for your z/OS environment before, the Zowe SMP/E distribution of Zowe provides a number of JCL jobs to assist with this purpose.

Language environment

To ensure that Zowe has enough memory, the recommended HEAP64 site should be large enough.

```
HEAP64(512M,4M,KEEP,256M,4M,KEEP,OK,FREE)
```

OMVS segment

Users who install Zowe to run Zowe scripts need to have an OMVS segment. If the user profile doesn't have OMVS segment, the following situations might occur:

- When you access USS through TSO OMVS, you will see the following message:

```
FSUM2057I No session was started. This TSO/E user ID does not have access
to OpenMVS.+
FSUM2058I Function = sigprocmask, return value = FFFFFFFF, return code =
0000009C, reason code = 0B0C00FB
```

Action: Create an OMVS segment with a UID.

- When you access USS through SSH, you will see the following message:

```
Access denied with SSH
```

Address space region size

Java as a prerequisite for Zowe requires a suitable z/OS region size to operate successfully while you install and configure Zowe. It is suggested that you do not restrict the region size, but allow Java to use what is necessary. Restricting the region size might cause failures with storage-related error messages such as the following one:

```
JVMJ9VM015W Initialization error for library j9gc29(2)
Error: Could not create the Java Virtual Machine.
Error: A fatal exception has occurred. Program will exit
```

You can fix the storage-related issue by making one of the following changes:

- ASSIZEMAX parameter

The ASSIZEMAX parameter is the maximum size of the process's virtual memory (address space) in bytes.

To specify the JVM maximum address space size on a per-user basis, set the ASSIZEMAX configuration parameter to the value of 2147483647.

Note: Running a shell script via TSO OMVS will run the shell in the TSO address space, unless you specify _BPX_SHAREAS=NO when invoking OMVS. If you are using TSO OMVS to install Zowe, you will need export _BPX_SHAREAS=NO to make the ASSIZEMAX change effective.

- SIZE parameter of TSO segment

Set SIZE operand of TSO segment to the value of 2096128.

Note: If you set export _BPX_SHAREAS=YES in your shell setup as recommended, Java will run in the TSO address space and the SIZE change will work.

- ulimit -A

The maximum address space size for the process should be at least 250 M, in units of 1024 bytes. For example, ulimit -A 250000.

Note: Running ulimit -a displays the current process limits.

Installing Zowe z/OS components

z/OS Installation Roadmap

There are two parts to installing ZoweTM on z/OS.

The first part is the Zowe runtime, consisting of the following components:

- Zowe Application Framework
- z/OS Explorer Services
- Zowe API Mediation Layer
- ZSS

The second part is the Zowe Cross Memory Server, an authorized server application that provides privileged services to Zowe in a secure manner.

If you want to use Docker, instead follow this related page: [Docker Installation Roadmap \(Technical Preview\)](#) on page 166.

For more information on the Zowe components and how they are used to launch an instance of Zowe, see [Planning the installation](#).

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

SMP/E build

What is your preferred installation method?

Conv

Download the Zowe SMP/E build

Shell script

Choose a method to install the SMP/E build

z/OSMF workflow

Install the Zowe SMP/E build using JCLs

Install the Zowe SMP/E build with z/OSMF workflow
ZWEWRF01

Shell script

Run shell script
zowe-install

Stage 3:
Configure the Zowe runtime and start Zowe

Choose a method to configure Zowe

JCL, Shell scripts

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 70 for details.

Stage 2: Install the Zowe z/OS runtime

1. Ensure that the software requirements are met. The prerequisites are described in [System requirements](#) on page 72.
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.

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 102.
- For how to obtain the SMP/E build and install it, see [Installing Zowe SMP/E](#) on page 108.

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 SZWEAAUTH 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 z/OS runtime

You can configure the Zowe runtime with one of the following methods depending on your needs.

- Use JCL and shell scripts
- Use z/OSMF Workflows

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 126 and [Configure Zowe with z/OSMF Workflows](#) on page 157.

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 TLS. For instructions, see [Configuring Zowe certificates](#) on page 136 and [Configure Zowe with z/OSMF Workflows](#) on page 157.

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. Configure and start the ZWESISTC cross memory server and install the load libraries. For instructions, see [Installing and configuring the Zowe cross memory server \(ZWESISTC\)](#) on page 146 and [Configure Zowe Cross Memory Server](#) on page 158.
4. 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 150 and [Configure Zowe with z/OSMF Workflows](#) on page 157.

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 APPF-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 SZWEAAUTH.

Note

For more information about Gateway and Discovery Service parameters that can be set during the Zowe runtime configuration, see [Advanced Gateway features configuration](#) on page 196 and [Discovery Service runtime configuration parameters](#).

5. Configure and start the ZWESVSTC started task. For instructions, see [Installing and starting the Zowe started task \(ZWESVSTC\)](#) on page 156.

Stage 4: Verify the installation

Verify that Zowe is installed correctly on z/OS. See [Verifying Zowe installation on z/OS](#) on page 161.

Looking for troubleshooting help?

If you encounter unexpected behavior when installing or verifying the Zowe runtime on z/OS, see the [Troubleshooting](#) on page 400 section for tips.

Installing Zowe runtime from a convenience build

You install the Zowe™ convenience build by obtaining a PAX file for a build and using this to create the Zowe runtime environment.

After you [Obtaining and preparing the convenience build](#) on page 103, you can take the following steps to complete the installation.

- [Step 1: Locate the install directory](#) on page 105
- [Step 2: Choose a runtime USS folder](#) on page 105
- [Step 3: Choose a dataset HLQ for the SAMPLIB and LOADLIB](#) on page 105
- [Step 4 \(Method 1\): Install the Zowe runtime using shell script](#) on page 106
- [Step 4 \(Method 2\): Install the Zowe runtime using z/OSMF Workflow](#) on page 106

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 that is named `zowe-V.v.p.pax`, where

- V indicates the Major Version
- v indicates the Minor Version
- p indicates the Patch Version

The numbers are incremented each time a release is created, so the higher the numbers, the later the release. For more information about the Zowe release number, see [Understanding the Zowe release](#) on page 400.

To download the PAX file, open your web browser and click the **Zowe z/OS Convenience build** button on the [Zowe Download](#) website to save it to a folder on your desktop.

After you have the `zowe-V.v.p.PAX` file, follow these steps.

1. **(Optional)** Verify the integrity of the PAX file to ensure that the file you download is officially distributed by the Zowe project. This step is only needed if you are unsure of the provenance of the PAX file and want to ensure that it is an original Zowe release driver.

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.v.p 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.v.p>.pax
```

Where `zowe-V.v.p` 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 `l ls` 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.v.p>.pax
```

Where `zowe-V.v.p` 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.v.p>.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 storage requirements are included here.

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

The SZWESAMP data set contains 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.
ZWEKRING	JCL member to configure a z/OS keyring containing the Zowe certificate
ZWENOKYR	JCL member to undo the configuration steps performed in ZWEKRING
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 Auxiliary server

Member name	Purpose
ZWESIPRG	Console commands to APF authorize the cross memory server load library
ZWESISCH	PPT entries required by Cross memory server and its Auxiliary address spaces to run in Key(4)

The SZWEAAUTH 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 3a: Choose a log directory (optional)

By default, during installation and configuration, various logs will be created in `/global/zowe/logs` if it is writable, or `~/zowe/logs`. If neither of these directories exists, or is writable by the user who installs Zowe, or you want to override and provide your own directory that contains logs, you can specify this with the `-l` parameter.

Next, you can install the Zowe runtime via different methods.

Step 4 (Method 1): Install the Zowe runtime using shell script

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> [-l <LOG_DIR>]
```

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.

Step 4 (Method 2): Install the Zowe runtime using z/OSMF Workflow

A z/OSMF workflow provides the ability to encapsulate a task as a set of dependent steps. These can be divided across different areas of an organization and can form the basis for the automated auditable processes.

z/OSMF workflows consist of a workflow definition that users then operate and manage as workflow tasks. z/OSMF Workflow tasks can help to guide the activities of system programmers, security administrators, and others who are responsible for managing the configuration of the system. For more information on z/OSMF workflows, see [z/OS 2.2 workflows](#), [z/OS 2.3 workflows](#), and [z/OS 2.4 workflows](#).

Zowe provides a z/OSMF workflow definition that can be used to create a runtime environment from the Zowe convenience build. Register and execute the z/OSMF workflow to create a runtime environment with the z/OSMF web interface.

Perform the following steps to register and execute the Zowe runtime installation 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** field.
 - The path to the workflow definition file is `<extracted_pax_folder>/files/workflows/ZWEWRF04.xml` file.

5. **(Optional)** Enter the path to the customized variable input file that you prepared in advance.
 - The path to the variable input file is located is <extracted_pax_folder>/files/workflows/ZWEWRF04.xml file.
 - 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. 10. Select or enter an Owner Use ID and select **Assign all steps to owner user ID**.
9. Select **Finish**.

The workflow is registered in z/OSMF and ready to execute.

10. Select the workflow that you registered from the workflow list.

11. Execute the steps in order. The following steps are displayed that are ready to execute manually:

- **Define Variables**
 - Define the values for variables for the convenience build runtime installation.
- **Allocate ZFS data set**
 - Execute the step to allocate the zFS data set for the Zowe USS.
- **Zowe make dir**
 - Execute the step create a directory for the Zowe USS file system.
- **Mount ZFS**
 - Execute the step to mount the zFS data set to the created directory
- **Set Mountpoint Owner**
 - Execute the step to sets the user who executes the step as the owner of the mountpoint.
 - **Run install script** Execute the step executes the Zowe convenience build install script.

12. 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**.

For general information about how to execute z/OSMF workflow steps, watch the [z/OSMF Workflows Tutorial](#).

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

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](#) that describes how to [Configuring the z/OS system for Zowe](#) on page 126 and [Configuring Zowe certificates](#) on page 136.

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 150.

Zowe has two started tasks that need to be installed and configured ready to be started. These are the Zowe server, see [Installing and starting the Zowe started task \(ZWESVSTC\)](#) on page 156 and the Zowe cross memory server, see [Installing and configuring the Zowe cross memory server \(ZWEISISTC\)](#) on page 146.

Installing Zowe SMP/E

Contents

- [Introduction](#) on page 109
 - [Zowe description](#) on page 109
 - [Zowe FMIDs](#) on page 109
- [Program materials](#) on page 109
 - [Basic machine-readable material](#) on page 109
 - [Program publications](#) on page 109
 - [Program source materials](#) on page 109
 - [Publications useful during installation](#) on page 109
- [Program support](#) on page 110
 - [Statement of support procedures](#) on page 110
- [Program and service level information](#) on page 110
 - [Program level information](#) on page 110
 - [Service level information](#) on page 110
- [Installation requirements and considerations](#) on page 110
 - [Driving system requirements](#) on page 111
 - [Driving system machine requirements](#) on page 111
 - [Driving system programming requirements](#) on page 111
 - [Target system requirements](#) on page 111
 - [Target system machine requirements](#) on page 111
 - [Target system programming requirements](#) on page 111
 - [DASD storage requirements](#) on page 112
 - [FMIDs deleted](#) on page 114
- [Installation instructions](#) on page 115
 - [SMP/E considerations for installing Zowe](#)
 - [SMP/E options subentry values](#)
 - [Overview of the installation steps](#) on page 115
 - [Download the Zowe SMP/E package](#)
 - [Allocate file system to hold the download package](#) on page 116
 - [Upload the download package to the host](#) on page 117
 - [Extract and expand the compressed SMPMCS and RELFILEs](#) on page 118
 - [GIMUNZIP](#) on page 119
 - [Sample installation jobs](#) on page 120
 - [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 122
 - [Allocate z/OS UNIX Paths](#)
 - [Create DDDEF entries](#) on page 123
 - [Perform SMP/E APPLY](#)
 - [Perform SMP/E ACCEPT](#)
 - [Run REPORT CROSSZONE](#) on page 125
 - [Cleaning up obsolete data sets, paths, and DDDEFs](#) on page 125
 - [Activating Zowe](#) on page 125
 - [File system execution](#) on page 125

- [Zowe customization](#) on page 125

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 109 identifies the basic program materials and documentation for Zowe.
- [Program support](#) on page 110 describes the support available for Zowe.
- [Program and service level information](#) on page 110 lists the APARs (program level) and PTFs (service level) that have been incorporated into Zowe.
- [Installation requirements and considerations](#) on page 110 identifies the resources and considerations that are required for installing and using Zowe.
- [Installation instructions](#) on page 115 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 115 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

The Zowe SMP/E package is a distribution of Zowe version 1.9.0 with an FMID of AZWE001.

Subsequent releases of the Zowe z/OS components are delivered as rollup PTFs on zowe.org. Because of the file size of the PTF, it is packaged as two co-requisite PTFs, which are made available in a single Zip file.

Zowe release	PTF 1	PTF 2
1.10	UO01939	UO01940
1.11	UO01942	UO01943
1.12	UO01945	UO01946
1.13	UO01948	UO01949

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	/
File System(s)	9000 Tracks	/
Web Download	26111 Tracks	These are temporary data sets, which can be removed after the SMP/E install.

Notes:

- 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.
- 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 FMID install of Zowe SMP/E but will be required for subsequent SYSMODS 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
SZWEAUTHPF	Load Modules	ANY	U	PDSE	U	0	15	N/A
SZWESAMPSamples		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
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.SMPMCSSEQ		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 116
3. [Upload the download package to the host](#) on page 117
4. [Extract and expand the compressed SMPMCS and RELFILES](#) on page 118
5. [Sample installation jobs](#) on page 120
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 122
10. [Allocate z/OS UNIX paths](#)
11. [Create DDDEF entries](#) on page 123
12. [Perform SMP/E APPLY](#)
13. [Perform SMP/E ACCEPT](#)
14. [Run REPORT CROSSZONE](#) on page 125
15. [Cleaning up obsolete data sets, paths, and DDDEFs](#) on page 125

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 FMID AZWE001** button to save the file to a folder on your desktop.

You will receive one ZIP package on your desktop. You can extract the following files from the package.

- **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
//**
//***** This job must be updated to reflect your environment.
//** 
//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-FTP D1 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 please
Password: tsopw
230 tsouid is loaded on. Working directory is "tsouid.".
ftp> cd @zfs_path@
250 HFS directory @zfs_path@ is the current working directory
ftp> ascii
200 Representation type is Ascii NonPrint
ftp> put c:/AZWE001.readme.txt
200 Port request OK.
150 Storing data set @zfs_path@/AZWE001.readme.txt

```

```

250 Transfer completed successfully.
ftp: 0344 bytes sent in 0.01 sec. (1366.67 Kbs)
ftp binary
200 Representation type is Image
ftp> put c:\AZWE001.pax.Z
200 Port request OK.
145 Storing data set @zfs_path@/AZWE001.pax.Z
250 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@mvsaddrtsouid@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 119 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" />
<ARCHDEV archid="AZWE001.F3"
newname="@PREFIX@.ZOWE.AZWE001.F3" />
<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
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 APPLY 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 118 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 119. Review the following sections for example error and actions that you can take to resolve the error.

- [ZWE2RCVE](#) on page 121
- [ZWE1SMPE](#) and [ZWE4ZFS](#) on page 121
- [ZWEMkdir](#), [ZWE1SMPE](#), [ZWE2RCVE](#), [ZWE3ALOC](#), [ZWE4ZFS](#) and [ZWE5MKD](#)

ZWE2RCVE

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

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

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

This makes it as below:

```
ADD DDDEF(SMPLIB) 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 ALLOC'D ALLOC'D AZWEZFS - SPACE REQUESTED NOT AVAILABLE
IEF272I ZWE3ALOC ALLOC'D ALLOC'D - 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)

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.

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 ZWE7APPLY 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 ZWE7APPLY 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 or 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 HIPERs 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 or 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 112 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 [Configuring Zowe Application Framework](#) on page 179.
- For more information about how to use Zowe, see [Getting started tutorial](#) on page 203.

Installing Zowe SMP/E build 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 h1q.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 h1q.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.

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 [Configuring Zowe Application Framework](#) on page 179.
- For more information about how to use Zowe, see [Getting started tutorial](#) on page 203.

API Mediation Layer as a standalone component

As a Zowe user, follow the procedure in this article to start the API Mediation Layer independently of other Zowe components. By default, the Gateway, Zowe System Services, and Virtual Desktop start when Zowe runs. To limit consumed resources when the Virtual Desktop or Zowe System Services are not required, it is possible to specify which components start in the context of Zowe. No change is required during the installation process to support this setup.

Once Zowe is installed, use the following procedure to limit which components start.

Follow these steps:

1. Open the file <Zowe instance directory>/instance.env.
2. Find the property ZWE_LAUNCH_COMPONENTS and set discovery,gateway,api-catalog
3. Restart Zowe&trade.

To learn more about the related section of the environment file, see [Component groups](#) on page 151. We recommend you open this page in a new tab.

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 the 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.

When you run the ZWESECUR JCL, it does not perform the following initialization steps. Therefore, you must complete these steps manually for a z/OS environment.

- [Grant users permission to access z/OSMF](#)
- [Configure an ICSF cryptographic services environment](#) on page 128
- [Configure multi-user address space \(for TSS only\)](#) on page 132

The ZWESECUR JCL performs the following initialization steps so you do not need to perform them manually if you have successfully run the JCL. These steps are included for reference if you prefer to manually configure the z/OS environment or want to learn more about user IDs, groups, and associated security permissions that are required to operate Zowe.

- [User IDs and groups for the Zowe started tasks](#) on page 133
- [Configure ZWESVSTC to run under ZWESVUSR user ID](#)
- [Configure the cross memory server for SAF](#) on page 134

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. This step is not included in ZWESECUR because it must be done for every TSO user ID who wants to access Zowe's z/OS services. The list of those user IDs is not known by ZWESECUR and will typically be the operators, administrators, developers, or anyone else in the z/OS environment who is logging in to Zowe.

Note: You can skip this section if you use Zowe without z/OSMF. Zowe can operate without z/OSMF but services that use z/OSMF REST APIs will not be available, specifically the USS, MVS, and JES Explorers and the Zowe Command Line Interface files, jobs, workflows, tso, and console groups.

For every TSO user ID that is going to log on to Zowe and use services that require z/OSMF,

- 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(IUUUSER) 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(IUUUSER)
TSS ADD(userid) GROUP(IUUSRGP)
```

Configure an ICSF cryptographic services environment

The zssServer uses cookies that require random number generation for security. To learn more about the zssServer, see the [Zowe architecture](#). Integrated Cryptographic Service Facility (ICSF) is a secure way to generate random numbers.

If you have not configured your z/OS environment for ICSF, see [Cryptographic Services ICSF: System Programmer's Guide](#) for more information. To see whether ICSF has been started, check whether the started task ICSF or CSF is active.

The Zowe z/OS environment configuration JCL member ZWESECUR does not perform any steps related to ICSF that is required for zssServer that the Zowe desktop uses. Therefore, if you want to use Zowe desktop, you must perform the steps that are described in this section manually.

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 152.

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.

Configure multi-user address space (for TSS only)

The ZWESVSTC started task is multi-user address space, and therefore a TSS FACILITY needs to be defined and assigned to the started task. Then, all acids signing on to the started task will need to be authorized to the FACILITY.

The following example shows how to create a new TSS FACILITY.

Example:

In the TSSPARMS, add the following lines to create the new FACILITY:

```
FACILITY(USER11=NAME=ZOWE )
FACILITY( ZOWE=MODE=FAIL )
FACILITY( ZOWE=RES )
```

For more information about how to administer Facility Matrix Table, see [How to Perform Facility Matrix Table Administration](#).

To assign the FACILITY to the started task, issue the following command:

```
TSS ADD( ZWESVUSR ) MASTFAC( ZOWE )
```

To authorize a user to sign on to the FACILITY, issues the following command:

```
TSS ADD(user_acid) FAC( ZOWE )
```

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 user ID must have a valid OMVS segment.

Zowe requires a user ID ZWESIUSR to execute the cross memory server started task ZWESISTC. This user ID 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.

If you have run ZWESECUR, you do not need to perform the steps described in this section, because the TSO commands to create the user IDs and groups are executed during the JCL sections of ZWESECUR.

```
/* group for started tasks */  
...  
/* userid for ZOWE main server */  
...  
/* userid for XMEM cross memory server */  
...
```

If you have not run ZWESECUR and are manually creating the user ID and groups in your z/OS environment, the commands are described below for reference.

- 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 have run ZWESECUR, you do not need to perform the steps described in this section, because they are executed during the JCL section of ZWESECUR.

```
/* started task for ZOWE main server */  
...
```

If you have not run ZWESECUR and are configuring your z/OS environment manually, the following steps describe how to configure the started task ZWESVSTC to run under the correct user ID and group.

- 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)
```

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-privileged clients.

If you have run ZWESECUR you do not need to perform the steps described in this section, as they are executed during the JCL section of ZWESECUR.

```
/* permit Zowe main server to use XMEM cross memory server */  
...
```

If you have not run ZWESECUR and are configuring your z/OS environment manually, the following steps describe how to configure the cross memory server for SAF.

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 user ID 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(ZWESVUSR) 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 main Zowe server to use identity mapping

This security configuration is necessary for API ML to be able to map client certificate to a z/OS identity. A user running API Gateway must have read access to the RACF general resource IRR.RUSERMAP in the FACILITY class. To set up this security configuration, submit the ZWESECUR JCL member. For users upgrading from version 1.18 and lower use the following configuration steps.

Using RACF

If you use RAFC, verify and update permission in the FACILITY class.

Follow these steps:

1. Verify user ZWESVUSR has read access.

```
RLIST FACILITY IRR.RUSERMAP AUTHUSER
```

2. Add user ZWESVUSR permission to read.

```
PERMIT IRR.RUSERMAP CLASS(FACILITY) ACCESS(READ) ID(ZWESVUSR)
```

3. Activate changes.

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

Using ACF2

If you use ACF2, verify and update permission in the FACILITY class.

Follow these steps:

1. Verify user ZWESVUSR has read access.

```
SET RESOURCE(FAC)
LIST LIKE(IRR-)
```

2. Add user ZWESVUSR permission to read.

```
RECKEY IRR.RUSERMAP ADD(SERVICE(READ) ROLE(&STCGRP.) ALLOW)
```

Using TSS

If you use TSS, verify and update permission in FACILITY class.

Follow these steps:

1. Verify user ZWESVUSR has read access.

```
TSS WHOHAS IBMFAC(IRR.RUSERMAP)
```

2. Add user ZWESVUSR permission to read.

```
TSS PER(ZWESVUSR) IBMFAC(IRR.RUSERMAP) ACCESS(READ)
```

Configuring Zowe certificates

Zowe uses a certificate to encrypt data for communication across secure sockets. An instance of Zowe references a USS directory referred to as a KEYSTORE_DIRECTORY which contains information about where the certificate is located.

Learn more about the key concepts of Zowe certificates in the following sections.

Northbound Certificate

The Zowe certificate is used by the API Mediation Layer on its northbound edge when identifying itself and encrypting <https://> traffic to web browsers or REST client applications. If the Zowe Command Line Interface (CLI) has been configured to use the Zowe API Mediation Layer then the CLI is a client of the Zowe certificate. For more information, see [Using the Zowe Command Line Interface, Integrating with the API Mediation Layer](#).

Southbound Certificate

As well as being a server, Zowe itself is a client to services on the southbound edge of its API Mediation Layer that it communicates to over secure sockets. These southbound services use certificates to encrypt their data, and Zowe uses a trust store to store its relationship to these certificates. The southbound services that are started by Zowe itself and run as address spaces under its ZWESVSTC started task (such as the API discovery service, the explorer JES REST API server) re-use the same Zowe certificate used by the API Mediation Layer on its northbound client edge.

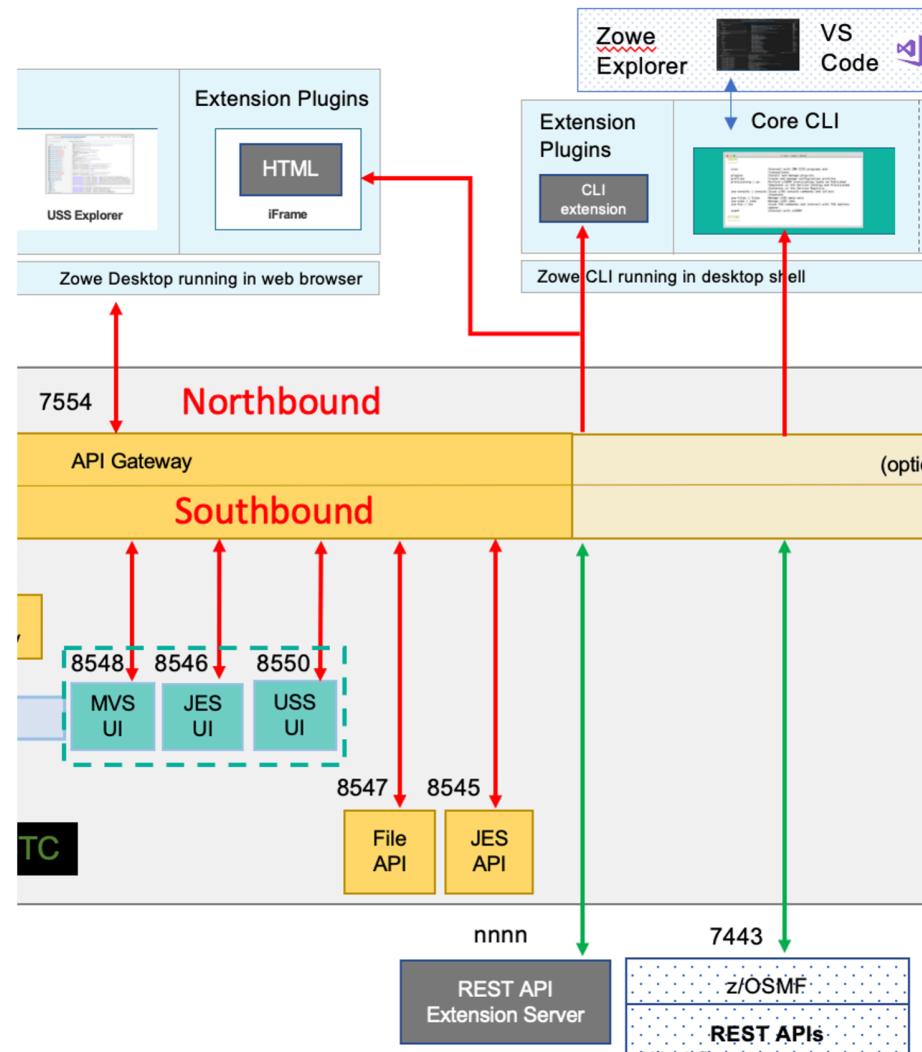
Trust store

As well as Zowe using its certificates intra-address space, to encrypt messages between its servers, Zowe uses external services on z/OS (such as z/OSMF or Zowe conformant extensions that have registered themselves with the API Mediation Layer). These services will present their own certificate to the API Mediation Layer, in which case the trust store is used to capture the relationship between Zowe's southbound edge and these external certificates.

If you wish to disable the trust store validation of southbound certificates, you can set the value `VERIFY_CERTIFICATES=true` to `false` in the `zowe-setup-certificates.env` file in the `KEYSTORE_DIRECTORY`. A scenario when this is recommended is if certificate being presented to the API Mediation Layer is self-signed (that is, from an unknown certificate authority). For example, the z/OSMF certificate may be self-signed in which case the Zowe API Mediation Layer will not recognize the signing authority.

Certificates in the Zowe architecture

The [Zowe architecture](#) on page 13 shows the Zowe API Mediation Layer positioned on the client-server boundary between applications such as web browsers or the Zowe CLI accessing z/OS services. The following diagram is a section of the architecture annotated to describe the role of certificates and trust stores.



The lines shown in bold red are communication over a TCP/IP connection that is encrypted with the Zowe certificate.

- On the northbound edge of the API gateway, the certificate is used between client applications such as web browsers, Zowe CLI, or any other application wishing to access Zowe's REST APIs.
- On the southbound edge of the API Gateway, there are a number of Zowe micro services providing HTML GUIs for the Zowe desktop or REST APIs for the API Catalog. These also use the Zowe certificate for data encryption.

The lines in bold green are external certificates for servers that are not managed by Zowe, such as z/OSMF itself or any Zowe conformant REST API or App Framework servers that are registered with the API Mediation Layer. For the API Mediation Layer to be able to accept these certificates, they either need to be signed by a recognized certificate authority, or else the API Mediation Layer needs to be configured to accept unverified certificates. Even if the API Mediation Layer is configured to accept certificates signed by unverified CAs on its southbound edge, client applications on the northbound edge of the API gateway will be presented with the Zowe certificate.

Keystore versus key ring

Zowe supports certificates that are stored in a USS directory **Java KeyStore** format.

Beginning with release 1.15, Zowe is including the ability to work with certificates held in a **z/OS Keyring**. Support for Keyring certificates is currently incomplete and being provided as a beta technical preview for early preview by customers. If you have any feedback using keyrings please create an issue in the [zowe-install-packaging repo](#). It is expected that in a future release keyring support will be made available as a fully supported feature.

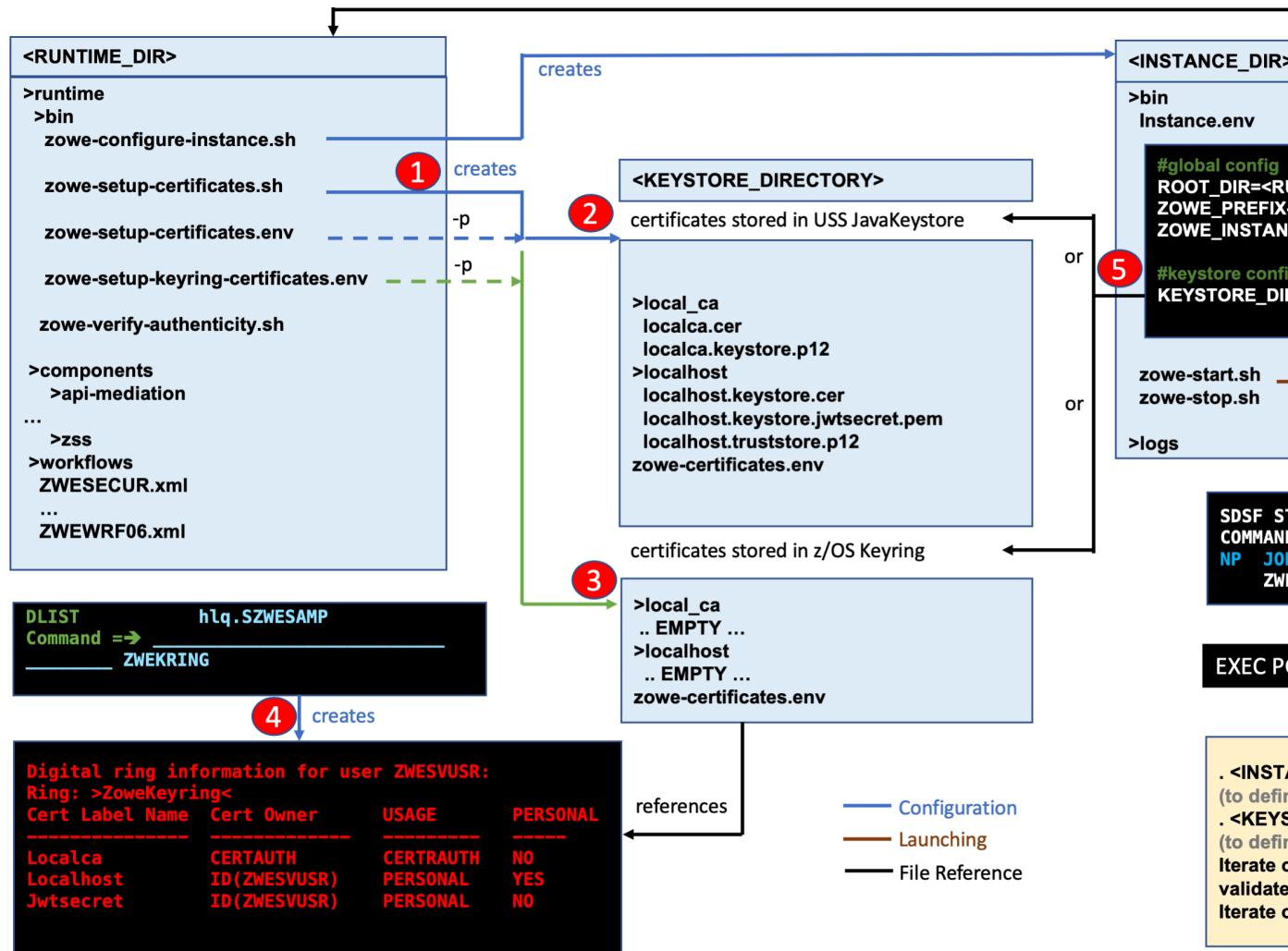
Keystore directory creation

The KEYSTORE_DIRECTORY is created by running the script <RUNTIME_DIR>/bin/zowe-setup-certificates.sh. This script has a number of input parameters that are specified in a configuration file whose location is passed as an argument to the -p parameter.

The configuration file <RUNTIME_DIR>/bin/zowe-setup-certificates.env is provided for setting up a Keystore directory that contains the Zowe certificate in JavaKeystore format. The configuration file <RUNTIME_DIR>/bin/zowe-setup-certificates-keyring.env is provided for setting up a Keystore directory that references the Zowe certificate held in a z/OS keyring.

The .env configuration file should be customized based on security rules and practices for the z/OS environment. Once the script has been successfully executed and the KEYSTORE_DIRECTORY is created successfully, it is referenced by a Zowe launch instance.env file. A KEYSTORE_DIRECTORY can be used by more than one instance of Zowe. See [Keystore configuration](#) on page 152 for more information.

The Zowe launch diagram shows the relationship between a Zowe instance directory, a Zowe runtime directory, the Zowe keystore directory, and (if used to store the Zowe certificate) the z/OS keyring.



You create a **KEYSTORE_DIRECTORY** in USS by using the script `zowe-setup-certificates.sh` (1) with a `-p` argument that specifies a `.env` configuration file.

- If the `-p` argument file `zowe-setup-certificates.env` (2) is used, the **KEYSTORE_DIRECTORY** will contain the certificate, the certificate authority, the trust store, and the JWT Secret.
- If the `-p` argument file `zowe-setup-keyring-certificates.env` (3) is used, the **KEYSTORE_DIRECTORY** contains no certificates and is a pass-through to configure a Zowe instance to use a z/OS keyring.

The JCL member `ZWEKRING` (4) is used to create a z/OS Keyring to hold the Zowe certificate and its signing certificate authority.

At launch time, a Zowe instance is started using the script `<INSTANCE_DIR>/bin/zowe-start.sh` which takes configuration arguments from `<INSTANCE_DIR>/instance.env`. The argument (5) `KEYSTORE_DIRECTORY=<KEYSTORE_DIRECTORY>` specifies the path to the keystore directory that Zowe will use.

For more information on the Zowe launch topology, see [Topology of the Zowe z/OS launch process](#).

Configuring Zowe certificates in UNIX files

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

Zowe is launched, the instance directory configuration file `instance.env` specifies the location of the keystore directory. See [Creating and configuring the Zowe 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 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](#) on page 343. 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.

We recommend that you start with the local API Mediation Layer CA for an initial evaluation.

You can use the `<RUNTIME_DIR>/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 `<RUNTIME_DIR>/bin/zowe-setup-certificates.env`.

Generate certificate with the default values

The script reads the default variable values that are provided in the `<RUNTIME_DIR>/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 runtime directory:

```
<RUNTIME_DIR>/bin/zowe-setup-certificates.sh
```

generates the keystore in `/global/zowe/keystore`. On many z/OS installations access to this location will be restricted to privileged users so this step should be done by a system programmer with site knowledge for where the certificate should be stored in a way that the public key can be read but private key access is controlled.

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 [-l <log_directory>]
```

where <your_directory> specifies the location of your customized environment file and <log-directory> is an optional parameter that overrides the default log output directory of /global/zowe/logs, if it is writable, or ~/zowe/logs.

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, see [Keystore configuration](#) on page 152.

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. See $LOG_FILE for more details
ERROR: z/OSMF is not trusted by the API Mediation Layer. Make sure
ZOWE_ZOSMF_HOST and ZOWE_ZOSMF_PORT variables define the desired z/OSMF
instance.
ZOWE_ZOSMF_HOST=${ZOWE_ZOSMF_HOST} ZOWE_ZOSMF_PORT=${ZOWE_ZOSMF_PORT}
You can also specify z/OSMF certificate explicitly in the ZOSMF_CERTIFICATE
environmental variable in the zowe-setup-certificates.env file.
```

This error must be resolved before you can proceed with the next installation step.

Note:

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.

Using web tokens for SSO on ZLUX and ZSS

Users must create a PKCS#11 token before continuing. This can be done through the USS utility, "gskkyman".

Creating a PKCS#11 Token

Ensure that the SO.TOKEN_NAME profile exists in CRYPTOZ, and that the user who will be creating tokens has either UPDATE or CONTROL access.

1. Define profile: "RDEFINE CRYPTOZ SO.TOKEN_NAME"
2. Add user with UPDATE access: "PERMIT SO.** ACCESS(UPDATE) CLASS(CRYPTOZ) ID(USERID)"
3. Ensure profile was created: "RLIST CRYPTOZ **"
4. Activate class with new profile:
 - "SETROPTS RACLIST(CRYPTOZ)"
 - "SETROPTS CLASSACT(CRYPTOZ)"

A user should now be able to use "gskkyman" to create a token.

Accessing token

Ensure USER.TOKEN_NAME profile exists in CRYPTOZ:

1. Define profile: "RDEFINE CRYPTOZ USER.TOKEN_NAME"
2. Add user with READ access: "PERMIT USER.TOKEN_NAME ACCESS(UPDATE) CLASS(CRYPTOZ) ID(USERID)"
3. Ensure profile was created: "RLIST CRYPTOZ **"

4. Activate class with new profile:

- "SETROPTS RACLIST(CRYPTOZ)"
- "SETROPTS CLASSACT(CRYPTOZ)"

Configure zowe-setup-certificates.env using the following parameters. Both are required to enable SSO.

- PKCS#11 token name for SSO. Must already exist.
PKCS11_TOKEN_NAME=<newly created token name>
- PKCS#11 token label for SSO. Must not already exist.
PKCS11_TOKEN_LABEL=<unique label>

Enabling SSO

1. Run zowe-setup-certificates.sh.

- If you are upgrading from an older version of Zowe that has the apiml configured: "rerun zowe-setup-certificates.sh"
- If upgrading, point the zowe instance to the newly generated keystore, or overwrite the previous one.

2. In the ZSS server configuration, enable SSO and input your token name/label:

```
"agent": {
  //host is for zlux to know, not zss
  "host": "localhost",
  "http": {
    "ipAddresses": [ "0.0.0.0" ],
    "port": 0000
  },
  "jwt": {
    "enabled": true,
    "fallback": false,
    "token": "TOKEN.NAME",
    "label": "KEY_NAME"
  },
}
```

Hints and tips

Learn about some hints and tips that you might find useful when you create certificates.

You create the certificates by running the script `zowe-setup-certificates.sh`. You do not need to rerun the script after the first time you install Zowe, unless instructed otherwise by SMP/E HOLDDATA or the release notes for that release.

The creation of the certificates is controlled by the `zowe-setup-certificates.env` file, and you should have placed a copy of that file in your instance directory `INSTANCE_DIR`.

1. Keystore

In your copy of the `zowe-setup-certificates.env` file, specify the location where you want the `zowe-setup-certificates.sh` script to place the keys it generates.

```
KEYSTORE_DIRECTORY=/my/zowe/instance/keystore
```

By default, a keystore can be shared by all instances, which is also recommended. The default location is `/global/zowe/keystore`. You can use a different shared location if you prefer. The Zowe instance uses the keystore that you specify in `instance.env` in your instance directory `INSTANCE_DIR`. This can be the shared location or you can create another keystore in a different location for that instance and use that one instead. A single, shared keystore is recommended.

2. Hostname and IP address

You specify the hostname and IP address with the following keywords in the `zowe-setup-certificates.env` file.

```
HOSTNAME=
IPADDRESS=
```

The certificates require the value of `HOSTNAME` to be an alphabetic hostname. Numeric hostnames such as an IP address are not allowed.

The `zowe-setup-certificates.sh` script attempts to discover the IP address and hostname of your system if you leave these unconfigured in `zowe-setup-certificates.env`.

On systems with their own internal IP domain, the hostname might not resolve to the external IP address. This happens on ZD&T ADCD-derived systems, where the hostname is usually `S0W1.DAL-EBIS.IHOST.COM` which resolves to `10.1.1.2`. When the script cannot determine the hostname or the external IP address, it will ask you to enter the IP address manually during the dialog. If you have not specified a value for `HOSTNAME` in `zowe-setup-certificates.env`, then the script will use the given IP address as the hostname. This will fail because certificates cannot have a numeric hostname.

Therefore, you must specify an alphabetic hostname such as the following one on ZD&T systems before you run the script `zowe-setup-certificates.sh`.

```
HOSTNAME=S0W1.DAL-EBIS.IHOST.COM
```

The values of `HOSTNAME` and `IPADDRESS` that the script discovered are appended to the `zowe-setup-certificates.env` file unless they were already set in that file or as shell environment variables before you ran the script.

Configuring Zowe certificates in a key ring

Zowe is able to work with certificates held in a **z/OS Keyring**. For background on Zowe certificates, see [Configuring Zowe certificates](#) on page 136. To configure Zowe certificates in a key ring, run the `ZWEKRING` JCL that contains the security commands to create the key ring and manage the certificates that Zowe will use. The `ZWEKRING` JCL is provided as part of the PDS sample library `SZWEESAMP` that is delivered with Zowe.

Before you submit the JCL, you must [Customizing the ZWEKRING JCL](#) on page 145 and review it with a system programmer who is familiar with z/OS certificates and key rings. The JCL member contains commands for three z/OS security managers: RACF, TopSecret, and ACF/2.

The `ZWEKRING` JCL contains commands for the following scenarios:

- Creation of a local CA which is used to sign a locally generated certificate, both of which are placed into the key ring.
- (**Beta**) Importing an existing certificate already held in z/OS to the key ring for use by Zowe.
- (**Beta**) Creation of a locally generated certificate and signing it with an existing certificate authority, and placing the certificate into the key ring.

Note: The scenarios marked **Beta** are provided for technical preview. If you have any feedback on using key rings, create an issue in the Zowe community repo at <https://github.com/zowe/community>.

After you run the `ZWEKRING` JCL, a key ring that contains the Zowe certificate is created. In order for a Zowe instance to work with the keystore certificate, you also need to create a USS keystore directory. This USS keystore directory does not contain any certificates, but is required for the Zowe `instance.env` file to configure the Zowe shell correctly so that the keystore certificate can be located by the Zowe runtime.

To create the USS keystore directory after successfully running `ZWEKRING` JCL member, run the script `<RUNTIME_DIR>/bin/zowe-setup-certificates.sh`. This script has an input parameter `-p` which specifies the location of a configuration file controlling how and where the directory and its contents are created. Copy the file `<RUNTIME_DIR>/bin/zowe-setup-certificates.env` to a writeable location and review

and edit its contents to match property values used in ZWEKRING JCL member. Then, run the script by using the following command:

```
zowe-setup-certificates.sh -p <path to zowe-setup-keyring-certificates.env>
```

Customizing the ZWEKRING JCL

To customize the ZWEKRING JCL, edit the JCL variables at the beginning of the JCL and carefully review and edit all the security commands that are valid for your security manager. Review the information in this section when you customize the JCL.

PRODUCT variable

The PRODUCT variable specifies the z/OS security manager. The default value is RACF. Change the value to ACF2 or TSS if you are using Access Control Facility CA-ACF2 or CA Top Secret for z/OS as your z/OS security manager.

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

HOSTNAME and IPADDRESS

The Zowe certificate is used on the northbound edge of the API Mediation Layer to encrypt data between web browser and other client applications such as the Zowe command line interface. These client applications will validate that the network TCP/IP address that they have accessed the encrypted data from matches the network address in the certificate. If the address does not match, the browser will not continue as it will consider the site as unsecure.

To ensure that the browser is able to establish a secure connection, set the HOSTNAME and IPADDRESS in the ZWEKRING JCL member to match the hostname and TCP/IP address of the Zowe API Mediation Layer.

```
/*      * Hostname of the system where Zowe is to run
/*
/*      * SET HOSTNAME=' '
/*
/*      * IP address of the system where Zowe is to run
/*
/*      * SET IPADDRES=' '
/*
/*      * Keyring for the Zowe userid
```

ZOWERING and LABEL labels

The ZOWERING label is used for the name of the key ring created. The default value is ZoweKeyring. The LABEL label specifies the certificate name and defaults to localhost.

```
//          SET ZOWERING='ZoweKeyring'
//          * Zowe's certificate label
//          SET      LABEL='localhost'
```

- The value of the ZOWERING label should match the value of the ZOWE_KEYRING variable in the zowe-setup-keyring-certificates.env file.
- The value of the LABEL label should match the value of the KEYSTORE_ALIAS variable in the zowe-setup-keyring-certificates.env file.

ROOTZFCA label

The ROOTZFCA label connects the root CA of the z/OSMF certificate with the Zowe key ring.

When to set this label?

The value of the parameter VERIFY_CERTIFICATES in the zowe-certificates.env file in the KEYSTORE_DIRECTORY controls whether Zowe's servers validate the authenticity of any southbound certificates at runtime. If the value is true, then the certificate must be signed by a recognized certificate authority (CA), and if the value is false then self-signed certificates are allowed. This section of the keystore configuration is only required if you are using VERIFY_CERTIFICATES=true.

When you set VERIFY_CERTIFICATES=true, then Zowe will validate the authenticity of the z/OSMF certificate, so the root CA of the z/OSMF certificate must be connected with the Zowe key ring. You can connect them by setting the label ROOTZFCAs.

```
// *      * Name/Label of the root CA of the z/OSMF certificate
//      SET ROOTZFCAs=
```

If you are unsure of the root CA you can find it by listing the chain of the z/OSMF certificate using the following commands:

- RACF

```
RACDCERT ID(IZUSVR) LISTCHAIN(LABEL('DefaultzOSMFCert.IZUDFLT'))
```

- Top Secret

```
TSS LIST(IZUSVR) LABLCERT('DefaultzOSMFCert.IZUDFLT') CHAIN
```

- ACF2

```
SET PROFILE(USER) DIVISION(CERTDATA)
CHKCERT IZUSVR LABEL(DefaultzOSMFCert.IZUDFLT) CHAIN
```

Results

When the ZWEKRING JCL runs successfully, it will create a key ring named `ZoweKeyring` owned by `ZWESVUSR` containing the following:

- The Zowe certificate (called `localhost`)
- The local CA (called `ZoweCert`)
- The certificate used to encrypt the JSON Web Token (JWT) required for single sign-on (called `jwtsecret`)

When the `zowe-setup-certificates.sh` script executes successfully, it will generate the USS `KEYSTORE_DIRECTORY` that contains the file `zowe-certificates.env`. This file is used in the Zowe instance configuration step. See [Keystore configuration](#) on page 152.

Cleanup

The JCL member `ZWENOKYR` provided in the PDS sample library `SZWESAMP` contains the inverse commands contained in `ZWEKKRING`. This allows an environment to be cleaned up and have one or more certificates, key rings, and certificate authorities created by `ZWEKRING` removed from the z/OS environment. This is useful if you are creating a DevOps pipeline to install and configure and environment for Zowe using `ZWEKRING` and want to clean that environment before rerunning the pipeline.

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. If you wish to start Zowe without the desktop (for example bring up just the API Mediation Layer), you do not need to install and configure a cross memory server and can skip this step. The cross memory server is needed to be able to log on to the Zowe desktop and operate its apps such as the File Editor.

To install and configure the cross memory server, you must define APF-authorized load libraries, program properties table (PPT) entries, and a parmlib. This requires familiarity with z/OS.

- [PDS sample library and PDSE load library](#) on page 147
- [Load module](#) on page 147
 - [APF authorize](#) on page 147
 - [Key 4 non-swappable](#) on page 148
- [PARMLIB](#) on page 148

- [PROCLIB](#) on page 148
- [SAF configuration](#) on page 149
- [Summary of cross memory server installation](#) on page 149
- [Starting and stopping the cross memory server on z/OS](#) on page 149
- [Zowe auxiliary service](#) on page 149
 - [When to configure the auxiliary service](#) on page 150
 - [Installing the auxiliary service](#) on page 150

PDS sample library and PDSE load library

The cross memory server runtime artifacts, the JCL for the started tasks, the parmlib, and members containing sample configuration commands are found in the SZWESAMP PDS sample library.

The load modules for the cross memory server and an auxiliary server it uses are found in the SZWEAAUTH PDSE.

The location of SZWESAMP and SZWEAAUTH for a convenience build depends on the value of the `zowe-install.sh -h` argument. For more information, see [Step 3: Choose a dataset HLQ for the SAMPLIB and LOADLIB](#) on page 105.

For an SMP/E installation, SZWESAMP and SZWEAAUTH are the SMP/E target libraries whose location depends on the value of the `#th1q` placeholder in the sample member AZWE001.F1 (ZWE3ALOC).

The cross memory server is a long running server process that, by default, runs under the started task name ZWESISTC with the user ID ZWESIUSR and group of ZWEADMIN.

The ZWESISTC started task 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.

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 load library SZWEAAUTH and PARMLIB data sets. The cross memory server loads some functions to LPA for its PC-cp services.

To install the cross memory server, enable the PROCLIB, PARMLIB, and load module. This topic describes the steps to do this manually.

If you want to install and configure the cross memory server by using scripts, see [Scripted installation and configuration of Zowe z/OS components](#).

Load module

The cross memory server load module ZWESIS00 is installed by Zowe into a PDSE SZWEAAUTH. For the cross memory server to be started, the load module needs to be APF-authorized and the program needs to run in key(4) as non-swappable.

APF authorize

APF authorize the PDSE SZWESAUTH. This allows the SMP/E APPLY and RESTORE jobs used for applying maintenance to be operating on the runtime PDSE itself when PTF maintenance is applied.

Do not add the SZWEAAUTH data set to the system LNKLST or LPALST concatenations.

To check whether a load library is APF-authorized, you can issue the following command:

```
D PROG,APF,DSNAME=hlq.SZWEAAUTH
```

where the value of DSNAME is the name of the SZWEAAUTH data set as created during Zowe installation that contains the ZWESIS01 load module.

Issue one of the following operator commands to dynamically add the load library to the APF list (until next IPL), where the value of DSNAME is the name of the SZWEAAUTH data set, as created during Zowe installation.

- If the load library is not SMS-managed, issue the following operator command, where `volser` is the name of the volume that holds the data set:

```
SETPROG APF,ADD,DSNAME=hlq.SZWEAUTH,VOLUME=volser
```

- If the load library is SMS-managed, issue the following operator command:

```
SETPROG APF,ADD,DSNAME=hlq.SZWEAUTH,SMS
```

Add one of the following lines to your active PROGxx PARMLIB member, for example

SYS1.PARMLIB (PROG00), to ensure that the APF authorization is added automatically after next IPL. The value of DSNAME is the name of the SZWEAUTH data set, as created during Zowe installation:

- If the load library is not SMS-managed, add the following line, where `volser` is the name of the volume that holds the data set:

```
APF ADD DSNAME=hlq.SZWEAUTH VOLUME=volser
```

- If the load library is SMS-managed, add the following line:

```
APF ADD DSNAME=hlq.SZWEAUTH SMS
```

The PDS member SZWESAMP (ZWESIMPRG) contains the SETPROG statement and PROGxx update for reference.

Key 4 non-swappable

The cross memory server load module ZWESIS01 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 SCHEDEXxx member of the system PARMLIB.

```
PPT PGMNAME(ZWESIS01) KEY(4) NOSWAP
```

The PDS member SZWESAMP (ZWESISCH) contains the PPT lines for reference.

Then, issue the following command to make the SCHEDEXxx changes effective:

```
/SET SCH=xx
```

PARMLIB

The ZWESISTC started task must find a valid ZWESIPxx PARMLIB member in order to be launched successfully. The SZWESAMP PDS created at installation time contains the member ZWESIP00 with default configuration values. You can copy this member to another data set, for example your system PARMLIB data set, or else leave it in SZWESAMP.

If you choose to leave ZWESIPxx in the installation PDS SZWESAMP used at installation time, this has advantages for SMP/E maintenance because the APPLY and RESTORE jobs will be working directly against the runtime library.

Wherever you place the ZWESIP00 member, ensure that the data set is listed in the PARMLIB DD statement of the started task ZWESISTC.

PROCLIB

For the cross memory server to be started, you must move the JCL PROCLIB ZWESISTC member from the installation PDS SAMPLIB SZWESAMP into a PDS that is on the JES concatenation path.

You need to update the ZWESISTC member in the JES concatenation path with the location of the load library that contains the load module ZWESI00 by editing the STEPLIB DD statement of ZWESISTC. Edit the PARMLIB DD statement to point to the location of the PDS that contains the ZWESIP00 member.

For example, the sample JCL below shows ZWESVSTC where the APF-authorized PDSE containing ZWESI00 is ZWESVUSR.SZWEAUTH and the PDS PARMLIB containing ZWESIP00 is ZWESVUSR.SZWESAMP.

```
//ZWESIS01 EXEC PGM=ZWESIS01,REGION=&RGN,
//           PARM='NAME=&NAME, MEM=&MEM'
//STEPLIB   DD   DSNAME=ZWESVUSR.SZWEAUTH,DISP=SHR
//PARMLIB   DD   DSNAME=ZWESVUSR.SZWESAMP,DISP=SHR
//SYSPRINT  DD   SYSOUT=*
```

SAF configuration

You must configure the z/OS system in order to correctly run the cross memory server. The steps to perform this are included in the JCL member ZWESECUR that is used to configure a z/OS environment for Zowe, and documented in the section [Configure the cross memory server for SAF](#) on page 134.

Summary of cross memory server installation

You can start the cross memory server using the command /S ZWESISTC once the following steps have been completed.

- JCL member ZWESVSTC is copied from SZWESAMP installation PDS to a PDS on the JES concatenation path.
- The PDSE Load Library SZWEAUTHis APF-authorized, or Load module ZWESI00 is copied to an existing APF Auth LoadLib.
- The JCL member ZWESVSTC DD statements are updated to point to the location of ZWESI00 and ZWESIP00.
- The load module ZWESI00 must run in key 4 and be non-swappable by adding a PPT entry to the SCHEDxx member of the system PARMLIB PPT PGMNAME(ZWESI00) KEY(4) NOSWAP.

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 ZWESASTC 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 started task for the main Zowe servers is independent of the ZWESISTC cross memory server, which 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 its 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 not 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 439.

Zowe auxiliary service

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.

When to configure the auxiliary service

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. A vendor product documentation will specify whether it needs the Zowe auxiliary service to be configured so ensure that it is needed before attempting the configuration steps.

If you are just using core Zowe functionality, you **do not** need to configure the auxiliary service. Even with the Zowe auxiliary service configured, there is no situation under which you should manually start the ZWESASTC started task.

Installing the auxiliary service

To install the auxiliary service to allow it to run, you take similar steps to install and configure the cross memory server as described above, but with a different JCL PROBLIC member and a different load module. There is no PARMLIB for the auxiliary service.

- JCL member ZWESASTC is copied from SZWESAMP installation PDS to a PDS on the JES concatenation path.
- The PDSE load library SZWEAUX is APF-authorized, or load module ZWESAU is copied to an existing APF Auth LoadLib.
- The load module ZWESAU must run in key 4 and be non-swappable by adding a PPT entry to the SCHEDxx member of the system PARMLIB PPT PGMNAME(ZWESAU) KEY(4) NOSWAP.

Important!

The cross memory ZWESISTC task starts and stops the ZWESASTC task as needed. **Do not start the ZWESASTC task manually.**

Creating and configuring the Zowe instance directory

The Zowe instance directory or <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.

Note: The creation of an instance directory will set default values for users who want to run all Zowe z/OS components. If you are using Docker, you must make a small configuration change to disable the components on z/OS that will instead run in Docker.

The instance directory <INSTANCE_DIRECTORY>/bin contains a number of key scripts

- zowe-start.sh is used to start the Zowe runtime by launching the ZWESVSTC started task.
- zowe-stop.sh is used to stop the Zowe runtime by terminating the ZWESVSTC started task.
- zowe-support.sh can be used to capture diagnostics around the Zowe runtime for troubleshooting and off-line problem determination, see [Capturing diagnostics to assist problem determination](#) on page 401.

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 136. 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 126.

Creating an instance directory

To create an instance directory, use the zowe-configure-instance.sh script.

Navigate to the Zowe runtime directory <RUNTIME_DIR> and execute the following commands:

```
<RUNTIME_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 `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.

In order to allow the `ZWESVSTC` started task to have permission to access the contents of the `<INSTANCE_DIR>` the `zowe-configure-instance.sh` script sets the group ownership of the top level directory and its child to be `ZWEADMIN`. If a different group is used for the `ZWESVSTC` started task you can specify this with the optional `-g` argument, for example.

```
<RUNTIME_DIR>/bin/zowe-configure-instance.sh -c <PATH_TO_INSTANCE_DIR> -g
<GROUP>
```

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 that 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. In addition to the mediation layer, the z/OS Explorer services are included here as well.
- `DESKTOP` will start the Zowe desktop that is the browser GUI for hosting Zowe applications such as the 3270 Terminal emulator or the File Explorer. It will also start ZSS. The Zowe desktop is a node application and uses the version specified by the `NODE_HOME` value.
- `ZSS` will start the ZSS server without including the Desktop and Application Framework server. This can be used with Docker so that you do not run servers on z/OS that will already be running within Docker. This may also be useful if you want to utilize ZSS core features or plug-ins without needing the Desktop. ZSS is a pre-requisite for the Zowe desktop, so when the `DESKTOP` group is specified then the zss server will be implicitly started. For more information on the zssServer and the technology stack of the Zowe servers see [Zowe architecture](#) on page 13.
- Vendor products may extend Zowe with their own component group that they want to be lifecycled by the Zowe `ZWESVSTC` started task and run as a Zowe sub address space. To do this, specify the fully qualified directory provided by the vendor that contains their Zowe extension scripts. This directory will contain a `start.sh` script (**required**) that is called when the `ZWESVSTC` started task is launched, a `configure.sh` script (**optional**) that performs any configuration steps such as adding iFrame plug-ins to the Zowe desktop, and a `validate.sh` script (**optional**) that can be used to perform any pre-launch validation such as checking system prerequisites. For more information about how a vendor can extend Zowe with a sub address space, see the [Onboarding Overview](#) on page 302 section.

Note: If you are using Docker, it is recommended to remove `GATEWAY` and `DESKTOP` from `LAUNCH_COMPONENT_GROUPS` by setting `LAUNCH_COMPONENT_GROUPS=zss`. This will prevent duplication of servers running both in Docker and on z/OS.

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.
- **SKIP_NODE :** When Zowe starts, it checks whether the NODE_HOME path is a valid node runtime. If not, it will prompt for the location of where node can be located. Specify a value of 1 to bypass this step, or 0 for the check to occur. This may be useful in an automation scenario where the zowe-start.sh script is run unattended and the makeup of the components being launched does not require a node runtime.
- **ROOT_DIR:** The directory where the Zowe runtime is located, also referred to as the <RUNTIME_DIR>. 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 returns 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 136.

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** - App 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 132 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 these microservices use to provide access to their services. You can leave default values for components that are not in use. 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. Used when LAUNCH_COMPONENT_GROUPS includes GATEWAY.
- DISCOVERY_PORT: The port the discovery service will use. Used when LAUNCH_COMPONENT_GROUPS includes GATEWAY.
- GATEWAY_PORT: The port the API gateway service will use. Used when LAUNCH_COMPONENT_GROUPS includes GATEWAY. 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. Used when LAUNCH_COMPONENT_GROUPS includes GATEWAY.
- FILES_API_PORT: The port the files API service will use. Used when LAUNCH_COMPONENT_GROUPS includes GATEWAY.
- JES_EXPLORER_UI_PORT: The port the jes-explorer UI service will use. Used when LAUNCH_COMPONENT_GROUPS includes GATEWAY.
- MVS_EXPLORER_UI_PORT: The port the mvs-explorer UI service will use. Used when LAUNCH_COMPONENT_GROUPS includes GATEWAY.
- USS_EXPLORER_UI_PORT: The port the uss-explorer UI service will use. Used when LAUNCH_COMPONENT_GROUPS includes GATEWAY.
- ZOWE_ZLUX_SERVER_HTTPS_PORT: The port used by the Zowe desktop. Used when LAUNCH_COMPONENT_GROUPS includes 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. Used when LAUNCH_COMPONENT_GROUPS includes DESKTOP or ZSS.

Note: If all of the default port values are acceptable, the ports do not need to be changed. To allocate ports for the Zowe runtime servers, ensure that the ports are not in use.

To determine which ports are not available, follow these steps:

1. Display the list of ports that are in use with the following command:

```
TSO NETSTAT
```

2. Display the 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 to z/OS 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 emulator inside the Zowe desktop web page. This port is the number used by the z/OS telnet service and defaults to 23. The USS command `netstat -b | grep TN3270` can be used to display the telnet port used on a z/OS system.
- **ZOWE_ZLUX_SECURITY_TYPE:** The *3270 Terminal* application needs to know whether the telnet service is using `tls` or `telnet` for security. The default value is blank for `telnet`.

Gateway configuration

The following parameters can be set to customize the configuration of the Gateway:

- **APIML_ALLOW_ENCODED_SLASHES:** When this parameter is set to `true`, the Gateway allows encoded characters to be part of URL requests redirected through the Gateway.
- **APIML_CORS_ENABLED:** When this parameter is set to `true`, CORS are enabled in the API Gateway for Gateway routes `api/v1/gateway/**`.
- **APIML_PREFER_IP_ADDRESS:** Set this parameter to `true` to advertise a service IP address instead of its hostname.
- **APIML_GATEWAY_TIMEOUT_MILLIS:** Specifies the timeout for connection to the services in milliseconds.
- **APIML_SECURITY_X509_ENABLED:** Set this parameter to `true`, to enable the client certificate authentication functionality through ZSS.
- **APIML_SECURITY_ZOSMF_APPLID:** The z/OSMF APPLID used for PassTicket.
- **APIML_SECURITY_AUTH_PROVIDER:** The authentication provider used by the API Gateway. By default, the API Gateway uses z/OSMF as an authentication provider. It is possible to switch to SAF as the authentication provider instead of z/OSMF.
- **APIML_DEBUG_MODE_ENABLED :** When this parameter is set to `true`, detailed logging of activity by the API mediation layer occurs. This can be useful to diagnose unexpected behavior of the API gateway, API discovery, or API catalog services. Default value is `false`.

Refer to detailed section about [Advanced Gateway features configuration](#) on page 196

Cross memory server

- **ZOWE_ZSS_XMEM_SERVER_NAME:** For the Zowe Desktop to operate communication with the Zowe cross memory server. The default procedure name `ZWESIS_STD` is used for the cross memory server. However, this can be changed in the `ZWESISTC` PROBLIC member. This might occur to match local naming standards, or to allow isolated testing of a new version of the cross memory server while an older version is running concurrently. The Zowe desktop that runs under the `ZWESVSTC` started task will locate the appropriate cross memory server running under its started task `ZWESISTC` using the `ZOWE_ZSS_XMEM_SERVER_NAME` value. If this handshake cannot occur, users will be unable to log in to the Zowe desktop. See [ZSS server unable to communicate with X-MEM](#) on page 440.

```
// ZWESISTC PROC NAME='ZWESIS_STD',MEM=00,RGN=0M
```

Extensions

- **ZWEAD_EXTERNAL_STATIC_DEF_DIRECTORIES:** Full USS path to the directory that contains static API Mediation Layer .yml definition files. For more information, see [Add a definition in the API Mediation Layer in the Zowe runtime](#) on page 326. Multiple paths should be semicolon separated. This value allows a Zowe instance to be configured so that the API Mediation Layer can be extended by third party REST API and web UI servers.
- **EXTERNAL_COMPONENTS:** For third-party extenders to add the full path to the directory that contains their component lifecycle scripts. For more information, see [Zowe lifecycle - Zowe extensions](#).

High Availability

The high availability (HA) feature of Zowe is under development and has not been fully delivered. The following values are work in progress towards HA capability. They are not used and will be documented in more detail once HA support is finalized in a future Zowe release.

- ZWE_DISCOVERY_SERVICES_LIST: (*Work in progress*) **Do not modify this value** from its supplied default of `https:// ${ZOME_EXPLORER_HOST} : ${DISCOVERY_PORT} /eureka/`.
- ZWE_CACHING_SERVICE_PORT=7555: (*Work in progress*) This port is not yet used so the value does not need to be available.
- ZWE_CACHING_SERVICE_PERSISTENT=VSAM: (*Work in progress*)
- ZWE_CACHING_SERVICE_VSAM_DATASET: (*Work in progress*)

Configuring a Zowe instance via `instance.env` file

When configuring a Zowe instance through the `instance.env` file, `ZOME_IP_ADDRESS` and `ZOME_EXPLORER_HOST` are used to specify where the Zowe servers can be reached.

However, these values may not reflect the website name that you access Zowe from. This is especially true in the following cases:

- You are using a proxy
- The URL is a derivative of the value of `ZOME_EXPLORER_HOST`, such as `myhost` versus `myhost.mycompany.com`

In these cases, it may be necessary to specify a value for `ZWE_EXTERNAL_HOSTS` in the form of a comma-separated list of the addresses from which you want to access Zowe in your browser.

In the previous example, `ZWE_EXTERNAL_HOSTS` could include both `myhost` and `myhost.mycompany.com`. In the `instance.env`, this would look like: `ZWE_EXTERNAL_HOSTS=myhost,myhost.mycompany.com`

This configuration value maybe used for multiple purposes, including referrer-based security checks. In the case that the values are not specified, referrer checks will use the default values of `ZOME_IP_ADDRESS`, `ZOME_EXPLORER_HOST`, and the system's hostname. Therefore, if these values are not what you put into your browser, you will want to specify `ZWE_EXTERNAL_HOSTS` to set the correct value.

- `ZOME_EXPLORER_FRAME_ANCESTORS`: The MVS, USS, and JES Explorer are served by their respective explorer UI address spaces. These are accessed through the Zowe desktop where they are hosted as iFrames. To protect against double iFrame security vulnerabilities, browsers all of the valid address that may be used by the browser must be explicitly declared in this property. The default values are: `" ${ZOME_EXPLORER_HOST} : *, ${ZOME_IP_ADDRESS} : * "`. If there are any other URLs by which the Zowe Explorers can be served, then these should be appended to the preceding comma-separated list.

Hints and tips

Learn about some hints and tips that you might find useful when you create and configure the Zowe instance.

When you are configuring Zowe on z/OS, you need to [Configuring Zowe certificates](#) on page 136, and then create the Zowe instance.

The creation of a Zowe instance is controlled by the [Reviewing the `instance.env` file](#) on page 151 in your instance directory `INSTANCE_DIR`.

1. Keystore

Edit the `instance.env` file to set the keystore directory to the one you created when you ran `zowe-setup-certificates.sh`.

The keyword and value in `instance.env` should be the same as in `zowe-setup-certificates.env`, as shown below

```
KEYSTORE_DIRECTORY=/my/zowe/instance/keystore
```

2. Hostname and IP address

The `zowe-configure-instance.sh` script handles the IP address and hostname the same way `zowe-setup-certificates.sh` does.

In `instance.env`, you specify the IP address and hostname using the following keywords:

```
ZOWE_EXPLORER_HOST=
ZOWE_IP_ADDRESS=
```

The `ZOWE_EXPLORER_HOST` value must resolve to the external IP address, otherwise you should use the external IP address as the value for `ZOWE_EXPLORER_HOST`.

The `zowe-configure-instance.sh` script will attempt to discover the IP address and hostname of your system if you leave these unset.

When the script cannot determine the hostname or the IP address, it will ask you to enter the IP address manually during the dialog. If you have not specified a value for `ZOWE_EXPLORER_HOST`, then the script will use the IP address as the hostname.

The values of `ZOWE_EXPLORER_HOST` and `ZOWE_IP_ADDRESS` that the script discovered are appended to the `instance.env` file unless they were already set in that file or as shell environment variables before you ran the script.

Installing and starting 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 Zowe. You can do these manually (as described in this topic) or use scripts to install and configure the cross memory server (see [Installing and Configuring Zowe z/OS components using scripts](#)).

Step 1: Copy the PROCLIB member ZWESVSTC

When the Zowe runtime is launched, it is run under a z/OS started task with the PROCLIB member named `ZWESVSTC`. A sample PROCLIB is created during installation into the PDS `SZWESAMP` (`ZWESVSTC`). To launch Zowe as a started task, you must copy this member to a PDS that is in the proclib concatenation path.

Step 2: Configure ZWESVSTC to run under the correct user ID

The `ZWESVSTC` should be configured as a started task under the `ZWESVUSR` user ID with the administrator user ID of `ZWEADMIN`. If you do not have these IDs already created, the commands to create the user ID and group are supplied in the PDS member `ZWESECUR`. See [Configuring the z/OS system for Zowe](#) on page 126. To associate the `ZWESVSTC` started task with the user ID and group, see [Configuring the z/OS system for Zowe](#) on page 126. 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. To see whether the started task has successfully launched see [Troubleshooting installation and startup of Zowe z/OS components](#) on page 407

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 `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, the JCL will need to know the instance directory containing the launch and configuration information. To do this add the INSTANCE parameter on the START command when you start Zowe in SDSF:

```
/S ZWESVSTC, INSTANCE='$ZOWE_INSTANCE_DIR', JOBNAME='ZWEXSV'
```

The \$ZOWE_INSTANCE_DIR argument is the fully qualified path to the USS directory containing the instance.env file containing the Zowe configuration.

The JOBNAME='ZWEXSV' argument 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.

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 instance directory that contains the configurable Zowe instance directory.

Configure Zowe with z/OSMF Workflows

As a system programmer, after you install Zowe, you can register and execute the z/OSMF workflows in the web interface to complete the Zowe configuration. z/OSMF helps to simplify the Zowe configuration tasks and reduce the level of expertise that is needed for Zowe configuration.

Ensure that you meet the following requirements before you start the Zowe configuration:

- Installed and configured z/OSMF
- Installed Zowe with either SMP/E build or convenience build

You can complete the following tasks with the z/OSMF workflows:

- [Configure z/OS Security Manager](#) on page 157
- [Configure Zowe certificates](#) on page 158
- [Configure Zowe Cross Memory Server](#) on page 158
- [Create and configure the Zowe instance directory and start the Zowe started task](#) on page 159

Configure z/OS Security Manager

Configure the z/OS security manager to prepare for launching the Zowe started tasks. The workflow definition file is provided to assist with the security configuration. The workflow definition file allows you to configure z/OS security manager by using one of RACF, ACF2, or TSS security systems.

Register the **ZWESECUR.xml** workflow definition file in the z/OSMF web interface to configure z/OS security manager. The path to the workflow definition file is <pathPrefix>/workflows/.

Perform the following steps after you register the workflow definition file:

1. Define Values for Variables

Review all the parameters and customize the values for variables to meet the z/OS security requirements. We recommend that the security administrator at your site reviews and edits the values for security group variables.

Zowe package includes the variable input file that is **ZWESECUR.properties**. Optionally, you can use this file to customize the values for variables in advance. Upload the prepared properties file while you register the workflow definition. Values from this file override the default values for the workflow variables.

2. Execute JCL

Execute the step to complete the z/OS security manager configuration.

After you execute these steps, the groups, user IDs and started tasks are assigned based on the customized values. For instructions on how to register and execute the workflow, see [Register and execute workflow in the z/OSMF web interface](#) on page 160.

Configure Zowe certificates

z/OSMF workflow lets you generate certificate signed by the Zowe API Mediation Layer and keystores in the specified location. Zowe uses the keystore directory to hold the certificate to encrypt communication between Zowe clients and the Zowe z/OS servers. The keystore directory also holds the truststore that is used to hold public keys of any servers that Zowe trusts.

Register the ZWEWRF05 member that is located <pathPrefix>/workflows/ZWEWRF05.xml data set in the z/OSMF web interface. After you register the workflow definition file, you can execute the following steps.

1. Define Variables

Review all the parameters and customize the values for variables to meet the z/OS security requirements.

Zowe package includes the variable input file ZWEWRF05.properties and the path is <pathPrefix>/workflows/ZWEWRF05.properties. Optionally you can use this file to customize the values for variables in advance. Upload the prepared properties file when you register the workflow definition file. Values from this file override the default values for the workflow variables.

2. Generate new custom zowe-setup-certificates.env file

Execute the step to generate a new custom zowe-setup-certificates.env file based on the custom values that you provide for variables in the first step.

3. Execute zowe-setup-certificates.sh

Execute the step to run the shell script to generate the custom certificates based on the defined values for variables and values for parameters in the provided environment file.

After you execute these steps, the keystore and certificates are successfully generated based on the custom values. For general instruction on how to register and execute the workflow, see [Register and execute workflow in the z/OSMF web interface](#) on page 160.

Configure Zowe Cross Memory Server

The Zowe cross memory server provides privileged cross-memory services to the Zowe Desktop and runs as an APF-authorized program. Multiple Zowe desktop instances can use the same cross memory server. Use the z/OSMF workflow to install, configure, and launch the cross memory server if you want to use the Zowe desktop. The z/OSMF workflow also lets you create APF-authorized load libraries that are required to install and configure the cross memory server.

Register the ZWEWRF06.xml workflow definition file that is located in <pathPrefix>/workflows/. After you complete the workflow registration, Perform the following steps to configure the Zowe cross memory server:

1. Define values for Variables

The workflow includes the list of Zowe cross memory server configuration and the started task variables. Enter the custom values for variables based on your mainframe environment and Zowe cross memory server configuration requirements.

Zowe package includes ZWEWRF06.properties variable input file and the path is <pathPrefix>/workflows/ZWEWRF05.properties. Optionally you can use this file to customize the values for variables in advance. Upload the prepared properties file when you register the workflow definition file. Values from this file override the default values for the workflow variables.

2. Allocate Cross Memory Server Data Sets

Execute the step to allocate Target DSN for PARMLIB, Target DSN for PROCLIB, and log directory data sets that are required for XMEM configuration.

3. Copy artifacts

Execute the step to populate the data sets that are allocated in the previous step with the necessary artifacts such as load modules, parmlib members and others. This step also copies the cross memory server STC to the proclib.

4. 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 SCHEDEXX member of the system PARMLIB. PPT PGMNAME(ZWESIS01) KEY(4) NOSWAP

The PDS member SZWESAMP contains the PPT lines for reference. Issue the following command to make the SCHEDEXX changes effective. /SET SCH=xx

For more information, see [Key 4 non-swappable](#) on page 148.

5. Retrieve the LOADLIB volume

This step allows you to automatically retrieve the VOLUME for non-SMS LOADLIB. Run this step to retrieve the actual VOLUME of the LOADLIB.

6. APF Authorize Load Library

Creates APF-authorized load library that is required to install and configure the cross memory server. Execute the step to APF authorize the XMEM LOADLIB.

7. Start the XMEM Server

Execute this step to start the Cross Memory Server started task.

After you complete these steps, the Zowe cross memory server is configured and installed to start the Zowe Desktop instance. For instruction on how to register and execute the workflow, See, [Register and execute workflow in the z/OSMF web interface](#) on page 160.

Create and configure the Zowe instance directory and start the Zowe started task

The Zowe instance directory contains configuration data that is 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 is read from files in the instance directory and logs will be written to files in the instance directory. Zowe has three runtimes namely: the z/OS Service microservice server, the Zowe Application Server, and the Zowe API Mediation Layer microservices.

Register **ZWEWRF03.xml** workflow definition file in the z/OSMF web interface to create a Zowe instance directory and start the Zowe started task. The path to the workflow definition file is <pathPrefix>/workflows/

After you register the workflow definition file, perform the following steps to complete the process:

1. Define Variables

The workflow includes the list of instance configuration and the Zowe started task variables. Enter the values for variables based on your mainframe environment, Zowe instance configuration, and started task requirements.

Zowe package includes the variable input file that is **ZWEWRF03.properties** and the path is <pathPrefix>/files/workflows/ZWEWRF03.properties. Optionally you can use this file to customize the values for

variables in advance. This automates the workflow execution, saving time and effort when deploying multiple standardized Zowe instances. Values from this file override the default values for the workflow variables.

2. Create a Zowe instance

Execute the step to create a Zowe instance directory. This step creates instances for all the micro services. That is z/OS Service microservice server, the Zowe Application Server, and the Zowe API Mediation Layer microservices.

3. Change the instance configuration

Execute the step to configure the Zowe instance. The configuration of the Zowe instance depends on the values for variables that you defined in the first step. **Note: If you are planning to use Docker, be sure to select only to start LAUNCH_COMPONENT_GROUP=ZSS, otherwise more components of Zowe than necessary will be run on z/OS, such as API Mediation Layer and the App Framework. You can skip configuration for those components here, as they will run in Docker.**

4. Copy the STC to the procedure library

Skip this step if the procedure library is empty.

5. Start the Zowe instance

Execute the step to start the instance.

After you execute each step, the step is marked as Complete. After completing the workflow execution, you can view the Zowe started task.

Register and execute workflow in the z/OSMF web interface

z/OSMF workflow simplifies the procedure to configure and start Zowe. Perform the following steps to register and execute the workflow in the z/OSMF web interface:

1. Log in to the z/OSMF web interface and select **Use Desktop Interface**.
2. Select the Workflows File.
3. Select **Create Workflow** from the **Actions** menu.

The **Create Workflow** panel appears.

4. Enter the complete USS path to the workflow you want to register in the **Workflow Definition File** field.

- If you installed Zowe with the SMP/E build, the workflow is located in the SMP/E target zFS file system that was mounted during the installation.
- (Optional) Enter the complete USS path to the edited workflow properties file in the Workflow Variable Input File field. Use this file to customize product instances and automate workflow execution, saving time and effort when deploying multiple standardized Zowe instances. Values from this file override the default values for the workflow variables.

The sample properties file is located in the same directory with the workflow definition file. Create a copy of this file, and then modify as described in the file. Set the field to the path where the new file is located. Note: if you use the convenience build, the workflows and variable input files are located in the USS runtime folder in files/workflows

The following table provides the list of Zowe Components Workflow Definition files and their corresponding variable input files.

Configuration Tasks	Workflow Definition File Name	Properties File Name	Workflow Definition File Path	Variable Input file Path
Configure z/OS Security Manager	ZWESECUR.xml	ZWESECUR.properties<pathPrefix>/workflows//ZWESECUR.xml	<pathPrefix>/workflows/ZWESECUR.properties	
Configure Zowe Certificates	ZWEWRF05.xml	ZWEWRF05.properties<pathPrefix>/workflows//ZWEWRF05.xml	<pathPrefix>/workflows/ZWEWRF05.properties	

Configuration Tasks	Workflow Definition File Name	Properties File Name	Workflow Definition File Path	Variable Input file Path
Configure Cross Memory Server	ZWEWRF06.xml	ZWEWRF06.properties<pathPrefix>/workflows//ZWEWRF06.xml	<pathPrefix>/workflows/ZWEWRF06.xml	ZWEWRF06.properties
Create Instance Directory and Start the Zowe started Task	ZWEWRF03.xml	ZWEWRF03.properties<pathPrefix>/workflows//ZWEWRF03.xml	<pathPrefix>/workflows/ZWEWRF03.xml	ZWEWRF03.properties

1. Select the System where the workflow runs.
2. Select **Next**.
3. Specify a unique Workflow name.
4. Select or enter an Owner user ID, and select **Assign all steps to owner user ID**.
5. Select Finish.

The **workflow** is registered in z/OSMF. The workflow is available for execution to deploy and configure the Zowe instance.

6. Execute the steps in order. 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**.

Repeat the previous two steps to complete all items until the option **Finish** is available.

7. Select **Finish**.

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

Verifying Zowe installation on z/OS

After the Zowe™ started task ZWESVSTC is running, follow the instructions in the following sections to verify that the components are functional.

- [Verifying Zowe Application Framework installation](#) on page 161
- [Verifying API Mediation installation](#) on page 162
- [Verifying z/OS Services installation](#)

Note: Not all components may have been started. Which components have been started depends on your setting of the variable LAUNCH_COMPONENT_GROUPS in the `instance.env` file. If you defined the value GATEWAY, the API Mediation Layer and z/OS Services are started. If you defined the value DESKTOP, the Zowe Application Framework (also known as Zowe desktop) is started. Those using Docker may only have ZSS started. For more information, see [Component groups](#) on page 151.

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` where,

- *myHost* is the host on which you installed the Zowe Application Server.

- *httpsPort* is the port number value ZOWE_ZLUX_SERVER_HTTPS_PORT in `instance.env`. For more information, see [Ports](#) on page 153.

For example, if the Zowe Application Server runs on host *myhost* and the port number that is assigned to ZOWE_ZLUX_SERVER_HTTPS_PORT is 12345, you specify `https://myhost:12345`. The web desktop uses page direct to the actual initial page which is `https://myhost:12345/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html`. If the redirect fails, try the full URL.

If the desktop appears but you are unable to log on, check [Cannot log in to the Zowe Desktop](#) on page 439 for troubleshooting tips.

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://myhost:httpsPort/api/v1/apicatalog/application/health
```

where,

- *myHost* is the host on which you installed the Zowe API Mediation Layer.
- *httpsPort* is the port number value GATEWAY_PORT in `instance.env`. For more information, see [Ports](#) on page 153.

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://myhost:httpsPort/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.

Verifying z/OS Services installation

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 GATEWAY_PORT in the `instance.env` file used to launch Zowe. For more information, see [Ports](#) on page 153.

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.

Upgrading the z/OS system for Zowe

If you installed Zowe previously, the system is already prepared and configured to launch a Zowe instance. A Zowe configuration consists of three USS directories. See [Topology of the Zowe z/OS launch process](#) for more information.

- The runtime directory that contains the binary and executable files. See [RUNTIME_DIR](#) on page 71 for more information. This directory is read only and contains a `manifest.json` file that can be used to identify its release number. See [Check the Zowe release number](#) on page 401 for more information. A new Zowe runtime directory is created when a new version of Zowe is installed.
- The instance directory that is used to launch the Zowe started task ZWESVSTC. See [INSTANCE_DIR](#) on page 72 for more information. The instance directory is read/write as it contains log files and the file `instance.env` that contains environment launch values.
- The keystore directory that contains details about certificates used by Zowe. See [INSTANCE_DIR](#) on page 72 for more information.

Zowe installation also creates two data sets.

- **SZWESAMP**

This contains 10 members. Four of them are JCLs to configure (and unconfigure Zowe). Three are PROCLIB members. One contains PPT entries. One is a PARMLIB member and one contains console commands. See [Step 3: Choose a dataset HLQ for the SAMPLIB and LOADLIB](#) on page 105 for more information.

- **SZWEAUTH**

This is an APF-authorized PDSE load library containing the load modules ZWESIS01 for the cross memory server and ZWESAUX for the auxiliary address space. See [Step 3: Choose a dataset HLQ for the SAMPLIB and LOADLIB](#) on page 105 for more information.

Upgrading the instance directory

A Zowe instance directory is created using the script `zowe-configure-instance.sh` in the `<RUNTIME_DIR>/bin` directory.

After installing a new version of Zowe, the new runtime will either be in the same directory as the previous version, for example, `/usr/lpp/zowe` if SMP/E has been used, or else may be in a different directory, for example, `~/zowe/zowe-r.M.m` if using a convenience build installation.

In both situations, you can keep and reuse the instance directory that is used for the previous version of Zowe to launch the new version of Zowe. To do this, run the script `zowe-configure-instance.sh` from the new `<RUNTIME_DIR>/bin` directory with the `-c` argument pointing to the location of the existing instance directory. This is the same method used to create an instance directory with default values in an empty target directory, however if `-c` argument is a pre-existing instance directory rather than wiping and creating fresh contents, the contents are updated. In the situation where the previous instance directory was created from a different runtime directory, the `ROOT_DIR=` value in `instance.env` will be updated to reference the `<RUNTIME_DIR>` from which `zowe-configure-instance.sh` was executed. In addition the `manifest.json` file in the instance directory will be updated with the "version:" of the `<RUNTIME_DIR>`. This can be used as a way to see the Zowe version that an instance was last configured from. See [Check the Zowe release number](#) on page 401.

The `zowe-configure-instance.sh` script will detect if there are new configuration values that have been introduced since the instance directory was last created, and append these to `instance.env` with default values. New values added will be echoed in the shell running the `zowe-configure-instance.sh` script, and are described in [Reviewing the instance.env file](#). Values in `instance.env` previously changed from their default, such as port values or locations of dependent runtimes, are not modified.

The zowe-configure-instance.sh script will echo any values that are added to the instance.env file.

```
Missing properties that will be appended to /u/winchk/zowe-instance/
instance.env:
```

Updating the PROCLIB members

Zowe releases contain two proclib members, ZWESISTC and ZWESVSTC in the PDS SZWESAMP. When the previous release of Zowe was installed, these would have been copied to a PDS in the proclib concatenation path and defined to run under their respective user IDs of ZWESVUSR and ZWESIUSR. See [Installing and starting the Zowe started task \(ZWESVSTC\)](#) on page 156 and [Installing and configuring the Zowe cross memory server \(ZWESISTC\)](#) on page 146.

The proclib members do not usually get updated between Zowe releases, so during an upgrade you may keep the previous JCL proclibs. If the proclib members are updated then the HOLDDATA and the [Release notes](#) on page 20 will describe any changes and alert you that the proclibs need to be updated. If you are upgrading Zowe and jumping releases, for example moving from 1.12 to 1.16, then you should check the HOLDDATA (for SMP/E) and the release notes for all intervening releases (1.12, 1.13, 1.14, 1.15) to see if the proclibs have changed.

Updating the cross memory server load modules

Zowe releases contain two load modules, ZWESIS00 for the cross memory server and ZWESAUX for the auxiliary server. There are delivered in a PDSE SZWEAAUTH that is created by the installation process.

If this PDSE is the same one used by the ZWESISTC proclib that starts the cross memory server, then because the installation replaces the data set and its contents no action is required. This is the recommended approach to configure a Zowe environment.

If you have copied the SZWEAAUTH members to another PDSE that you are using as a runtime load module, then you should recopy the updated members to the runtime location.

Updating the system and security configuration

The JCL member ZWESECUR delivered in the PDS member SZWESAMP contains the TSO commands used to configure a z/OS environment for launching Zowe.

The contents of ZWESECUR do not usually get updated between Zowe releases, so during an upgrade you should not need to rerun the JCL. If there are additions, then the HOLDDATA for the SMP/E release and the [Release notes](#) on page 20 will describe the changes.

Upgrading the keystore directory

When the previous release of Zowe was configured, a keystore directory would have been created. This would either contain the Zowe certificate, or else reference a SAF keyring that contains the Zowe certificate. See [Configuring Zowe certificates](#) on page 136. The USS keystore directory is created using the script <RUNTIME_DIR>/bin/zowe-setup-certificates.sh that is delivered with the new Zowe release, and creates the USS file <KEYSTORE_DIRECTORY/zowe-setup-certificates.env containing key value parameters used by the Zowe runtime to locate its certificate.

Typically, the <KEYSTORE_DIRECTORY> is compatible with later versions and can be used when moving forward to a new Zowe release. There are situations when new functionality is introduced into a Zowe release when new key values pairs may be introduced to the <KEYSTORE_DIRECTORY>/zowe-setup-certificates.env file, in which case the new release will describe the new functionality in its HOLDDATA (for SMP/E) or release notes. If this occurs, it will be necessary to create a new KEYSTORE_DIRECTORY in order to use the new functionality. This was the case when Zowe 1.15 introduced support for storing certificates in a SAF keyring.

Zowe provides a JCL member SZWESAMP (ZWEKRING) to create the keystore and populate it with the Zowe certificate. Unless instructed by the HOLDDATA or release notes, there is no need to re-create the keystore or certificate and ones used with previous Zowe releases can be reused.

Service disruption during upgrades

When Zowe is upgraded, the started tasks ZWESVSTC and ZWESISTC need to be stopped and started for the changes to take effect. This will cause active operations to fail. Idle, or passive operations, such as an open webpage on the Zowe desktop may continue to work without disruption under the following circumstances:

- SSO is being used
- SSO is not being used, but a new page in the browser is opened to re-login to Zowe

In both cases, the original webpages will have access to currently valid credentials, allowing the web page to issue new operations without reloading a page. However, in-process operations that failed during the upgrade, unless retired manually or automatically may necessitate a page reload.

Zowe extensions

Zowe extensions can depend on servers, databases, and other software that is outside the scope of Zowe. Therefore, you should refer to each plug-in's documentation to understand what to expect when an upgrade occurs.

Zowe desktop

A system programmer with admin rights may invoke the role base access control (RBAC) controlled App Server endpoint /server/reload to reload the server without the need for an end user webpage reload. This endpoint and more are documented in swagger [here](#).

If the plug-in uses the ZSS server to provide a REST API, ZSS must be restarted causing the same disruption behavior as a Zowe server upgrade.

If the plug-in has web content that shows in the Zowe desktop without an iframe, you must reload the desktop to see the updated content.

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
```

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 152

After 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 Zowe installation, the z/OS environment is altered to allow Zowe to operate. See [Configuring the z/OS System for Zowe](#) for details. 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. Zowe provides a JCL member ZWENOSEC 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.

5. Reverse the z/OS key ring updates from ZWEKRING job.

The ZWEKRING JCL member provided in the SZWESAMP member can be used to create a key ring that contains the Zowe certificate(s) and certificate authority. If you want to remove the key ring and its certificate(s) and certificate authority, you can use the JCL member ZWENOKYR that contains the undo steps to reverse the configuration performed in ZWEKRING.

Installing Zowe Docker Bundle

Docker Installation Roadmap (Technical Preview)

The Zowe Docker build is a technical preview. Technical previews are for testing only and not ready for production. Any feedback that you can provide is highly appreciated.

There are three parts to using Docker to install ZoweTM.

- The Zowe z/OS runtime which contains the ZSS component.
- The Zowe Cross Memory Server. This is an authorized server application that provides privileged services to Zowe in a secure manner.
- The Zowe Docker image, which runs on a Linux or zLinux host. It consists of three components: Zowe Application Framework, z/OS Explorer Services, and Zowe API Mediation Layer.

Using the Zowe Docker bundle first requires setting up your z/OS system for Zowe. The steps for z/OS setup are detailed on the page: [z/OS Installation Roadmap](#) on page 99.

NOTE: If you want to install all server components on z/OS instead of using Docker, completing the z/OS install instructions in the above document is all that is required.

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.

NOTE: 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

SMP/E build

What is your preferred installation method?

Conv

Download the Zowe SMP/E build

Shell script

Choose a method to install the SMP/E build

z/OSMF workflow

Install the Zowe SMP/E build using JCLs

Install the Zowe SMP/E build with z/OSMF workflow
ZWEWRF01

Shell script

Run shell script
zowe-install

Stage 3:
Configure the Zowe runtime and start Zowe

Choose a method to configure Zowe

JCL, Shell scripts

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 70 for details.

Stage 2: Install the Zowe runtime on z/OS

Complete the tasks in [Stage 2 of z/OS Installation Roadmap](#).

Stage 3: Configure the Zowe z/OS runtime

First you must complete the tasks in [Stage 3 of z/OS installation Roadmap](#).

After, you should edit or review the chosen Component groups in the Zowe instance directory that was created. For use with Docker, only the Component group ZSS is required. This means that at minimum, the file `instance.env` will have the value `LAUNCH_COMPONENT_GROUPS=ZSS` set. See [Component groups](#) on page 151.

Stage 4: Verify the installation

Verify that Zowe is installed correctly on z/OS. See [Verifying Zowe installation on z/OS](#) on page 161.

Stage 5: Install Docker image

Get the latest Docker Image for the Zowe Server Components. See [Installing Zowe runtime Docker Image \(Technical Preview\)](#) on page 169.

Stage 6: Configure Docker container

Extract and customize the start script, instance directory, and keystore before running a Docker container. See [Configuring Zowe runtime Docker Container \(Technical Preview\)](#) on page 170.

Looking for troubleshooting help?

If you encounter unexpected behavior when installing or verifying the Zowe runtime on z/OS, see the [Troubleshooting](#) on page 400 section for tips.

Installing Zowe runtime Docker Image (Technical Preview)

Docker is a way to create a pre-packaged set of software and configuration called an "Image". Images are used to create Docker "Containers", which run the Image contents in isolation from the other software running on the same system. Docker containers are the runtime environment, and Images are what they are created from.

The majority of the Zowe server runtime is available in the form of a Docker Image, among other options. To use this image, you must have set up the Zowe server runtime on z/OS, z/OSMF, or both depending on which Zowe components you will use.

If you have not set up the Zowe server runtime on z/OS, please follow the steps found in [Docker Installation Roadmap \(Technical Preview\)](#) on page 166.

This guide assumes you are using Linux or zLinux and have already downloaded Docker itself. If you have not yet done so, please review [System requirements](#) on page 72.

Installing via Docker Hub

Zowe's Docker Image is hosted on [Docker Hub](#), which is the default location from which you can use the Docker command line utility to download and update Docker Images. On Docker Hub, the Zowe server runtime image is named `ompzowe/server-bundle`.

You can download a Docker Image by using the Docker command line utility `docker pull imagename` where `imagename` is one of the following:

- The latest version of zowe, `ompzowe/server-bundle:latest`
- The latest version for the platform you are running on, such as `ompzowe/server-bundle:amd64` for Linux

- A specific version by referencing the version's digest, such as `ompzowe/server-bundle@sha256:bdbc0617b02e16a452f6d4de50b8b13e56592e309b4c68f9ea52c82303ad57ec`

The latest digests can be seen on the [image's tags page](#).

Installing via direct download

You can install a Docker Image that has been downloaded as a `.tar` archive from anywhere, such as [Zowe.org](#).

Loading an image from .tar file

To install a Docker Image that you have downloaded as a tar file from somewhere, transfer the file to the destination host and then run the following command: `docker image load -i path_to_tar`

Confirming the installation

The `docker images` command lists the images a system currently has, which make them available for creating containers from.

# docker images			
REPOSITORY	SIZE	TAG	IMAGE ID
ompzowe/server-bundle	1.27GB	amd64	ceb8c50d2381

CREATED 2 hours ago

Upgrading

Once installed, it is possible to upgrade an image by using `docker pull` with the same *imagename* as before, or by using `docker image load` to load another image of the same type. Newer containers can be created from newer images. In Zowe, configuration can be persisted between containers. More information on this subject can be found in [Configuring Zowe runtime Docker Container \(Technical Preview\)](#) on page 170 documentation.

When upgrading, it is possible that the previous image may persist. You may see the old image tagged as `<none>`.

# docker images			
REPOSITORY	SIZE	TAG	IMAGE ID
ompzowe/server-bundle	1.27GB	amd64	ceb8c50d2381
2 hours ago			
<code><none></code>		<code><none></code>	1e52fadcd918
weeks ago	3.03GB		2

If you see this and want to clean up the older images to preserve storage space, you can run the command `docker rmi IMAGE_ID` to remove an image, where `IMAGE_ID` is the code seen from the `images` command.

Configuring Zowe runtime Docker Container (Technical Preview)

Configuring the Zowe runtime Docker Image has similarities to [Creating and configuring the Zowe instance directory](#) on page 150. However, there are three major differences:

- Ports are managed between Docker and the host rather than in the `instance.env` file
- Plugins can be added from the host by using a Docker mount
- External certificates can be used from a Docker mount

Working with Docker mounts

Docker has a feature called a "mount", which allows you to share a folder from the host system into one or more containers. Zowe can use the mount feature in order to share important settings and content, such as certificates and plugins between multiple instances of Zowe. Additionally, mounts keep these objects intact when upgrading between Zowe versions.

Quick start for the Zowe runtime in Docker

The Zowe Docker Image comes with a sample script for starting a container of Zowe, plus a basic instance configuration.

1. Note the ID of the image.

```
# docker images
REPOSITORY           TAG      IMAGE ID
CREATED             SIZE
ompzowe/server-bundle    amd64   ceb8c50d2381
2 hours ago          1.27GB
```

1. Start a container of the image without starting Zowe yet by using the environment value `ZOWE_START=0`

```
# docker run -it --env ZOWE_START=0 ceb8c50d2381 &
```

1. Make note of the new container's ID.

```
# docker container list
CONTAINER ID        IMAGE               COMMAND            CREATED
STATUS              NAMES
7664336131e9       ceb8c50d2381      "/root/zowe/run.sh"   3
minutes ago         Up 3 minutes      7553-7554/tcp, 8544/tcp
```

1. Copy the samples out.

```
# docker cp 7664336131e9:/root/zowe/samples/start.sh .
# docker cp 7664336131e9:/root/zowe/instance .
```

This will generate a sample script for running docker containers, `start.sh`, and a sample instance as the folder `instance`.

It is recommended to customize `start.sh`, however test installs can skip to [Starting the container](#) on page 172.

Within `start.sh`, you will be able to see parameters to customize ports, specify which Zowe components to start, specify where the z/OS system is located, and more.

Note: The Zowe keystore cannot be copied in this way because it does not exist initially. If you need to initialize a keystore, you can start Zowe in the container temporarily by omitting `ZOWE_START=0` and run a `docker cp` command to copy out `/global/zowe/keystore` to make desired edits.

Customizing Zowe container start script

There are many different ways to configure a Zowe docker container:

- `-h <hostname>` - hostname of docker host (hostname of your laptop, for example, `myhost.acme.net`)
- `--env ZOWE_IP_ADDRESS=<ip>` - The IP which the servers should bind to. Should not be a loopback address.
- `--env ZOSMF_HOST=<zosmf_hostname>` - z/OSMF hostname (for example, `mf.acme.net`)
- `--env ZOSMF_PORT=<zosmf_port>` - z/OSMF port (for example, 1443)
- `--env ZWED_agent_host=<zss_hostname>` - ZSS host (for example, `mf.acme.net`)
- `--env ZWED_agent_http_port=<zss_port>` - ZSS port z/OSMF port (for example, 60012).
- `--env LAUNCH_COMPONENT_GROUPS=<DESKTOP or GATEWAY>` - What component you want to start.
 - DESKTOP - only desktop
 - GATEWAY - only GATEWAY + explorers
 - GATEWAY,DESKTOP - both
- `-v [LOCAL_KEYSTORE]:/root/zowe/certs:rw` - Uses external keys and certificates for HTTPS. Keystore directory structure is the same as with Zowe on z/OS.

- `--env EXTERNAL_CERTIFICATE=<keystore.p12>` - location of p12 keystore. (optional)
- `--env EXTERNAL_CERTIFICATE_ALIAS=<alias>` - valid alias within keystore. (optional)
- `--env EXTERNAL_CERTIFICATE_AUTHORITIES=<CA.cer>` - location of x509 Certificate Authority (optional)
- `-v [LOCAL_APPS_DIR]:/root/zowe/apps:ro` - Adds App Framework Apps to the container.
- `-v [LOCAL_INSTANCE_DIR]:/root/zowe/external_instance:rw` - (Recommended) Uses a Zowe instance directory from outside the container. Recommended to save preferences between upgrades and to have multiple containers of Zowe sharing configurations.
- `--env EXTERNAL_INSTANCE=/root/zowe/external_instance` - Used together with the `-v` command to use an external instance directory.

Note: External certificates are optional, but recommended to resolve self-signed certificate warnings.

Using an instance directory external to the Zowe container

Each Zowe container comes with a simple instance directory setup, but it is recommended that this only be used for development, as changes made to the instance will not remain after upgrade and it prevents sharing configuration across multiple containers.

Instead, having an instance directory external to the container solve these issues.

If you have migrated the instance directory from z/OS, copied the simple instance directory from the container, or otherwise have a pre-existing instance directory, you can use it with a Zowe container by using a volume mount command in the start script.

```
-v [LOCAL_INSTANCE_DIR]:/root/zowe/external_instance:rw \
--env EXTERNAL_INSTANCE=/root/zowe/external_instance \
```

See [Creating and configuring the Zowe instance directory](#) to review options for instance directory configuration.

Using external certificates

Zowe's keystore can be used to configure which keys and certificates will be used by Zowe for HTTPS connections.

The keystore directory configuration and functionality for Docker is identical to the configuration and functionality on z/OS, except for limitations on storage types. Currently, the Zowe bundle Docker image only supports file-based keys and certificates, such as P12 and PEM files.

To use external certificates, the [Using an instance directory external to the Zowe container](#) on page 172 must set the value of KEYSTORE_DIRECTORY to `/root/zowe/certs`, and the keystore directory should exist outside of Docker but mounted to each Docker container via the `-v` docker command, for example `-v [LOCAL_KEYSTORE]:/root/zowe/certs:rw`

See [Configuring Zowe certificates](#) on page 136 for more information.

Starting the container

The recommended way to start your first container is by running the `start.sh` script. You can choose to run it with `nohup`, `&`, or the `docker --detach` command as ways to run the container independent of the terminal.

When the container is running, the servers' log output may be printed to the screen depending on the above commands, but the servers will also log to a folder within the instance directory, `$INSTANCE_DIR/logs`.

After startup, you can verify that Zowe is running by opening the browser to:

- API Mediation Layer: `https://your_hostname:7554`
- App Framework: `https://your_hostname:8544`

Or, if the ports were modified, `https://your_hostname:$GATEWAY_PORT` and `https://your_hostname:$APP_SERVER_PORT`

Using Zowe-based products, plugins and apps

To use Zowe-based software with the docker container, you must make that software visible to the Zowe that is within Docker by mapping a folder on your host machine to a folder visible within the docker container.

To share a host directory *HOST_DIR* into the docker container destination directory *CONTAINER_DIR* with read/write access, simply add this line to your docker run command: `-v [HOST_DIR]:[CONTAINER_DIR]:ro`

You can have multiple such volumes, but for Zowe Application Framework plugins, the value of *CONTAINER_DIR* must be `/root/zowe/apps`

Application Framework plugins within the root directory of `/root/zowe/apps` will be automatically installed at start up, but if you have a product which has subdirectories of plugins, it may need manual installation or configuration.

For those plugins, you can run their scripts or Zowe's own `install-app.sh` script by executing a command in Docker, such as: `docker exec -it [CONTAINER_ID] /root/zowe/instance/bin/install-app.sh [APPLICATION_DIR]`

Note: When installing Application Framework plugins, you can attempt to load them without a server restart via either clicking "Refresh Applications" in the launchbar menu of the Zowe Desktop, or by doing an HTTP GET call to `/plugins?refresh=true` to the app server. Some plugins may still need a server restart. Consult product documentation for specifics.

Zowe's docker mount locations

When attempting to share certificates, plugins, or instance configuration to a Zowe container, the mount destination is fixed and therefore the following must be used:

- Certificates: The Zowe keystore destination must be `/root/zowe/certs`
- App framework plugins: The folder that contains all plugins must be `/root/zowe/apps`
- Instance configuration: The folder that contains the contents of a Zowe `$INSTANCE_DIR` must be `/root/zowe/external_instance`

Installing Zowe CLI

Installing Zowe CLI

Install Zowe™ CLI on your computer.

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 61. You can learn about new CLI features in the [Release notes](#) on page 20.

Methods to install Zowe CLI

Use one of the following methods to install Zowe CLI.

- [Installing Zowe CLI from a local package](#) on page 173
- [Installing Zowe CLI from an online registry](#) on page 174

Note: If you do not have access to the public npm registry at your site, you might want to install the CLI via a proxy server. See [Install CLI from Online Registry Via Proxy](#) on page 175 for more information.

If you encounter problems when you attempt to install Zowe CLI, see [Troubleshooting Zowe CLI](#) on page 451.

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. Address the following software requirements for the core CLI:

- **Node.js:** Install a currently supported Node.js LTS version. For an up-to-date list of supported LTS versions, see [Nodejs.org](#).
 ::: 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. :::
- **npm:** Install a version of Node Package Manager (npm) that is compatible with your version of Node.js.
 ::: tip Npm is included with most Node.js installations. Issue the command `npm --version` to check your current version. You can reference the [Node.js release matrix](#) to verify that the versions are compatible. :::

1. (Linux only) Address the following software requirements for Secure Credential Storage:

- **(Graphical Linux)** Install `gnome-keyring` and `libsecret` on your computer.
- **(Headless Linux)** Follow the procedure documented in the [SCS plug-in Readme](#).

2. Navigate to [Zowe.org Downloads](#) and click the **CLI Core** button to download the core package. The "core" includes Zowe CLI and Secure Credential Store, which enhances security by encrypting your username and password.

A file named `zowe-cli-package-v.r.m.zip` is downloaded to your computer

3. (Optional) Click the **CLI Plugins** button to download the optional plugins.

A file named `zowe-cli-plugins-v.r.m.zip` is downloaded to your computer.

4. Unzip the contents of `zowe-cli-package-v.r.m.zip` (and optionally `zowe-cli-plugins-v.r.m.zip`) to a preferred location on your computer.

5. Open a command-line window. Issue the following commands in sequence against the extracted directory to install core Zowe CLI on your computer:

```
npm install -g zowe-cli.tgz
```

```
zowe plugins install secure-credential-store-for-zowe-cli.tgz
```

Notes:

- If the command returns an `EACCES` error, refer to [Resolving EACCESS permissions errors when installing packages globally](#) in the npm documentation.
- On Linux, you might need to prepend `sudo` to your npm commands. For more information, see [Troubleshooting Zowe CLI](#) on page 451.

6. (Optional) Address the [Software requirements for Zowe CLI plug-ins](#) on page 239. You can install most plug-ins without meeting the requirements, but they will not function until you configure the back-end APIs. The IBM Db2 plug-in requires [Installing](#) on page 245.

7. (Optional) Issue the following command to install each available plug-in:

```
zowe plugins install cics-for-zowe-cli.tgz db2-for-zowe-cli.tgz zos-ftp-for-zowe-cli.tgz ims-for-zowe-cli.tgz mq-for-zowe-cli.tgz
```

Important: Ensure that you meet the [Software requirements for Zowe CLI plug-ins](#) on page 239. You can install most plug-ins without meeting the requirements, but they will not function until you configure the back-end APIs. The IBM Db2 plug-in requires [Installing](#) on page 245.

Zowe CLI is installed on your computer. Issue the `zowe --help` command to view a list of available commands. For information about how to connect the CLI to the mainframe, create profiles, integrate with API ML, and more, see [Using CLI](#).

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. Address the following software requirements for the core CLI:

- **Node.js:** Install a currently supported Node.js LTS version. For an up-to-date list of supported LTS versions, see [Nodejs.org](#).
 ::: 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. :::
- **npm:** Install a version of Node Package Manager (npm) that is compatible with your version of Node.js.
 ::: tip Npm is included with most Node.js installations. Issue the command `npm --version` to check your current version. You can reference the [Node.js release matrix](#) to verify that the versions are compatible. :::

1. (Linux only) Address the following software requirements for Secure Credential Storage:

- **(Graphical Linux)** Install `gnome-keyring` and `libsecret` on your computer.
- **(Headless Linux)** Follow the procedure documented in the [SCS plug-in Readme](#).

2. Issue the following commands in sequence to install the core from the public npm registry. The "core" includes Zowe CLI and Secure Credential Store, which enhances security by encrypting your username and password.

```
npm install -g @zowe/cli@zowe-v1-lts
```

```
zowe plugins install @zowe/secure-credential-store-for-zowe-cli@zowe-v1-lts
```

Notes:

- If the command returns an EACCESS error, refer to [Resolving EACCESS permissions errors when installing packages globally](#) in the npm documentation.
 - On Linux, you might need to prepend `sudo` to your npm commands. For more information, see [Troubleshooting Zowe CLI](#) on page 451.
- 3. (Optional)** Address the [Software requirements for Zowe CLI plug-ins](#) on page 239. You can install most plug-ins without meeting the requirements, but they will not function until you configure the back-end APIs. The IBM Db2 plug-in requires [Installing](#) on page 245.
- 4. (Optional)** To install all available plug-ins to Zowe CLI, issue the following command:

```
zowe plugins install @zowe/cics-for-zowe-cli@zowe-v1-lts @zowe/db2-for-zowe-cli@zowe-v1-lts @zowe/ims-for-zowe-cli@zowe-v1-lts @zowe/mq-for-zowe-cli@zowe-v1-lts @zowe/zos-ftp-for-zowe-cli@zowe-v1-lts
```

Zowe CLI is installed on your computer. Issue the `zowe --help` command to view a list of available commands. For information about how to connect the CLI to the mainframe, create profiles, integrate with API ML, and more, see [Using CLI](#).

Install CLI from Online Registry Via Proxy

This topic describes how to install Zowe CLI using the NPM install command when you are working behind a proxy server. You will need to use this installation method if your site blocks access to public npm.

You can install Zowe CLI from an online registry via proxy on Windows, macOS, or Linux operating systems:

- This method requires access to an internal server that will allow you to connect to the appropriate registries. For other installation methods, see [Installing CLI](#).
- Your default registry must be public npm (or a mirror of public npm).
- If you previously installed the CLI and want to update to a current version, see [Update Zowe CLI](#).

Follow these steps:

- Identify the proxy server, including the IP address or hostname and the port number.

- If your proxy server **does not** require login credentials, issue the following commands to add the proxy URL to the NPM config file:

```
npm config set https-proxy http://proxy.[proxy_name].com:[port_number]
```

```
npm config set proxy http://proxy.[proxy_name].com:[port_number]
```

where is the IP or hostname and *is* the port number of the proxy server.

- If your proxy server **does** require login credentials, issue the following commands to add the proxy URL, with login credentials, to the NPM config file:

```
npm config set https-proxy http://[username]:[password]@proxy.[proxy_name].com:[port_number]
```

```
npm config set proxy http://[username]:[password]@proxy.[proxy_name].com:[port_number]
```

where and *are* the required login credentials, *is* the IP or hostname, and *is* the port number of the proxy server.

- Ensure that you meet the [Software requirements for Zowe CLI plug-ins](#) on page 239.

- To install Zowe CLI, issue the following command. On Linux, you might need to prepend sudo to your npm commands:

```
npm install @zowe/cli@zowe-v1-lts -g
```

- Install the Secure Credential Store, which lets you store your username, password, and other sensitive information in the credential vault on your computer instead of plaintext. Issue the following command:

```
zowe plugins install @zowe/secure-credential-store-for-zowe-cli@zowe-v1-lts
```

- (Optional)** To install open-source Zowe plug-ins, issue the following command:

```
```
zowe plugins install @zowe/cics-for-zowe-cli@zowe-v1-lts @zowe/ims-for-zowe-cli@zowe-v1-lts @zowe/mq-for-zowe-cli@zowe-v1-lts @zowe/zos-ftp-for-zowe-cli@zowe-v1-lts @zowe/db2-for-zowe-cli@zowe-v1-lts
````
```

Zowe CLI is installed.

- (Optional)** Verify that a Zowe plug-in is operating correctly.

```
zowe plugins validate [my-plugin]
```

where *is* the syntax for the plugin such as @zowe/cics@zowe-v1-lts

- (Optional)** Test the connection to z/OSMF. See [Testing Connection to z/OSMF](#)

- (Optional)** Access the Zowe CLI Help (`zowe --help`) or the Zowe CLI Web Help for a complete reference of Zowe CLI. After you install the CLI, you can connect to the mainframe directly issuing a command, by creating user profiles and making use of them on commands, or by using environment variables. For more information, see [Using CLI](#).

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.

- [Migrating to Long-term Support \(LTS\) version](#) on page 177
- [Identify the currently installed version of Zowe CLI](#) on page 177
- [Identify the currently installed versions of Zowe CLI plug-ins](#) on page 178
- [Update Zowe CLI from the online registry](#) on page 178
- [Update or revert Zowe CLI to a specific version](#) on page 178
- [Update Zowe CLI from a local package](#) on page 178

Migrating to Long-term Support (LTS) version

If you have an @lts-incremental version of Zowe CLI (Zowe v1.0.x - v1.8.x), you can update to @zowe-v1-lts (LTS version) to leverage new functionality and plug-ins.

Follow these steps:

1. Perform *one* of the following steps:
 - a. Delete the ~/.zowe/profiles directory from your computer. You can recreate the profiles manually after you update the CLI.
 - b. If you want to preserve your existing profiles, copy the contents of ~/.zowe/profiles or %homepath%\zowe\profiles to another directory on your computer.
2. Delete the ~/.zowe/plugins or %homepath%\zowe\plugins directory to uninstall all plug-ins.
3. Issue the following command to uninstall the pre-LTS version of core CLI

```
npm uninstall -g @brightside/core
```

Note: You might receive an ENOENT error when issuing this command if you installed Zowe CLI from a local package (.tgz) and the package was moved from its original location. In the event that you receive the error, open an issue in the Zowe CLI GitHub repository.

4. Install the most recent @zowe-v1-lts version CLI and optional plug-ins. For more information, see [Installing Zowe CLI](#) on page 173.
5. **(Optional)** If you deleted your profiles in Step 1, recreate the profiles that you need manually.
6. **(Optional)** If you copied your profiles to a local directory in Step 1, follow these steps:
 - a. Move the profile configuration files that you saved in Step 1 back to the ~/.zowe/profiles or %homepath%\zowe\profiles folder on your computer.
 - b. Issue the zowe scs update command to update profiles that are secured with the Secure Credential Store Plug-in.
 - c. Issue the command zowe profiles update zosmf <my-profile-name> --user <my-username> --password <my-password> to update z/osmf profiles to use the current option names.

You updated to the Zowe CLI LTS version!

Ensure that you review the [Release notes](#) on page 20, which describes **Notable Changes** in this version. We recommend issuing familiar commands and running scripts to ensure that your profiles/scripts are compatible. You might need to take corrective action to address the breaking changes.

Identify the currently installed version of Zowe CLI

Issue the following command:

```
zowe -V
```

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 previously installed the CLI as described in [Installing Zowe CLI from an online registry](#) on page 174.

Follow these steps:

1. To update Zowe CLI to the most recent @zowe-v1-lts version, issue the following command:

```
npm install -g @zowe/cli@zowe-v1-lts
```

2. To update existing plug-ins and install new plug-ins, issue the following command:

```
zowe plugins install @zowe/cics-for-zowe-cli@zowe-v1-lts @zowe/db2-for-zowe-cli@zowe-v1-lts @zowe/ims-for-zowe-cli@zowe-v1-lts @zowe/mq-for-zowe-cli@zowe-v1-lts @zowe/zos-ftp-for-zowe-cli@zowe-v1-lts @zowe/secure-credential-store-for-zowe-cli@zowe-v1-lts
```

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 6.1.2:

```
npm install -g @zowe/cli@6.1.2
```

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 from a Local package instructions. For more information, see [Uninstalling Zowe CLI](#) on page 178 and [Installing Zowe CLI from a local package](#) on page 173.

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 given command group. Issue 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 the following command:

```
npm uninstall --global @zowe/cli
```

Note: You might receive an ENOENT error when issuing this command if you installed Zowe CLI from a local package (.tgz) and the package was moved from its original location. In the event that you receive the error, open an issue in the Zowe CLI GitHub repository.

The uninstall process removes all Zowe CLI installation directories and files from your computer.

6. Delete the `~/ .zowe` or `%homepath%\ .zowe` 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.

Advanced Zowe configuration

Configuring Zowe Application Framework

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 `$RUNTIME_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 `instance.env` file in the instance directory, see [Creating and configuring the Zowe instance directory](#) on page 150.

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 `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.

Setting up the TN3270 mainframe terminal application plug-in

`_defaultTN3270.json` is a file in `tn3270-ng2/`, which is deployed during setup. Within this file, you can specify the following parameters to configure the terminal connection:

```
"host": <hostname>
"port": <port>
"security": {
  type: <"telnet" or "tls">
}
```

Setting up the VT Terminal application plug-in

`_defaultVT.json` is a file in `vt-ng2/`, which is deployed during setup. Within this file, you can specify the following parameters to configure the terminal connection:

```
"host": <hostname>
"port": <port>
"security": {
  type: <"telnet" or "ssh">
}
```

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 `$INSTANCE_DIR/workspace/app-server/serverConfig/server.json`. The defaults from which that file is generated are located at `$RUNTIME_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 hierarchy 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 | <code>zlux-app-server/defaults</code> | <code>zlux-app-server/deploy/product</code> | Official installs place <code>zlux-app-server</code> within <code><RUNTIME_DIR>/components/app-server/share</code> |
| siteDir | <code><INSTANCE_DIR>/workspace/app-server/site</code> | <code>zlux-app-server/deploy/site</code> | <code>INSTANCE_DIR</code> is <code>~/.zowe</code> if not otherwise defined. Site is placed within instance due to lack of <code>SITE_DIR</code> as of 1.8 |
| instanceDir | <code><INSTANCE_DIR>/workspace/app-server</code> | <code>zlux-app-server/deploy/instance</code> | |
| groupsDir | <code><INSTANCE_DIR>/workspace/app-server/groups</code> | <code>zlux-app-server/deploy/instance/groups</code> | |
| usersDir | <code><INSTANCE_DIR>/workspace/app-server/users</code> | <code>zlux-app-server/deploy/instance/users</code> | |
| pluginsDir | <code><INSTANCE_DIR>/workspace/app-server/plugins</code> | <code>zlux-app-server/deploy/instance/ZLUX/plugins</code> | Defaults located at <code>zlux-app-server/defaults/plugins</code> , previously at <code>zlux-app-server/plugins</code> |

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 390.

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 communication, you can use Application Transparent Transport Layer Security (AT-TLS) to enable Hyper Text Transfer Protocol Secure (HTTPS) communication with ZSS.

Before you begin, you must have a basic knowledge of your security product, e.g. 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](#).

You must have the authority to alter security definitions related to certificate management, and you must be authorized to work with and update the Policy Agent.

To configure HTTPS communication between ZSS and the Zowe App Server, you need a key ring which contains the ZSS server certificate and its Certificate Authority (CA) certificate. You can use an internal CA to create the ZSS server certificate, or you can buy the ZSS server certificate from a well-known commercial Certificate Authority. Next you define an AT-TLS rule which points to the key ring used by the ZSS server. Then you copy the CA certificate to the Zowe App Server key store and update the Zowe App Server configuration file.

Note: Bracketed values below (including the brackets) are variables. Replace them with values relevant to your organization. Always use the same value when substituting a variable that occurs multiple times.

Creating certificates and key ring for the ZSS server using RACF

In this step you will create a root CA certificate and a ZSS server certificate signed by the CA certificate. Next you create a key ring owned by the ZSS server with the certificates attached.

Key variables:

| Variable | Value |
|-----------------------|-------|
| [ca_common_name] | |
| [ca_label] | |
| [server_userid] | |
| [server_common_name] | |
| [server_label] | |
| [ring_name] | |
| [output_dataset_name] | |

Note:

- [server_userid] must be the ZSS server user ID.
- [server_common_name] must be the ZSS server host name.

1. Enter the following RACF command to generate a CA certificate:

```
RACDCERT CERTAUTH GENCERT +
SUBJECTSDN(CN(''[ca_common_name]'') +
OU(''[organizational_unit]'') +
O(''[organization_name]'') +
L(''[locality]'') SP(''[state_or_province]'') C(''[country]'')) +
KEYUSAGE(CERTSIGN) +
WITHLABEL(''[ca_label]'') +
NOTAFTER(DATE(''yyyy/mm/dd'')) +
SIZE(2048)
```

1. Enter the follow RACF command to generate a server certificate signed by the CA certificate:

```
RACDCERT ID(''[server_userid]'') GENCERT +
SUBJECTSDN(CN(''[common_name]'') +
OU(''[organizational_unit]'') +
O(''[organization_name]'') +
L(''[locality]'') SP(''[state_or_province]'') C(''[country]'')) +
KEYUSAGE(HANDSHAKE) +
WITHLABEL(''[server_label]'') +
NOTAFTER(DATE(''yyyy/mm/dd'')) +
SIZE(2048) +
SIGNWITH(CERTAUTH LABEL(''[ca_label]''))
```

1. Enter the following RACF commands to create a key ring and connect the certificates to the key ring:

```
RACDCERT ID([server_userid]) ADDRING([ring_name])
RACDCERT ID([server_userid]) CONNECT(ID([server_userid]) +
LABEL(''[server_label]'') RING([ring_name]) DEFAULT)
RACDCERT ID([server_userid]) CONNECT(CERTAUTH +
LABEL(''[ca_label]'') RING([ring_name]))
```

1. Enter the following RACF command to refresh the DIGTRING and DIGTCERT classes to activate your changes:

```
SETROPTS RACLIST(DIGTRING,DIGTCERT) REFRESH
```

- Enter the following RACF commands to verify your changes:

```
RACDCERT ID([server_userid]) LISTRING([ring_name])
RACDCERT ID([server_userid]) LISTCHAIN(LABEL('['[server_label]]'))'
```

- Enter the following RACF commands to allow the ZSS server to use the certificates. Only issue the RDEFINE commands if the profiles do not yet exist.

```
RDEFINE FACILITY IRR.DIGTCERT.LIST UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ACCESS(READ) +
    ID([server_userid])
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ACCESS(READ) +
    ID([server_userid])
SETROPTS RACLIST(FACILITY) REFRESH
```

Note: These sample commands use the FACILTY class to manage certificate related authorizations. You can also use the RDATALIB class, which offers granular control over the authorizations.

- Enter the following RACF command to export the CA certificate to a dataset so it can be imported by the Zowe App Server:

```
RACDCERT CERTAUTH EXPORT(LABEL('['ca_label']')) +
    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:

| | |
|-------------------------------|------------|
| TTLRule | ATTLS1~ZSS |
| { | |
| LocalAddr | All |
| RemoteAddr | All |
| LocalPortRange | [zss_port] |
| Jobname | * |
| Userid | * |
| Direction | Inbound |
| Priority | 255 |
| TTLGroupActionRef | gAct1~ZSS |
| TTLSEnvironmentActionRef | eAct1~ZSS |
| TTLConnectionActionRef | cAct1~ZSS |
| } | |
| TTLGroupAction | gAct1~ZSS |
| { | |
| TTLSEnabled | On |
| Trace | 1 |
| } | |
| TTLSEnvironmentAction | eAct1~ZSS |
| { | |
| HandshakeRole | Server |
| EnvironmentUserInstance | 0 |
| TTLKeyringParmsRef | key~ZSS |
| Trace | 1 |
| } | |
| TTLConnectionAction | cAct1~ZSS |
| { | |
| HandshakeRole | Server |
| TTLSCipherParmsRef | cipherZSS |
| TTLConnectionAdvancedParmsRef | cAdv1~ZSS |
| Trace | 1 |
| } | |
| TTLConnectionAdvancedParms | cAdv1~ZSS |

```
{
  SSLv3                      Off
  TLSv1                      Off
  TLSv1.1                     Off
  TLSv1.2                     On
  CertificateLabel             [personal_label]
}
TTLSSKeyringParms
{
  Keyring                     [ring_name]
}
TTLSCipherParms
{
  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] '' '[INSTANCE_DIR]/workspace/app-server/
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.

- 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)

- Create a new Zowe instance directory by following steps in [Creating and configuring the Zowe instance directory](#) on page 150.

Note: In the `instance.env` configuration file, specify ports that are not used by the first Zowe runtime.

- To restart the first Zowe runtime, in SDSF enter the following command:

```
/S ZWESVSTC,INSTANCE='$INSTANCE_DIR'
```

Where `$INSTANCE_DIR` is the Zowe instance directory.

- To specify a name for the new ZSS instance, follow these steps:

- Copy the PROCLIB member JCL named ZWESISTC that was installed with the new runtime.
- Rename the copy to uniquely identify it as the JCL that starts the new ZSS, for example ZWESIS02.
- 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.

- To start the new ZSS, in SDSF enter the following command:

```
/S ZWESIS02
```

- 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 ZWESVUSR.
- 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`

- To start the new Zowe runtime, in SDSF enter the following command:

```
/S ZWESVSTC,INSTANCE='$ZOWE_INSTANCE_DIR'
```

- 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 190 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. [Restart the app server](#).

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. [Restart the app server.](#)

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 375 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
```

- 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:

- 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`.
- 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>.<uri>
```

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 application developers maintain relatively short endpoint URL paths.

For information on endpoint URLs, see [Dataservice endpoint URL lengths and RBAC](#)

Multi-factor authentication configuration

[Multi-factor authentication](#) is an optional feature for Zowe.

As of Zowe version 1.8.0, the Zowe App Framework, Desktop, and all apps present in the SMP/E or convenience builds support [out-of-band MFA](#) by entering an MFA assigned token or passcode into password field of the Desktop login screen, or by accessing the app-server /auth REST API endpoint.

For a list of compatible MFA products, see [Known compatible MFA products](#)

Session duration and expiration

After successful authentication, a Zowe Desktop session is created by authentication plugins.

The duration of the session is determined by the plugin used. Some plugins are capable of renewing the session prior to expiration, while others may have a fixed session length.

Zowe is bundled with a few of these plugins:

- **apiml-auth:** Calls the Zowe API Mediation Layer from the app-server for authentication. By default, the Mediation Layer calls z/OSMF to answer the authentication request. The session created mirrors the z/OSMF session.
- **zosmf-auth:** Calls z/OSMF auth from the app-server to answer the authentication request. The created z/OSMF session is valid for about 8 hours.
- **zss-auth:** Calls Zowe ZSS from the app-server to answer the authentication request. The created ZSS session is valid for 1 hour, but is renewable on request prior to expiration. In the Desktop, the session is automatically renewed if the user is detected as active. If the user is detected as idle, the session will expire.

When a session expires, the credentials used for the initial login are likely to be invalid for re-use, since MFA credentials are often one-time-use or time-based.

In the Desktop, Apps that you opened prior to expiration will remain open so that your work can resume after entering new credentials.

Configuration

When you use the default Zowe SMP/E or convenience build configuration, you do not need to change Zowe to get started with MFA.

To configure Zowe for MFA with a configuration other than the default, take the following steps:

1. Choose an App Server security plugin that is compatible with MFA. The [Session duration and expiration](#) on page 191 plugins are all compatible.
2. Locate the App Server's configuration file in `$INSTANCE_DIR/workspace/app-server/serverConfig/server.json`
3. Edit the configuration file to modify the section `dataserviceAuthentication`.
4. Set `defaultAuthentication` to the same category as the plugin of choice, for example:
 - **apiml-auth:** "apiml"
 - **zosmf-auth:** "zosmf"
 - **zss-auth:** "zss"

- Define the plugins to use in the configuration file by adding a section for the chosen category within `dataserviceAuthentication.implementationDefaults` as an object with the attribute `plugins`, which is an array of plugin ID strings, where the plugins each have the following IDs:

- apiml-auth**: "org.zowe.zlux.auth.apiml"
- zosmf-auth**: "org.zowe.zlux.auth.zosmf"
- zss-auth**: "org.zowe.zlux.auth.zss"

The following is an example configuration for `zss-auth`, as seen in a default installation of Zowe:

```
"dataserviceAuthentication": {
  "defaultAuthentication": "zss",
  "implementationDefaults": {
    "zss": {
      "plugins": [
        "org.zowe.zlux.auth.zss"
      ]
    }
  }
}
```

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 <code>zluxserver.json</code> 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 <code>zluxserver.json</code> 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 <code>zluxserver.json</code> 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 <RUNTIME_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 configure Zowe CLI, such as changing log levels and setting the home directory location.

Tip: CLI configuration is stored on your computer in the directory C:\Users\user01\.zowe by default. The directory includes log files, profile information, and installed CLI plug-ins. When troubleshooting, refer to the logs in the `imperative` and `zowe` folders.

- [Setting CLI log levels](#) on page 195
- [Setting the CLI home directory](#) on page 195

Setting 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 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 |

Configuring the Zowe APIs

Review the security considerations for Zowe APIs and learn how to prevent the Denial of Service (DoS) attacks.

The default configuration before Zowe version 1.14.0 contains **Data sets and Unix files** and **Jobs** API microservices which might be vulnerable to DoS attacks in the form of slow https attacks. You can add additional configuration to the start script of these components in order to prevent resource starvation via slow https attacks.

- To update the configuration of the **Data sets and Unix files** component, modify the `start.sh` script within the runtime component directory /zowe/runtime/components/files-api/bin.

- To update the configuration of the **Jobs** component, modify the `start.sh` script within the runtime component directory `/zowe/runtime/components/jobs-api/bin`.

Ensure that the `-Dserver.connection-timeout=8000` parameter is set. This parameter specifies how long the component waits to receive all the required information from the client that makes a request.

See a snippet of a configured `start.sh` script for the Jobs component as follows:

```
_BPX_JOBNAME=${ZOWE_PREFIX}${COMPONENT_CODE} java -Xms16m -Xmx512m -
Dibm.serversocket.recover=true -Dfile.encoding=UTF-8 \
-Djava.io.tmpdir=/tmp -Xquickstart \
-Dserver.port=${JOBS_API_PORT} \
-Dcom.ibm.jsse2.overrideDefaultTLS=true \
-Dserver.ssl.keyAlias=${KEY_ALIAS} \
-Dserver.ssl.keyStore=${KEYSTORE} \
-Dserver.ssl.keyStorePassword=${KEYSTORE_PASSWORD} \
-Dserver.ssl.keyStoreType=${KEYSTORE_TYPE} \
-Dserver.compression.enabled=true \
-Dserver.connection-timeout=8000 \
-Dconnection.httpsPort=${GATEWAY_PORT} \
-Dconnection.ipAddress=${ZOWE_EXPLORER_HOST} \
-Dspring.main.banner-mode=off \
-Djava.protocol.handler.pkgs=com.ibm.crypto.provider \
-jar ${ROOT_DIR}/components/jobs-api/bin/jobs-api-server-1.0.0-boot.jar
&
```

In version 1.14.0 and later, the preceding snippet reflects the default configuration.

Advanced Gateway features configuration

As a system programmer who wants to configure advanced Gateway features of the API Mediation Layer, you can customize Gateway parameters by modifying either of the following files:

- `<Zowe install directory>/components/gateway/bin/start-gateway.sh`
- `<Zowe instance directory>/instance.env`

The parameters begin with the `-D` prefix, similar to all the other parameters in the file.

Note: Restart Zowe to apply changes to the parameter.

Follow the procedures in the following sections to customize Gateway parameters according to your preferences:

- [Prefer IP Address for API Layer services](#) on page 196
- [SAF as an Authentication provider](#) on page 197
- [Gateway retry policy](#) on page 197
- [Gateway client certificate authentication](#) on page 197
- [Gateway timeouts](#) on page 198
- [CORS handling](#) on page 199
- [Encoded slashes](#) on page 199
- [Connection limits](#) on page 199
- [API Mediation Layer as a standalone component](#) on page 199

Prefer IP Address for API Layer services

API Mediation Layer services use the hostname when communicating with each other. This behavior can be changed so that the IP address is used instead.

Follow these steps:

- Open the `<Zowe instance directory>/instance.env` configuration file.
- Find the property `APIML_PREFER_IP_ADDRESS` and set the value to `true`.
- Restart Zowe.

Note: Changing the value of this property might introduce problems with certificates. Ensure that the IP Address is present on the certificate SAN name.

SAF as an Authentication provider

By default, the API Gateway uses z/OSMF as an authentication provider. It is possible to switch to SAF as the authentication provider instead of z/OSMF. The intended usage of SAF as an authentication provider is for systems without z/OSMF. If SAF is used and the z/OSMF is available on the system, the created tokens are not accepted by z/OSMF. Use the following procedure to switch to SAF.

Follow these steps:

1. Open the <Zowe instance directory>/instance.env configuration file.
2. Find the property APIML_SECURITY_AUTH_PROVIDER and set the value to saf.
3. Restart Zowe&trade.

Authentication requests now utilize SAF as the authentication provider. API ML can run without z/OSMF present on the system.

Gateway retry policy

To change the Gateway retry policy, edit properties in the <Zowe install directory>/components/gateway/bin/start.sh file:

All requests are disabled as the default configuration for retry with one exception: the server retries GET requests that finish with status code 503. To change this default configuration, include the following parameters:

- **ribbon.retryableStatusCodes**

Provides a list of status codes, for which the server should retry the request.

Example: -Dribbon.retryableStatusCodes="503, 404"

- **ribbon.OkToRetryOnAllOperations**

Specifies whether to retry all operations for this service. The default value is false. In this case, only GET requests are retried if they return a response code that is listed in ribbon.retryableStatusCodes. Setting this parameter to true enables retry requests for all methods which return a response code listed in ribbon.retryableStatusCodes.

Note: Enabling retry can impact server resources due to request body buffering.

- **ribbon.MaxAutoRetries**

Specifies the number of times a failed request is retried on the same server. This number is multiplied with ribbon.MaxAutoRetriesNextServer. The default value is 0.

- **ribbon.MaxAutoRetriesNextServer**

Specifies the number of additional servers that attempt to make the request. This number excludes the first server. The default value is 5.

Gateway client certificate authentication

Note:

Beginning with release 1.19 LTS, it is possible to authenticate using client certificates. The feature is functional and tested, but automated testing on various security systems is not complete. As such, the feature is provided as a beta release for early preview. If you would like to offer feedback using client certificate authentication, please create an issue against the api-layer repository. Client Certificate authentication will move out of Beta once test automation is fully implemented across different security systems.

Use the following procedure to enable the feature of using a client certificate as a method of authentication for the API Mediation Layer Gateway.

Follow these steps:

1. Open the <Zowe instance directory>/instance.env configuration file.

2. Configure the following properties:

- **APIML_SECURITY_X509_ENABLED**

This property is the global feature toggle. Set the value to `true` to enable client certificate functionality.

- **APIML_SECURITY_ZOSMF_APPLID**

When z/OSMF is used as an authentication provider, provide a valid APPLID to allow for client certificate authentication. The API ML generates a passticket for the specified APPLID and subsequently uses this passticket to authenticate to z/OSMF. The default value in the installation of z/OSMF is `I2UDFLT`.

Note: The following steps are only required if the ZSS hostname or default Zowe user name are altered:

1. Open the file `<Zowe install directory>/components/gateway/bin/start.sh`.
2. Configure the following properties:

- **apiml.security.x509.externalMapperUrl**

The API Mediation Gateway uses an external API to map a certificate to the owner in SAF. This property informs the Gateway about the location of this API. ZSS is the API provider in Zowe. Provide the ZSS URL in the following format:

```
-Dapiml.security.x509.externalMapperUrl=http://localhost:<ZSS-PORT>/
certificate/x509/map
```

The default port is 8542. The hostname is `localhost` as the ZSS server is accessible only locally.

- **apiml.security.x509.externalMapperUser**

To authenticate to the mapping API, a JWT token is sent with the request. The token represents the user that is configured with this property. The user authorization is required to use the `IRR.RUSERMAP` resource within the `FACILITY` class. The default value is `ZWESVUSR`. Permissions are set up during installation with the `ZWESECUR` JCL or workflow.

To customize the `ZWESECUR` JCL or workflow (`// SET ZWEUSER=ZWESVUSR * userid` for Zowe started task), change the `apiml.security.x509.externalMapperUser` to a new value.

Restart Zowe&trade.

Gateway timeouts

Use the following procedure to change the global timeout value for the API Mediation Layer instance.

Follow these steps:

1. Open the file `<Zowe instance directory>/instance.env`.
2. Find the property `APIML_GATEWAY_TIMEOUT_MILLIS`, and set the value to the desired value.
3. Restart Zowe&trade.

If you require finer control, you can edit the `<Zowe install directory>/components/gateway/bin/start.sh`, and modify the following properties:

- **apiml.gateway.timeoutMillis**

This property defines the global value for http/ws client timeout.

Add the following properties to the file for the API Gateway:

Note: Ribbon configures the client that connects to the routed services.

- **ribbon.connectTimeout**

Specifies the value in milliseconds which corresponds to the period in which API ML should establish a single, non-managed connection with the service. If omitted, the default value specified in the API ML Gateway service configuration is used.

- **ribbon.readTimeout**

Specifies the time in milliseconds of inactivity between two packets in response from this service to API ML. If omitted, the default value specified in the API ML Gateway service configuration is used.

- **ribbon.connectionManagerTimeout**

The HttpClient employs a special entity to manage access to HTTP connections called by the HTTP connection manager. The purpose of an HTTP connection manager is to serve as a factory for new HTTP connections, to manage the life cycle of persistent connections, and to synchronize access to persistent connections. Internally, the connections that are managed serve as proxies for real connections. ConnectionManagerTimeout specifies a period during which managed connections with API ML should be established. The value is in milliseconds. If omitted, the default value specified in the API ML Gateway service configuration is used.

CORS handling

By default, Cross-Origin Resource Sharing (CORS) is disabled in the API Gateway for the Gateway routes `api/v1/gateway/**`. To enable CORS at the service level, it is necessary to enable CORS in the Gateway. Use the following procedure to enable CORS.

Follow these steps:

1. Open the file <Zowe instance directory>/instance.env.
2. Find the property `APIML_CORS_ENABLED` and set the value to `true`.
3. Restart Zowe&trade.

Requests through the Gateway now contain a CORS header.

Encoded slashes

By default, the API Mediation Layer accepts encoded slashes in the URL path of the request. If you are onboarding applications which expose endpoints that expect encoded slashes, it is necessary to keep the default configuration. We recommend that you change the property to `false` if you do not expect the applications to use the encoded slashes.

Use the following procedure to reject encoded slashes.

Follow these steps:

1. Open the file <Zowe instance directory>/instance.env.
2. Find the property `APIML_ALLOW_ENCODED_SLASHES` and set the value to `false`.
3. Restart Zowe&trade.

Requests with encoded slashes are now rejected by the API Mediation Layer.

Connection limits

By default, the API Gateway accepts up to 100 concurrent connections per route, and 1000 total concurrent connections. Any further concurrent requests are queued until the completion of an existing request. The API Gateway is built on top of Apache HTTP components that require these two connection limits for concurrent requests. For more information, see [Apache documentation](#).

Use the following procedure to change the number of concurrent connections.

Follow these steps:

1. Open the file <Zowe instance directory>/instance.env.
2. Find the property `APIML_MAX_CONNECTIONS_PER_ROUTE` and set the value to an appropriate positive integer.
3. Find the property `APIML_MAX_TOTAL_CONNECTIONS` and set the value to an appropriate positive integer.

API Mediation Layer as a standalone component

As a Zowe user, follow the procedure in this article to start the API Mediation Layer independently of other Zowe components. By default, the Gateway, Zowe System Services, and Virtual Desktop start when Zowe runs. To limit

consumed resources when the Virtual Desktop or Zowe System Services are not required, it is possible to specify which components start in the context of Zowe. No change is required during the installation process to support this setup.

Once Zowe is installed, use the following procedure to limit which components start.

Follow these steps:

1. Open the file <Zowe instance directory>/instance.env.
2. Find the property ZWE_LAUNCH_COMPONENTS and set discovery,gateway,api-catalog
3. Restart Zowe&trade.

To learn more about the related section of the environment file, see [Component groups](#) on page 151. We recommend you open this page in a new tab.

API Gateway configuration parameters

As an application developer who wants to change the default configuration of the API Mediation Layer, set the following parameters by modifying the <Zowe install directory>/components/gateway/bin/start.sh file:

- [Runtime configuration](#) on page 200
- [Service configuration](#) on page 202
- [Zuul configuration](#) on page 202
- [Hystrix configuration](#) on page 202

Runtime configuration

This section describes runtime configuration properties.

- **apimpl.service.hostname**

This property is used to set the API Gateway hostname.

- **apimpl.service.port**

This property is used to set the API Gateway port.

- **apimpl.service.discoveryServiceUrls**

This property specifies the Discovery Service URL used by the service to register to Eureka.

- **apimpl.service.preferIpAddress**

Set the value of this property to true to advertize a service IP address instead of its hostname.

Notes:

- If you set this property to true on the Discovery service, ensure that you modify the value of discoveryLocations: to use the IP address instead of the hostname. Failure to modify the discoveryLocations: value prevents Eureka from detecting registered services. As a result, the available-replicas is empty.
- Enabling this property may also cause issues with SSL certificates and Subject Alternative Name (SAN).
- **apimpl.cache.storage.location**

This property specifies the location of the EhCache used by Spring.

Note: It is necessary for the API ML process to have write access to the cache location.

- **apimpl.security.ssl.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 environments significantly degrades the overall security of the system.

- **apiml.security.auth.zosmfServiceId**

This parameter specifies the z/OSMF service id used as authentication provider. The service id is defined in the static definition of z/OSMF. The default value is `zosmf`.

- **apiml.zoweManifest**

This parameter lets you view the Zowe version by using the `/version` endpoint. To view the version requires setting up the launch parameter of the API Gateway - `apiml.zoweManifest` with a path to the Zowe build manifest.json file. This file is usually located in the root folder of Zowe build. If the encoding of manifest.json file is different from UTF-8 and IBM1047, it requires setting up the launch parameter of API Gateway - `apiml.zoweManifestEncoding` with correct encoding.

Note: It is also possible to know the version of API ML and Zowe (if API ML used as part of Zowe), using the `/api/v1/gateway/version` endpoint in the API Gateway service in the following format:

```
https://localhost:10010/api/v1/gateway/version
```

- **apiml.security.auth.tokenProperties.expirationInSeconds**

This property is relevant only when the JWT token is generated by the API Mediation Layer. API ML generation of the JWT token occurs in the following cases:

- z/OSMF is only available as an older version which does not support JWT tokens
- The SAF provider is used

To use a custom configuration for z/OSMF which changes the expiration of the LTPA token, it is necessary to also set the expiration in this parameter.

Note: The default value is 8 hours which mimicks the 8 hour default expiration of the LTPA token in z/OSMF.

Follow these steps:

1. Open the file <Zowe install directory>/components/gateway/bin/start.sh.
2. Find the line that contains `-cp ${ROOT_DIR}"/components/gateway/gateway-service.jar":/usr/include/java_classes/IRRacfc.jar`.
3. Before this line, add a new line in the following format:

```
-Dapiml.security.auth.tokenProperties.expirationInSeconds={expirationTimeInSeconds}
```

where:

- `{expirationTimeInSeconds}` refers to the specific time before expiration
- 1. Restart Zowe™
- **ibm.serversocket.recover**

In a multiple network stack environment (CINET), when one of the stacks fails, no notification or Java™ exception occurs for a Java program that is listening on an INADDR_ANY socket. When new stacks become available, the Java application does not become aware of them until the application rebinds the INADDR socket. By default, this parameter is enabled in the API Gateway. As a result, the NetworkRecycledException exception is thrown to the application to allow it to either fail or attempt to rebind. For more information, see the [IBM documentation](#).

- **java.io.tmpdir**

This property is a standard Java system property which is used by the disk-based storage policies. It determines where the JVM writes temporary files, including those written by these storage policies. The default value is typically `/tmp` on Unix-like platforms.

- **spring.profiles.include**

This property can be used to unconditionally add active profiles. For more information, see [Spring documentation](#).

- **server.maxTotalConnections and server.maxConnectionsPerRoute**

These two properties are used to set the number of concurrent connections. Any connection requests that are made that would put the number of connections over either of these limits are queued until an existing connection completes. The API Gateway is built on top of Apache HTTP components that require these two connection limits for concurrent requests. For more information, see [Apache documentation](#).

Service configuration

For information about service configuration parameters, see [Onboarding a REST API service with the Plain Java Enabler \(PJE\)](#).

Zuul configuration

As a provider for routing and filtering, the API Gateway contains a Zuul configuration as shown in the following example.

Example:

```
zuul:
  sslHostnameValidationEnabled: false
  addProxyHeaders: true
  traceRequestBody: true
  ignoreSecurityHeaders: false
  includeDebugHeader: false
  sensitiveHeaders: Expires,Date
  ignoredPatterns:
    - /ws/**
host:
  connectTimeoutMillis: ${apiml.gateway.timeoutMillis}
  socketTimeoutMillis: ${apiml.gateway.timeoutMillis}
  maxTotalConnections: ${server.maxConnectionsPerRoute}
  maxPerRouteConnections: ${server.maxTotalConnections}
  forceOriginalQueryStringEncoding: true
  retryable: true
  decodeUrl: false # Flag to indicate whether to decode the matched URL or
  use it as is
```

The Zuul configuration allows the API Gateway to act as a reverse proxy server through which API requests can be routed from clients on the northbound edge to z/OS servers on the southbound edge.

Note: For more information about Zuul configuration parameters, see the [Spring Cloud Netflix documentation](#).

Hystrix configuration

The API Gateway contains a Hystrix configuration as shown in the following example.

Example:

```
hystrix:
  command:
    default:
      fallback:
        enabled: false
      circuitBreaker:
        enabled: false
      execution:
        timeout:
          enabled: false
        isolation:
          thread:
            timeoutInMilliseconds:
              ${apiml.gateway.timeoutMillis}
            strategy: SEMAPHORE
```

```
semaphore:
    maxConcurrentRequests: ${server.maxTotalConnections}
```

Hystrix is a latency and fault tolerance library designed to isolate points of access to remote systems, services and third-party libraries, stop cascading failure, and enable resilience in complex distributed systems where failure is inevitable.

Note: For more information about Hystrix configuration parameters, see the [Netflix - Hystrix documentation](#).

Using Zowe

Getting started tutorial

Contents

- [Learning objectives](#) on page 203
- [Estimated time](#) on page 203
- [Prerequisites and assumptions](#) on page 204
- [Logging in to the Zowe Desktop](#) on page 204
- [Querying JES jobs and viewing related status in JES Explorer](#) on page 206
- [Using the 3270 Terminal in the Zowe Desktop to view the job](#) on page 208
- [Editing a data set in MVS Explorer](#) on page 218
- [Using the Zowe CLI to edit a data set](#) on page 219
- [Viewing the data set changes in MVS Explorer](#) on page 221
- [Next steps](#) on page 221
 - [Go deeper with Zowe](#) on page 221
 - [Try the Extending Zowe scenarios](#) on page 221
 - [Give feedback](#) on page 221

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 302
 - [Overview](#) on page 351
 - [Developing for Zowe CLI](#) on page 288

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 70.

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:
 - Using API mediation layer: `https://myhost:<gateway port>/ui/v1/zlux/`
 - Without the API mediation layer: `https://myhost:<appserver httpsPort>/`
 - 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.

There are two ways to log in to the Zowe Desktop:

- Through the API mediation layer: `https://myhost:<gateway port>/ui/v1/zlux/`
 - Example: `https://s0w1:7554/ui/v1/zlux/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html`
- Directly, if the mediation layer is not used: `https://myhost:<appserver httpsPort>/`
 - Example `https://s0w1:8544/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html`
- *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:

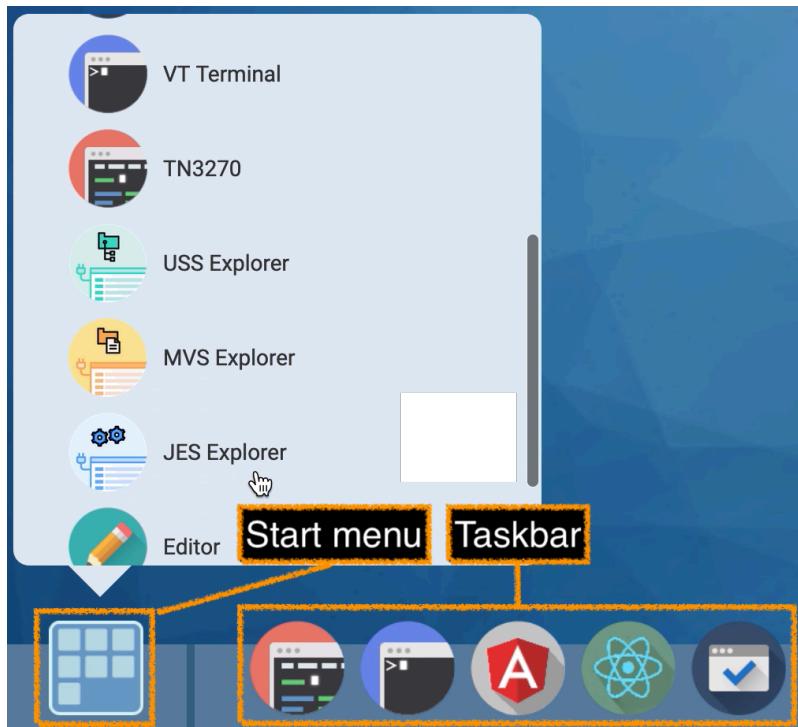
1. In the address field, enter the URL to access the Zowe Desktop. In this tutorial, the following URL is used as an example:

<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.

The screenshot shows a 'Job Filters' dialog box. At the top, there are two input fields: 'Owner= IBMUSER' and 'Prefix= *'. Below these, the 'Job Filters' section is expanded, showing three rows of filters:

| Owner | Prefix |
|---------|--------|
| IBMUSER | * |
| Job ID | Status |
| * | * |

At the bottom of the dialog are two buttons: 'APPLY' (dark blue) and 'RESET' (light blue).

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 (?).

The screenshot shows the same 'Job Filters' dialog box as before, but with different filter settings. The 'Owner' field now contains 'IBMUSER' (with the cursor inside). The 'Prefix' field contains 'SDSF*'. The 'Status' dropdown menu is open, showing 'ACTIVE' as the selected option. The other filter rows ('Job ID' and 'Status') remain at their defaults: '*' and '*' respectively.

- From the job filter results, click the job named **SDSF**. The data sets for this job are listed.



- 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.

```

1 //SDSF      JOB MSGLEVEL=1
2 //STARTING   JOB
3 XXDSF        The JOB Operation Field
4 XX           The first control statement. Marks the beginning of a job;
5 XX*          This assigns a name to the job.
6 XX*          The SDSFPARM DD Statement is optional. If it is not
7 XX*          present, SYS1.PARMLIB will be assumed.
8 XX*
9 XXDSF        CALL PGM=ISPFHELP,REGION=32M,TIME=1440,PARM=-M(EM) &
10 XX*
11 XX*          The SDSFPARM DD Statement is optional. If it is not
12 XX*          present, SYS1.PARMLIB will be assumed.

```

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:

- 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.

```
x - □ TN3270
z/OS V2R3 LVL1 PUT1803/RSU1803 IP Address = 127.0.0.1
VTAM Terminal = SC0TCP04

Application Developer System

      // 0000000  SSSSS
      // 00    00 SS
zzzzzz // 00    00 SS
      zz // 00    00 SSSS
      zz // 00    00   SS
zzzzzz // 00    00   SS
      // 0000000  SSSSS

System Customization - ADCD.Z23B.*


==> Enter "LOGON" followed by the TSO userid. Example "LOGON IBMUSER" or
==> Enter L followed by the APPLID
==> Examples: "L TSO", "L CICSTS53", "L CICSTS54", "L IMS14", "L IMS15"

MA A 24/1
```

A screenshot of the TN3270 terminal window. The title bar says 'TN3270'. The window content shows a z/OS V2R3 LVL1 session with the following text:
IP Address = 127.0.0.1
VTAM Terminal = SC0TCP04

Application Developer System

A series of binary digits (0000000, SSSSS, etc.) are displayed, likely representing a system customization or log entry.

System Customization - ADCD.Z23B.*

At the bottom, there are three command-line inputs:
==> Enter "LOGON" followed by the TSO userid. Example "LOGON IBMUSER" or
==> Enter L followed by the APPLID
==> Examples: "L TSO", "L CICSTS53", "L CICSTS54", "L IMS14", "L IMS15"

The bottom right corner shows '24/1'.

2. Enter the following command and press Enter to log on to TSO:

```
LOGON ibmuser
```

The terminal window title is 'TN3270'. The top status bar shows 'z/OS V2R3 LVLI PUT1803/RSUI803' on the left and 'IP Address = 127.0.0.1 VTAM Terminal = SC0TCP02' on the right. The main display area shows the following text:

```
Application Developer System
      // 0000000  SSSSS
      // 00      00 SS
zzzzzz // 00      00 SS
      zz // 00      00 SSSS
      zz // 00      00   SS
zzzzzz // 0000000  SSSS

System Customization - ADCD.Z238.*
```

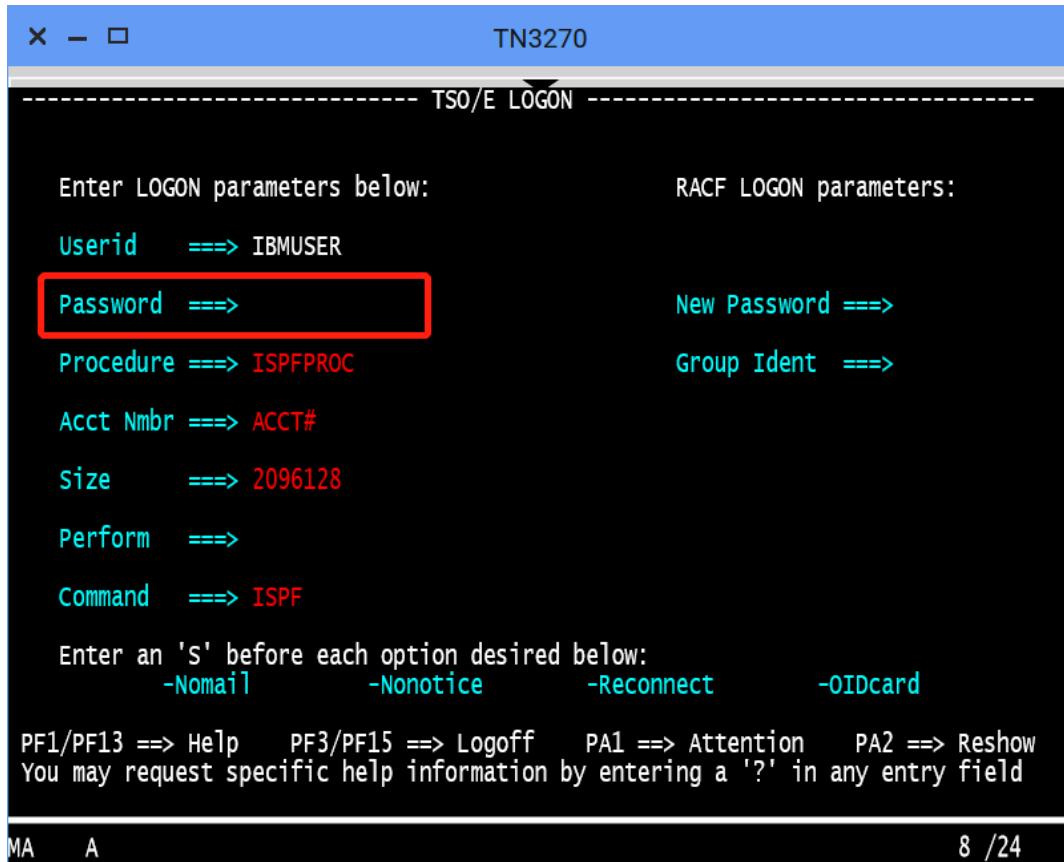
Below this, there are three lines of instructions:

```
==> Enter "LOGON" followed by the TSO userid. Example "LOGON IBMUSER" or
==> Enter L followed by the APPLID
==> Examples: "L TSO", "L CICSTS53", "L CICSTS54", "L IMS14", "L IMS15"
```

The command 'LOGON ibmuser' is highlighted in red at the bottom of the screen.

In the bottom left corner, there are two small icons: 'MA' and 'A'. In the bottom right corner, the text '24/14' is displayed.

3. On the TSO/E LOGON panel, enter the password sys1 in the **Password** field and press Enter.



You successfully log on to TSO.

- When you see the following screen, press Enter. The **ISPF Primary Option Menu** is displayed.

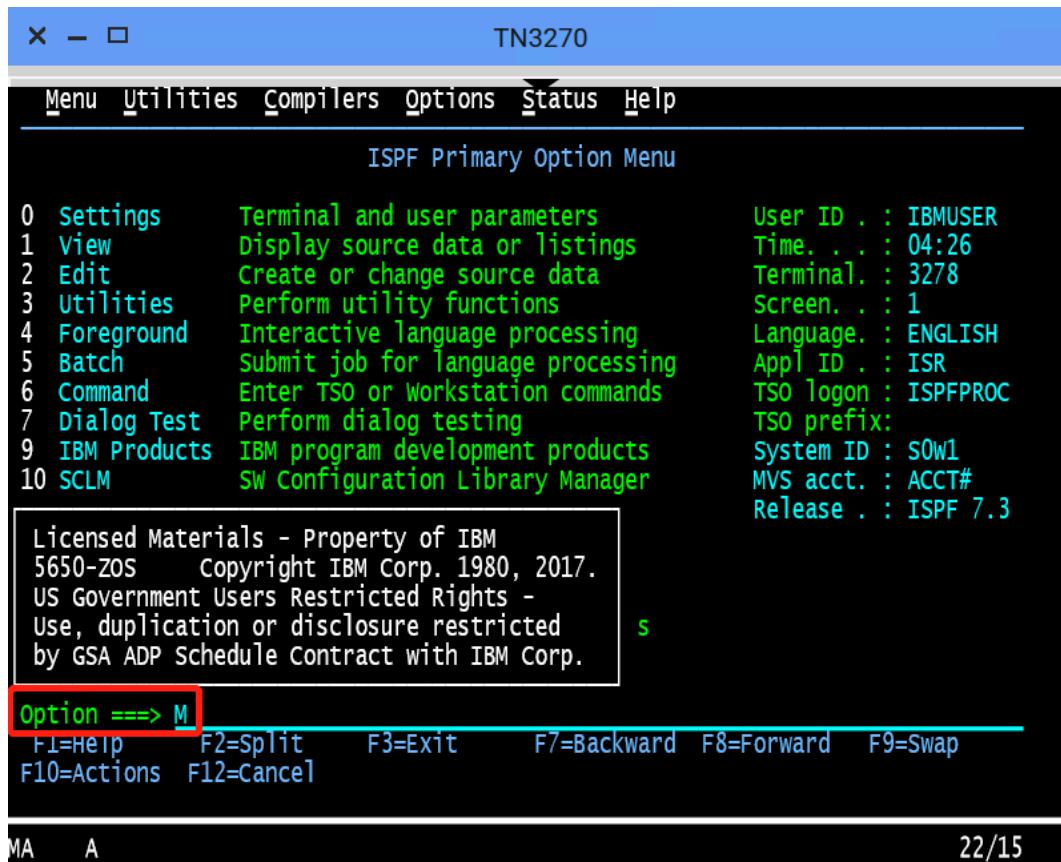
The screenshot shows a terminal window titled "TN3270". The window displays several system messages at the top, including logon information and broadcast message notices. Below these messages is a table showing user authentication details. The table has three columns: "USERID", "PASSWORD", and "COMMENT". The data in the table is as follows:

| USERID | PASSWORD | COMMENT |
|-------------------|----------------|-----------------------------|
| IBMUSER | - SYS1/IBMUSER | FULL AUTHORITY |
| ADCDMST | - ADCDMST | FULL AUTHORITY |
| ADCD A THRU ADCDZ | - TEST | LIMITED AUTHORITY(NO OMVS)* |
| OPEN1 THRU OPEN3 | - SYS1 | UID(0) (NO TSO) |

At the bottom of the window, the text "ISPF ***" is visible. The bottom status bar shows the characters "MA" and "A" on the left, and the page number "23/10" on the right.

5. Access SDSF to view output from a job. To do this,

- a. Type M at the **Option** prompt and press Enter.



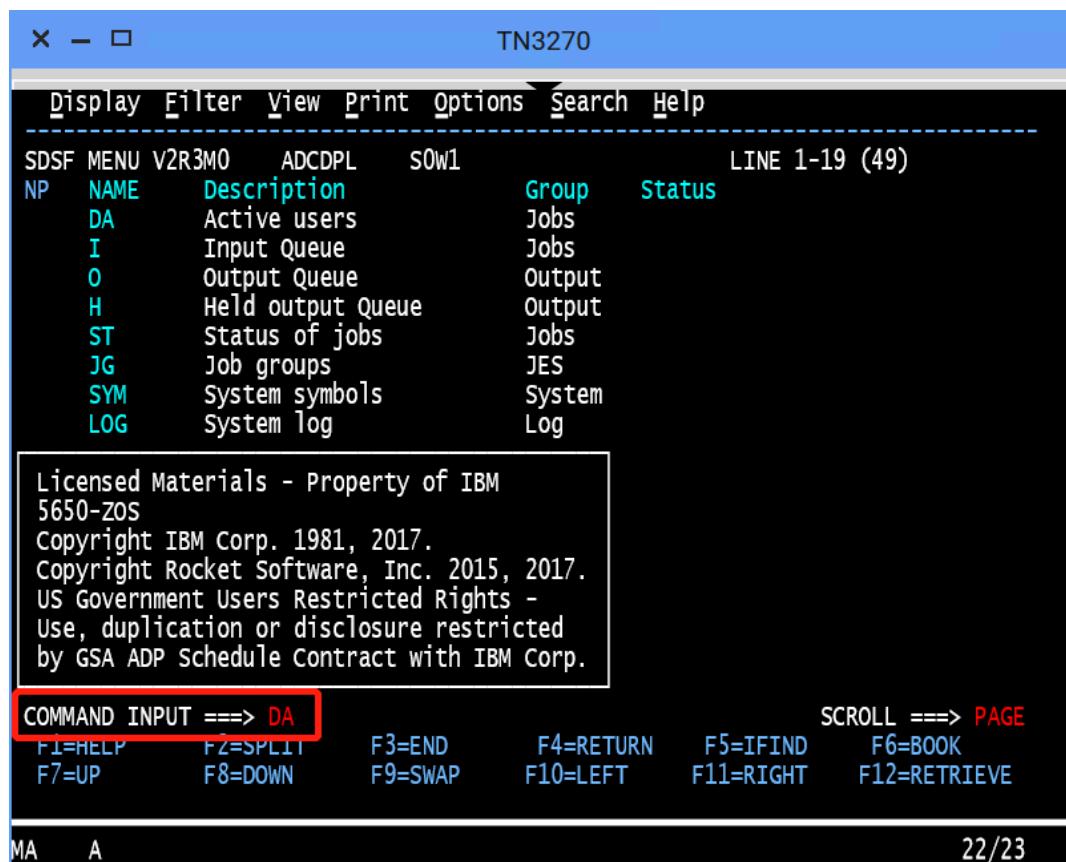
- b. Type 5 and press Enter.



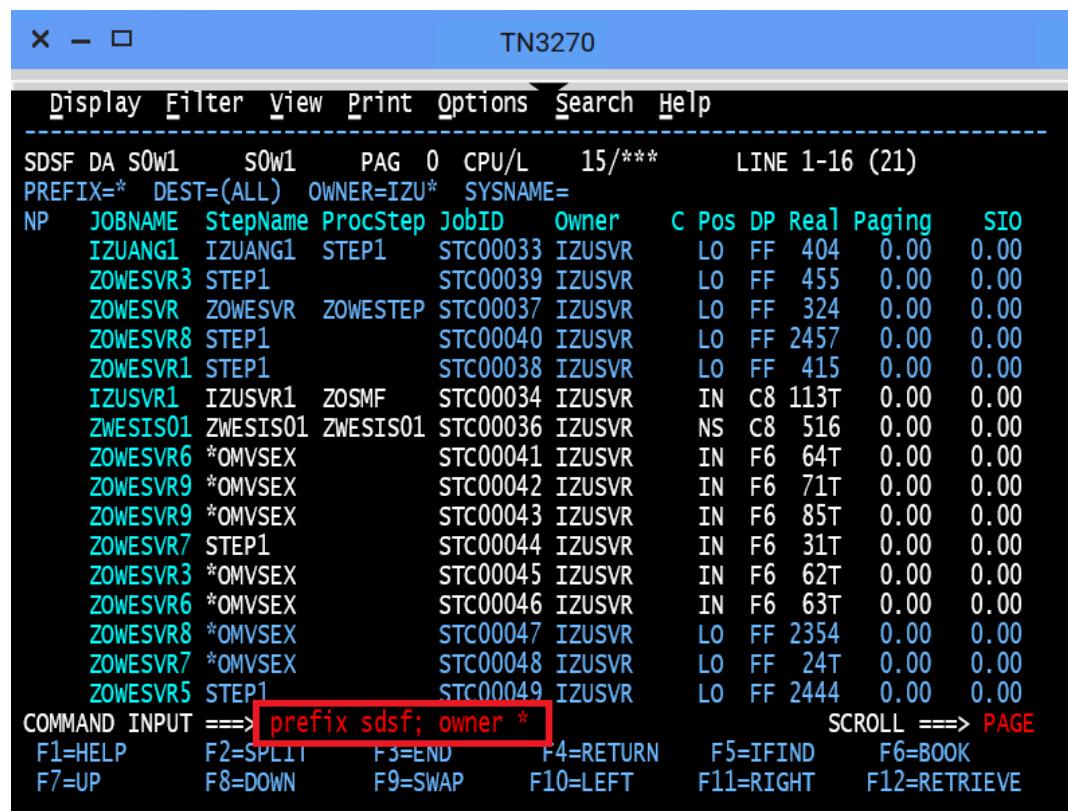
MA A

22/15

- 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.



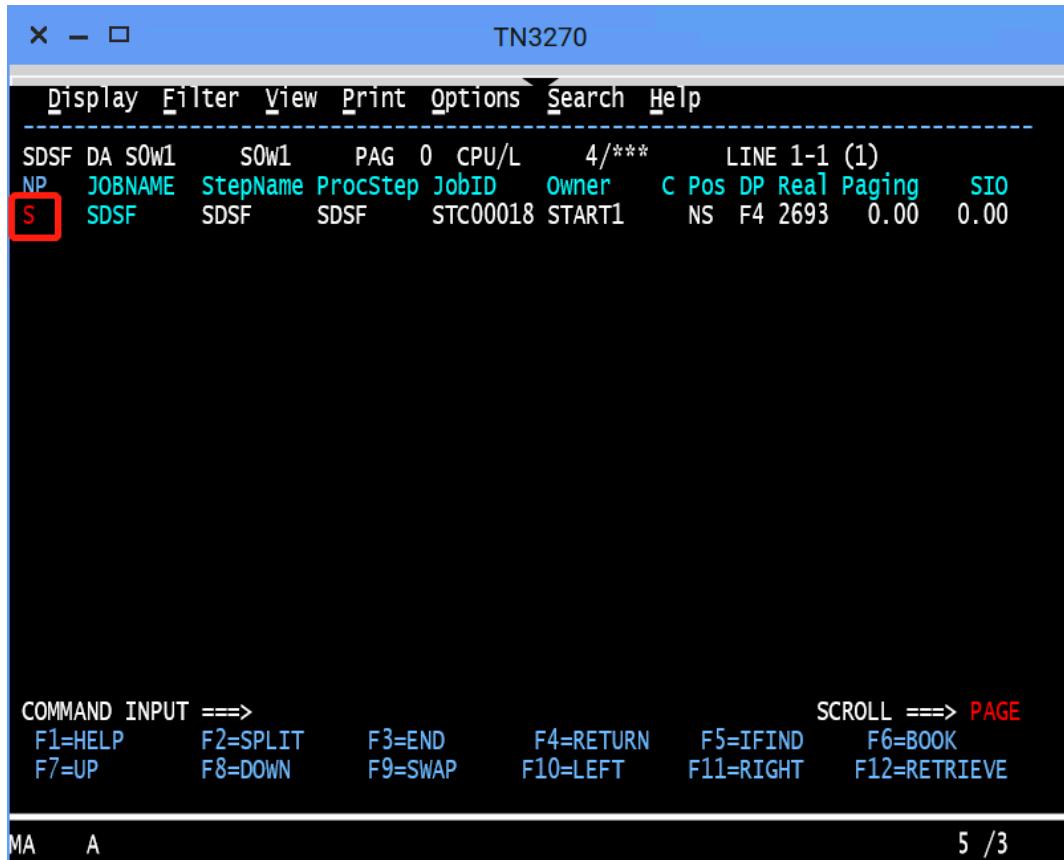
The screenshot shows a TN3270 terminal window titled "TN3270". The menu bar includes "Display", "Filter", "View", "Print", "Options", "Search", and "Help". The main area displays a list of jobs. The command input field at the bottom left contains "COMMAND INPUT ==> prefix sdsf; owner *". The scroll and page controls at the bottom right are labeled "SCROLL ==> PAGE".

| NP | JOBNAME | StepName | ProcStep | JobID | Owner | C | Pos | DP | Real | Paging | SIO |
|----|----------|----------|----------|----------|--------|----|-----|------|------|--------|-----|
| | IZUANG1 | IZUANG1 | STEP1 | STC00033 | IZUSVR | LO | FF | 404 | 0.00 | 0.00 | |
| | ZOWESVR3 | STEP1 | | STC00039 | IZUSVR | LO | FF | 455 | 0.00 | 0.00 | |
| | ZOWESVR | ZOWESTEP | | STC00037 | IZUSVR | LO | FF | 324 | 0.00 | 0.00 | |
| | ZOWESVR8 | STEP1 | | STC00040 | IZUSVR | LO | FF | 2457 | 0.00 | 0.00 | |
| | ZOWESVR1 | STEP1 | | STC00038 | IZUSVR | LO | FF | 415 | 0.00 | 0.00 | |
| | IZUSVR1 | IZUSVR1 | ZOSMF | STC00034 | IZUSVR | IN | C8 | 113T | 0.00 | 0.00 | |
| | ZWESIS01 | ZWESIS01 | ZWESIS01 | STC00036 | IZUSVR | NS | C8 | 516 | 0.00 | 0.00 | |
| | ZOWESVR6 | *OMVSEX | | STC00041 | IZUSVR | IN | F6 | 64T | 0.00 | 0.00 | |
| | ZOWESVR9 | *OMVSEX | | STC00042 | IZUSVR | IN | F6 | 71T | 0.00 | 0.00 | |
| | ZOWESVR9 | *OMVSEX | | STC00043 | IZUSVR | IN | F6 | 85T | 0.00 | 0.00 | |
| | ZOWESVR7 | STEP1 | | STC00044 | IZUSVR | IN | F6 | 31T | 0.00 | 0.00 | |
| | ZOWESVR3 | *OMVSEX | | STC00045 | IZUSVR | IN | F6 | 62T | 0.00 | 0.00 | |
| | ZOWESVR6 | *OMVSEX | | STC00046 | IZUSVR | IN | F6 | 63T | 0.00 | 0.00 | |
| | ZOWESVR8 | *OMVSEX | | STC00047 | IZUSVR | LO | FF | 2354 | 0.00 | 0.00 | |
| | ZOWESVR7 | *OMVSEX | | STC00048 | IZUSVR | LO | FF | 24T | 0.00 | 0.00 | |
| | ZOWESVR5 | STEP1 | | STC00049 | IZUSVR | LO | FF | 2444 | 0.00 | 0.00 | |

COMMAND INPUT ==> **prefix sdsf; owner *** SCROLL ==> PAGE

F1=HELP F2=SPL11 F3=END F4=RETURN F5=IFIND F6=BOOK
 F7=UP F8=DOWN F9=SWAP F10=LEFT F11=RIGHT F12=RETRIEVE

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.

```

TN3270

Display Filter View Print Options Search Help

SDSF OUTPUT DISPLAY SDSF      STC00018 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 0 W 1   --   N O

07.22.31 STC00018 ---- MONDAY,    01 APR 2019 ----
07.22.31 STC00018 IEF695I START SDSF      WITH JOBNME SDSF      IS ASSIGNED TO U
07.22.31 STC00018 $HASP373 SDSF      STARTED
07.22.32 STC00018 ISF724I SDSF level HQX77B0 initialization complete for server
07.22.33 STC00018 IEE252I MEMBER ISFPRM00 FOUND IN ADCL.Z23B.PARMLIB
07.22.33 STC00018 ISF739I SDSF parameters being read from member ISFPRM00 of da
07.22.46 STC00018 ISF728I SDSF parameters have been activated.
07.22.47 STC00018 ISF450I Server SDSF starting SDSFAUX.

1 //SDSF      JOB MSGLEVEL=1
2 //STARTING EXEC SDSF
3 XXSDSF      PROC M=00,          /* Suffix for ISFPRMxx */
XX             P='LC(A)'        /* USE SySout class A for SDSFLOG */
XX*
XX*
XX*      This is a Sample procedure to Start the SDSF Server.
F1=HELP      F2=SPLIT      F3=END       F4=RETURN     F5=IFIND      F6=BOOK
F7=UP        F8=DOWN       F9=SWAP      F10=LEFT      F11=RIGHT     F12=RETRIEVE

MA      A

```

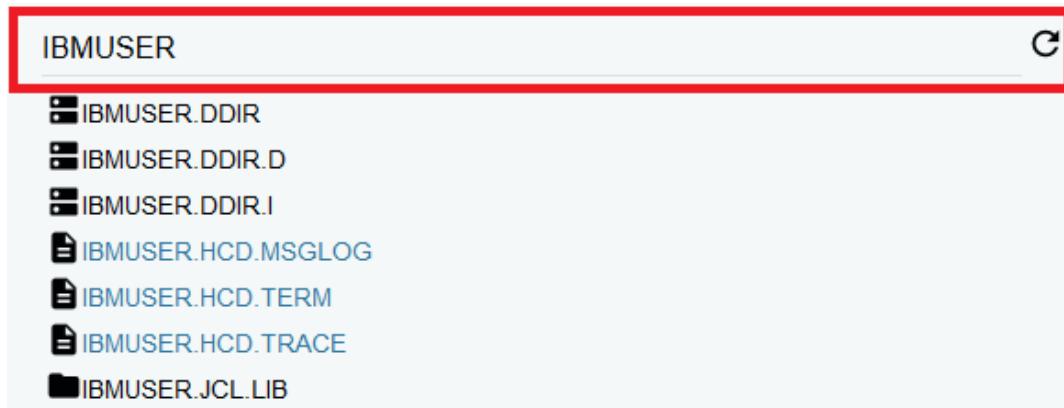
Close the 3270 Terminal window. In the next step, you will use the MVS Explorer to make changes to a data set.

Editing a data set in MVS Explorer

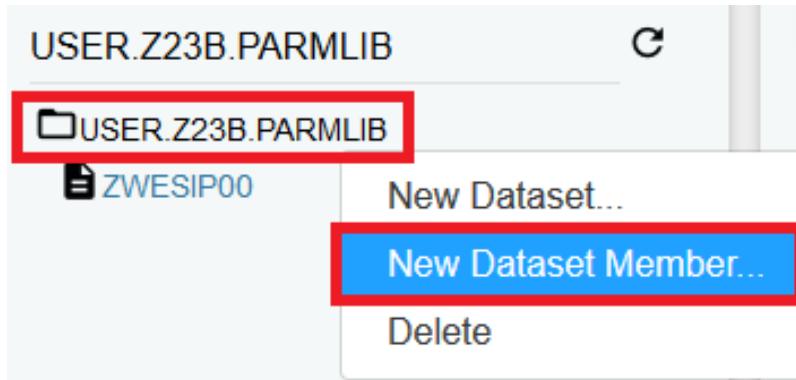
Use the MVS Explorer to create and edit a data set member and save the changes. The MVS Explorer view lets you browse the MVS file system by creating filters against data set names.

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.



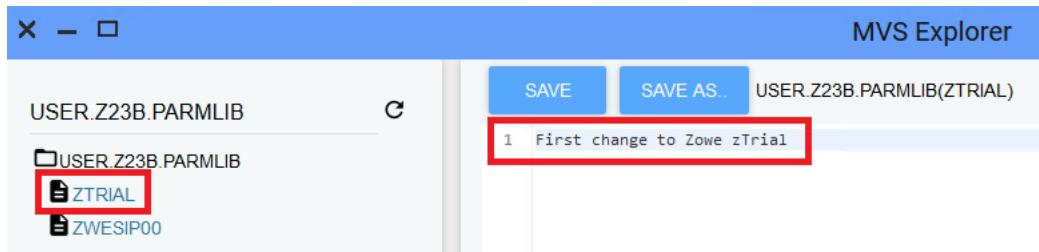
4. Enter USER.Z23B.PARMLIB in the **Filter** field to locate this data set and then click to expand it. Ensure that there is no extra space before the data set member name that you enter.
5. Right-click the USER.Z23B.PARMLIB data set and select **New Dataset Member**.



6. Enter **ZTRIAL** as the data set member name and click **OK** to create the data set member.

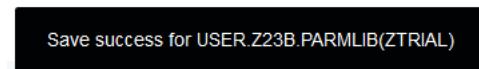


7. Click the data set member you just created and edit it by adding a new sentence, for example, First change to Zowe zTrial.



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.SMPLELIST
USER.Z23B.C1DTABL
USER.Z23B.CLIST
USER.Z23B.HELP
USER.Z23B.ISPMLIB
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.SVSEEXEC
USER.Z23B.TCPPARMS
USER.Z23B.UTAMLIB
USER.Z23B.UTAMLST
```

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 hierarchy 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
```

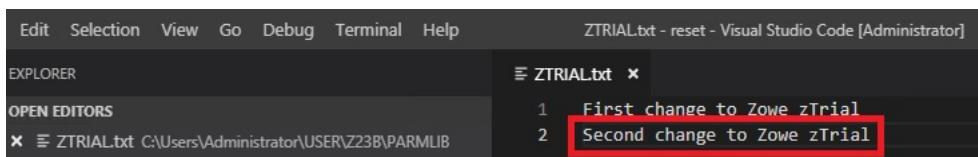
5. To open the data set member named ZTRIAL in Visual Studio Code, issue the following command against the same directory where you issued the download command:

```
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:

```
C:\Users\Administrator>zowe zos-files upload file-to-data-set USER/Z23B/PARMLIB/
ZTRIAL.txt "USER.Z23B.PARMLIB"
success: true
from: C:\Users\Administrator\USER\Z23B\PARMLIB\ZTRIAL.txt
to: USER.Z23B.PARMLIB(ZTRIAL)

file_to_upload: 1
success: 1
error: 0
skipped: 0

Data set uploaded successfully.
```

Congratulations! You used Zowe CLI to edit a data set member and upload the changes to mainframe.

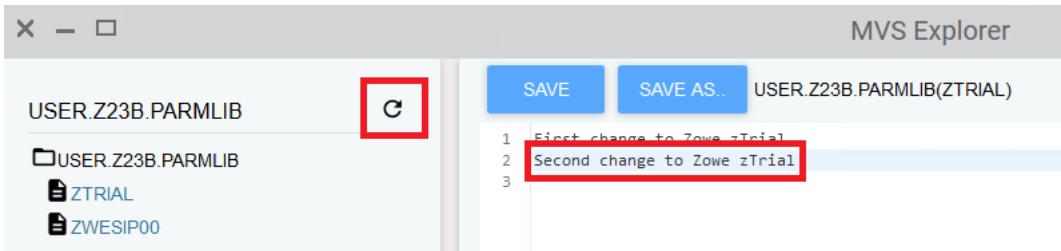
Close the Command Prompt window. In the next step, you will open the MVS Explorer again to view the updates that you made to the data set in this procedure.

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 222.

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 302 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 351.

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}}` or you can navigate to the direct Desktop URI at `https://{{myhost}}:{{httpsPort}}/ZLUX/plugins/org.zowe.zlux.bootstrap/web/`

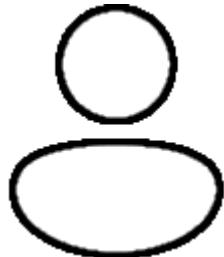
Where:

- *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 `server.json`. For example, if you run the Zowe Application Server on host *myhost* and the value that is assigned to `node.https.port` in `server.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 TSO 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 User icon in the lower right corner and click **Sign Out**.



Changing user password

1. Open the Preferences panel by clicking on the Preferences icon in the bottom right of the desktop.



1. Click the Change Password icon.
2. Fill out the Old Password and New Password fields.
3. Upon successful password change, you will be taken to the desktop.

Updating an expired password

1. Upon logging in with an expired password, a screen will be displayed prompting you to change your password.
2. Enter and confirm your new password in the corresponding fields.
3. Upon successful password change, you will be taken to the desktop.

Pinning applications to the task bar

1. Click the Start menu in the bottom left corner of the home screen.
2. Locate the application you want to pin.
3. Right-click the application icon and select **Pin to taskbar**.

Personalizing the Desktop

1. Click the **Preferences icon** to open the Preferences panel.



1. Click the **Personalization icon** to open the menu.



1. Drag an image into the wallpaper grid, or press the upload button, to upload a new Desktop wallpaper.
2. To set a new theme color, select a color from the palette or hue.
3. Use the lightness swatch bar to adjust the lightness of the color.
 - Adjusting the lightness will also change the lightness of secondary text.
1. Select a size (small, medium, or large) to adjust the scale of the Desktop UI.

Changing the desktop language

Use the Languages setting in the Preferences 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 Preferences 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 and using React.

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.

3270 Terminal

The 3270 Terminal plug-in provides a user interface that emulates the basic functions of IBM 3270 family terminals. On the "back end," the plug-in and the Zowe Application Server connect to any standard TN3270E server.

VT Terminal

The VT Terminal plug-in provides a user interface that emulates the basic functions of DEC VT family terminals. On the "back end," the plug-in and the Zowe Application Server connect to VT compatible hosts, such as z/OS UNIX System Services (USS), using standard network protocols.

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 and view datasets 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 Desktop 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 highlighting language

1. Click **Language** on the editor menu bar. A dropdown menu will be displayed.
2. From the dropdown, select the desired language. Plain Text will be chosen by default if the automatic language detection is not able to determine the language.

Open a dataset

To open a dataset, follow these steps:

1. From the **File** menu, select Open Datasets. You can also use (ALT+K).
2. In the Dataset field, specify the name of the dataset you want to open.
3. Click **Open**

Deleting a file or folder

1. In the file tree, right-click on a file or folder you want to delete.
2. From the right-click menu, click **Delete**. A warning dialogue will appear.
3. Click **Delete**

Opening a directory

1. From the **File** menu, select **Open Directory**. You can also use (ALT+O).
2. In the Directory field, specify 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. Double-clicking on a folder will make the active directory the newly specified folder.

Creating a new directory

1. Right-click on a location in the directory tree where you want to create a new directory.
2. From the right-click menu, click **Create a directory....**
3. Specify a directory name in the Directory Name field.
4. The Path will be set to the location that you initially right-clicked to open the dialogue. You can specify a different location in the Path field.
5. Click **Create**

Creating a new file

To create a new file, complete these steps:

1. From the **File** menu, select **New File**. You can also use (ALT+N).
2. From the **File** menu, select **Save** to save the newly created file. You can also use (Ctrl+S)
3. In the File Name field, specify the file name for the newly created file.
4. Choose an encoding option from the Encoding dropdown menu. The directory will be prefilled if you are creating the new file in an existing folder.
5. Click **Save**
6. To close a file, click the X icon in its tab, double-click on the tab, or use (Alt+W).

Hotkeys

The following hotkeys can be used in the editor to navigate or perform actions with only the keyboard.

- Shift TAB: Cycle through the menu bar, browsing type, search bar, file tree, and editor component.
- Individual options within the menu bar and individual nodes within the file tree can be navigated with the arrow keys and ENTER (to select).

| Hot Key | Command |
|------------------|------------------------------------|
| ALT+K | Open a dataset |
| ALT+O | Open a directory |
| ALT+N | Create a new file |
| ALT+W | Close tab |
| ALT+W+Shift | Close all tabs |
| CTRL+S | Save file |
| ALT+M | Navigate Menu bar (use arrow keys) |
| ALT+P | Search Bar focus |
| ALT+1 | Primary editing component focus |
| ALT+T+CTRL | Undo close/close all |
| ALT+R+Shift | Refresh active tab |
| ALT+PgUp(or <) | Switch to left tab |
| ALT+PgDown(or >) | Switch to right tab |
| ALT+B | Show/hide left-hand side file tree |

Using 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 `apimpl.catalog.tile` section defined in the `application.yml` of a service. A microservice that is onboarded to 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: [Add API Onboarding Configuration](#).

API Versioning

See [API Catalog and Versioning](#) for more information about the API versioning.

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:

- 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

- 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 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 are displayed.

Example:

API Catalog

< Back

Sample API Mediation Layer Applications

Applications which demonstrate how to make a service integrated to the API Mediation Layer ecosystem

discoverableclient sampleservice enablersv1sampleapp

Service Integration Enabler V2 Sample Application (Spring Boot 2.x)

API Doc Version: 1.0.0

[Base URL: <https://ca3x.ca.com:10010>]
[/api/v1/apicatalog/api/doc/discoverableclient/v1](#)

Sample service showing how to integrate a Spring Boot v2.x application

Other Operations General Operations

GET /ui/v1/discoverableclient/api/v1/instance/gateway-url What is the URI of the Gateway

- Select the version (**v1**, **v2**) to view the documentation of a specific API version.

Example:

discoverableclient

Service Spring Onboarding Enabler sample application API

Service Homepage

Instance URL: <https://localhost:10012/discoverableclient>

API Base Path: /discoverableclient/api/{api-version}

Service ID: discoverableclient

Sample API services to demonstrate Spring Onboarding Enabler

v1 v2

Service Spring Onboarding Enabler sample application API

API Version: 1.0.0

[Base URL: localhost:10010/discoverableclient/api/v1]

Sample API services to demonstrate Spring Onboarding Enabler

[Swagger/OpenAPI JSON Document](#)

[External documentation](#)

API Mediation Client test call

Api Mediation Client Test Controller

GET /apiMediationClient Indicate if registration with discovery service via API mediation client was successful

POST /apiMediationClient Forward registration to discovery service via API mediation client

DELETE /apiMediationClient Forward un-registration to discovery service via API mediation client

4. 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:

Service Integration Enabler V1 Sample App (spring boot 1.x)

API Doc Version: 1.0.0

[Base URL: <https://ca3x.ca.com:10010>]
</api/v1/apicatalog/apidoc/enablerv1sampleapp/v1>

Sample micro-service showing how to enable a Spring Boot v1.x application

The screenshot displays the API documentation for the 'V1EnablerSampleApp' sample controller. At the top, it shows the API Doc Version: 1.0.0 and the base URL: https://ca3x.ca.com:10010 /api/v1/apicatalog/apidoc/enablerv1sampleapp/v1. Below this, a brief description states: "Sample micro-service showing how to enable a Spring Boot v1.x application".

The main content area is titled 'V1EnablerSampleApp Sample Controller'. It includes a 'Parameters' section which notes 'No parameters' and a 'Responses' section. The 'Responses' section lists standard HTTP status codes (200, 401, 403, 404, 500) with their corresponding error messages ('OK', 'Unauthorized', 'Forbidden', 'URI not found', 'Internal Error') and example values. A dropdown menu indicates the 'Response content type' is set to 'application/json'.

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.

Swagger "Try it out" functionality in the API Catalog

The API Catalog enables users to call service APIs through the **Try it out** functionality. There are 2 types of endpoints:

- **Public endpoints**

Endpoints that are accessible without entering user credentials.

- **Protected endpoints**

Endpoints that are only accessible by entering user credentials. These endpoints are marked with a lock icon.

Example:

API Catalog Current state information

GET /containers Lists catalog dashboard tiles

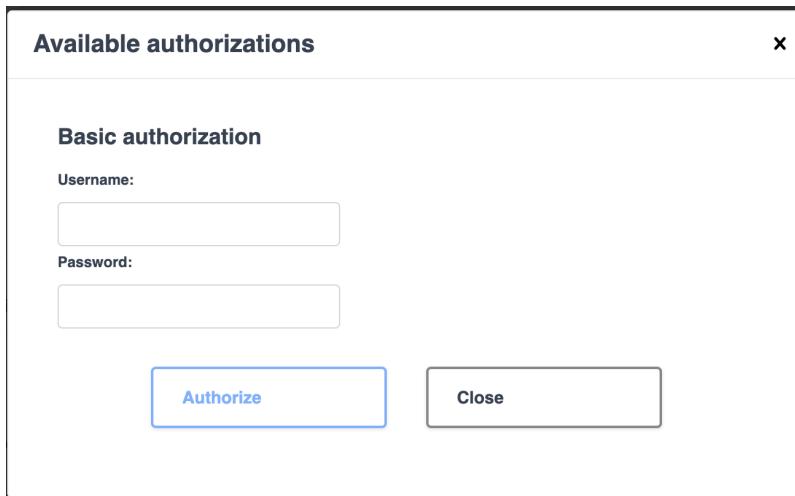
GET /containers/{id} Retrieves a specific dashboard tile information

API Documentation Service documentation

GET /apidoc/{serviceId}/{apiVersion} Retrieves the API documentation for a specific service version

Note: Before making requests to protected endpoints, authorize your session by clicking the lock icon and complete the required information in the Authorization modal shown below:

Example:



To demonstrate **Try it out**, we use the example of the Swagger Petstore.

Example:

[◀ Back](#)

Sample API Mediation Layer Applications

Applications which demonstrate how to make a service integrated to the API Mediation Layer ecosystem

[enablerv1sampleapp](#) [discoverableclient](#)

Service Spring Integration Enabler sample application API

[Service Homepage](#)

Sample API services to demonstrate Spring Integration Enabler

Service Spring Integration Enabler sample application API

API Version: 1.0.0

[Base URL: localhost:10010/api/v1/discoverableclient]

Sample API services to demonstrate Spring Integration Enabler

[Swagger/OpenAPI JSON Document](#)

[External documentation](#)

The pet API Pet Controller

| | | |
|---------------|------------|------------------------|
| GET | /pets/{id} | Find pet by id |
| PUT | /pets/{id} | Update an existing pet |
| DELETE | /pets/{id} | Delete a pet |
| GET | /pets | List all existing pets |
| POST | /pets | Add a new pet |

Make a request

This section outlines the process for making a request.

Follow these steps:

1. Expand the **POST Pet** endpoint.

This section outlines the process for making a request.

Follow these steps:

1. Expand the **POST Pet** endpoint.

2. Click Try it out.

Example:

The screenshot shows a 'Try it out' interface for a POST request to the '/pets' endpoint. The endpoint description is 'Creates a new pet'. There are two parameters: 'pet' (required) and 'object (body)'. The 'pet' parameter has a placeholder value: { "id": 1, "name": "Falco" }. The 'object (body)' parameter has a dropdown menu set to 'application/json'. Below the interface, there is a section titled 'Responses' with a 200 status code entry. The response body is labeled 'New created pet' and contains the same JSON object as the placeholder: { "id": 1, "name": "Falco" }.

After you click **Try it out**, the example value in the **Request Body** field becomes editable.

3. In the **Example Value** field, change the first id value to a random value. Change the second name value to a value of your choice, such as the name of a pet.

4. Click Execute.

Example:

The screenshot shows the API Catalog Swagger UI interface. At the top, it says "POST /pets Add a new pet". Below that, it says "Creates a new pet". Under "Parameters", there is a "pet * required" field with "object (body)" selected. The "Edit Value" field contains the following JSON:

```
{
  "id": null,
  "name": "Falco"
}
```

Below the parameters, there is a "Cancel" button. Under "Parameter content type", there is a dropdown menu set to "application/json". At the bottom, there is a large blue "Execute" button with a red border, which is highlighted by a large red rectangle. To the right of the execute button, there is some very small text that is mostly illegible.

The API Catalog Swagger UI submits the request and shows the *curl* that was submitted. The Responses section shows the response.

Example:

The screenshot shows the API Catalog Swagger UI interface under the "Responses" tab. It displays the following information:

- Curl:**

```
curl -X POST "https://localhost:10010/api/v1/discoverableclient/pets" -H "accept: application/json; charset=UTF-8" -H "Content-Type: application/json" -d "{<JSON>}"
```
- Request URL:**

```
https://localhost:10010/api/v1/discoverableclient/pets
```
- Server response:**

201
- Response body:**

Download

```
{
  "id": 6,
  "name": "Falco"
}
```

Static APIs refresh functionality in the API Catalog

The API Catalog enables users to manually refresh static service APIs. Use the **Refresh Static APIs** option if you change a static service API and want these changes to be visible in the API Catalog without restarting the Discovery Service.

Example:

The screenshot shows the API Catalog interface with a search bar at the top. Below it, the title "Available API services" is displayed. Three main sections are listed:

- API Mediation Layer API**: Describes the API Mediation Layer for z/OS internal API services. Status: All services are running.
- Sample API Mediation Layer Applications**: Describes applications demonstrating integration with the API Mediation Layer ecosystem. Status: All services are running.
- Static API Services**: Describes services demonstrating discoverability in the APIML ecosystem. Status: All services are running.

To refresh the status of a static service, click the **Refresh** option located in the upper right-hand side of the API Catalog UI. Successful requests return a pop-up notification that displays the message, **The refresh of static APIs was successful!**.

Example:

The screenshot shows the API Catalog interface after a refresh. A green notification bar at the top right displays the message "The refresh of static APIs was successful!". The main content area remains the same as the previous screenshot, showing the three service categories with their respective status indicators.

If the request fails, a dialog appears with an error message that describes the cause of the fail.

Example:

The screenshot shows the API Catalog interface. At the top, there's a search bar labeled "Search for APIs". Below it, a section titled "Available API services" lists the "API Mediation Layer API". A note below the service says: "The API Mediation Layer for z/OS internal API services. The API Mediation Layer provides a single point of access to mainframe REST APIs and offers enterprise cloud-like feature...". A green bullet point indicates "All services are running". To the right, there's a section titled "Sample API Mediation Layer Applications" which contains a red error message box. The error message reads: "(ZWEAC706E) 503 - Service Unavailable: The service could not be found. Check if the service is up and registered". A red "Close" button is at the bottom of the error box. The background features a light blue circuit board graphic.

Note: The manual **Refresh Static APIs** option applies only to static service APIs. Changes to the status of services that are onboarded to allow for dynamic discovery require a restart of the specific services where changes are applied. It is not necessary to restart the API Catalog or the Discovery Service.

Using Zowe SDKs

Leverage the Zowe Client Software Development Kits (SDKs) to build client applications and scripts that interface with the mainframe.

The SDKs include programmatic APIs, each of which performs a particular mainframe task. For example, one API package provides the ability to upload and download z/OS data sets. You can leverage that package to rapidly build a client application that interacts with data sets.

The following SDKs are available.

- Zowe Node.js Client SDK
- Zowe Python Client SDK

API documentation

For details about the available API endpoints, browse or download the [API Reference documentation](#).

Software requirements

Node.js

If you install Node SDK packages from the online registry, the required dependencies are installed automatically.

If you download Node SDK packages from Zowe.org, the folder contains dependencies that you must install manually. Extract the TGZ files from the folder, copy the files to your project, and issue the following commands to install the dependencies.

```
npm install imperative.tgz  
npm install core-for-zowe-sdk.tgz
```

Python

If you install Python SDK packages from the online registry, the required dependencies are installed automatically.

If you download the Python SDK packages from Zowe.org, the downloaded folder contains dependencies that you must install manually. Extract the WHL files from the folder, copy the files to your project, and issue the following command for each dependency:

```
pip install <fileName>.whl
```

Getting started

To get started, import the SDK packages to your project. You can pull the packages from an online registry, or download the packages from Zowe.org to install locally.

Install SDK from online registry

Pull the packages from an online registry such as npm or PyPi.

Follow these steps:

1. In command-line window, navigate to your project directory. Issue the following command to install a package from the registry:

- To import a Node.js package: `npm install <PackageName>`
- To import a Python package: `pip install <PackageName>`

where <packageName> is the name of the SDK package that you want to install, such as `zos-files-for-zowe-sdk`.

The packages are installed. Node packages are defined in `package.json` in your project. Python packages are installed by default to `$PYTHONPATH/Lib/site-packages` (Linux) or to the Python folder in your local / AppData folder (Windows).

2. **(Optional)** You might want to automatically update the SDK version when updates become available, or you might want to prevent automatic updates.
 - To define the versioning scheme for Node packages, use [npm semver](#).
 - To define versioning for Python packages, specify versions or version ranges in a `requirements.txt` file checked-in to your project. More information, see [pip documentation](#)

Install SDK from local package

Download and install the packages.

Follow these steps:

1. Navigate to [Zowe.org Downloads](#). Select your desired programming language in the **Zowe Client SDKs** section. The SDK is downloaded to your computer.
2. Unzip the SDK folder, which contains the packages for each set of functionality (such as z/OS Jobs). Copy each file that you want to install and paste them into your project directory.
3. Install required dependencies, which are included in the bundle. See [Software requirements](#) on page 237 above for more information.

- In a command-line window, navigate to your project directory. Issue *one* of the following commands.

- To install a Node.js package: `npm install <packageName>.tgz`
- To install a Python package: `pip install <packageName>.whl`

where `<packageName>` is the name of the package that you want to install, such as `zos-files-for-zowe-sdk`.

Repeat the command for each package that you need. Packages are now installed.

Using

After you install the SDK, you can make API calls to the mainframe from within your project.

Using - Node.js

For Node SDK usage and syntax examples, refer to the following package Readmes:

- [Core libraries](#) - Use shared libraries, such as `rest` to access z/OSMF REST APIs, `auth` for connecting to token-based authentication services, and more.
- [z/OS Console](#) - Perform z/OS console operations.
- [z/OS Files](#) - Work with data sets on z/OS.
- [z/OS Jobs](#) - Work with batch jobs on z/OS.
- [z/OS Management Facility](#) - Return data about z/OSMF, such as connection status or a list of available systems.
- [z/OS Provisioning](#) - Provision middleware and resources such as IBM CICS, IBM Db2, IBM MQ, and more.
- [z/OS TSO](#) - Interact with TSO/E address spaces on z/OS.
- [z/OS USS](#) - Work with UNIX system services (USS) files on z/OS.
- [z/OS Workflows](#) - Create and manage z/OSMF workflows on z/OS.

Using - Python

For information about the Python SDK, including usage and syntax examples, see the [Python SDK ReadTheDocs](#).

Contributing

For information about contributing to the open-source Zowe SDKs, see [Developing for Zowe SDKs](#) on page 396.

Zowe CLI extensions and plug-ins

Extending Zowe CLI

You can install plug-ins to extend the capabilities of 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 240
- [IBM® CICS® Plug-in for Zowe CLI](#) on page 244
- [IBM® Db2® Database Plug-in for Zowe CLI](#) on page 245
- [IBM® z/OS FTP Plug-in for Zowe CLI](#) on page 248
- [IBM® IMS™ Plug-in for Zowe CLI](#) on page 249
- [IBM® MQ Plug-in for Zowe CLI](#) on page 250
- [Secure Credential Store Plug-in for Zowe CLI](#) on page 251
- [Installing Zowe Explorer](#) on page 253

Software requirements for Zowe CLI plug-ins

Before you use Zowe™ CLI plug-ins, complete the following steps:

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.

| Plug-in | Required Configurations |
|--|---|
| IBM® CICS® Plug-in for Zowe CLI on page 244 | <ul style="list-style-type: none"> • 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 245 | <ul style="list-style-type: none"> • Download and prepare the ODBC driver (required for only package installations) and address the licensing requirements. • (MacOS) Download and Install Xcode. |
| IBM® z/OS FTP Plug-in for Zowe CLI on page 248 | <ul style="list-style-type: none"> • Ensure that z/OS FTP service is enabled and configured with JESINTERFACELEVEL = 2. • FTP over SSL is recommended. |
| IBM® IMS™ Plug-in for Zowe CLI on page 249 | <ul style="list-style-type: none"> • Ensure that IBM® IMS™ v14.1.0 or later is installed and running in your mainframe environment. • Configure IBM® IMS™ Connect. • Configure IBM IMS Operations APIs to enable communication between the CLI and the IMS instance. |
| IBM® MQ Plug-in for Zowe CLI on page 250 | <ul style="list-style-type: none"> • Ensure that IBM® MQ™ v9.1.0 or later is installed and running in your mainframe environment. Please read this blog for more information: Exposing the MQ REST API via the Zowe API Mediation Layer |
| Secure Credential Store Plug-in for Zowe CLI on page 251 | <ul style="list-style-type: none"> • (Graphical Linux) Install <code>gnome-keyring</code> and <code>libsecret</code> on your computer. • There are additional requirements for headless Linux systems. See the SCS plug-in Readme for details. |

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 241.
- 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 241.

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 239 that you install.
2. Issue the following command to install a plug-in from public npm:

```
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 | @zowe/cics-for-zowe-cli@zowe-v1-lts |
| IBM Db2 Plug-in for Zowe CLI | @zowe/db2-for-zowe-cli@zowe-v1-lts |
| IBM z/OS FTP Plug-in for Zowe CLI | @zowe/zos-ftp-for-zowe-cli@zowe-v1-lts |
| IBM IMS Plug-in for Zowe CLI | @zowe/ims-for-zowe-cli@zowe-v1-lts |
| IBM MQ Plug-in for Zowe CLI | @zowe/mq-for-zowe-cli@zowe-v1-lts |
| Secure Credential Store Plug-in for Zowe CLI | @zowe/secure-credential-store-for-zowe-cli@zowe-v1-lts |

3. (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 <@zowe/my-plugin1> <@zowe/my-plugin2> <@zowe/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 245.

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 239 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

- 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, or use your preferred method to unzip the files:

```
unzip zowe-cli-package-v.r.m.zip
```

Example:

```
unzip zowe-cli-package-1.9.0.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.

- Issue the following command against the extracted directory to install each available 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-for-zowe-cli.tgz |
| IBM Db2 Plug-in for Zowe CLI | db2-for-zowe-cli.tgz |
| IBM z/OS FTP Plug-in for Zowe CLI | zos-ftp-for-zowe-cli.tgz |
| IBM IMS Plug-in for Zowe CLI | ims-for-zowe-cli.tgz |
| IBM MQ Plug-in for Zowe CLI | mq-for-zowe-cli.tgz |
| Secure Credential Store Plug-in for Zowe CLI | secure-credential-store-for-zowe-cli.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 | Installation Command Syntax |
|-----------------------------------|-----------------------------|
| IBM CICS Plug-in for Zowe CLI | @zowe/cics-for-zowe-cli |
| IBM Db2 Plug-in for Zowe CLI | @zowe/db2-for-zowe-cli |
| IBM z/OS FTP Plug-in for Zowe CLI | @zowe/zos-ftp-for-zowe-cli |
| IBM IMS Plug-in for Zowe CLI | @zowe/ims-for-zowe-cli |
| IBM MQ Plug-in for Zowe CLI | @zowe/mq-for-zowe-cli |

| Plug-in | Installation Command Syntax |
|--|---|
| Secure Credential Store Plug-in for Zowe CLI | <code>@zowe/secure-credential-store-for-zowe-cli</code> |

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 @zowe/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 @zowe/my-plugin@zowe-v1-lts
```

The following example illustrates the syntax to use to update a plug-in to a specific version:

```
zowe plugins update @zowe/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 241.

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 | Installation Command Syntax |
|--|--|
| IBM CICS Plug-in for Zowe CLI | @zowe/cics-for-zowe-cli |
| IBM Db2 Plug-in for Zowe CLI | @zowe/db2-for-zowe-cli |
| IBM z/OS FTP Plug-in for Zowe CLI | @zowe/zos-ftp-for-zowe-cli |
| IBM IMS Plug-in for Zowe CLI | @zowe/ims-for-zowe-cli |
| IBM MQ Plug-in for Zowe CLI | @zowe/mq-for-zowe-cli |
| Secure Credential Store Plug-in for Zowe CLI | @zowe/secure-credential-store-for-zowe-cli |

Example:

The following example illustrates the command to uninstall the CICS plug-in:

```
zowe plugins uninstall @zowe/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 244
- [Commands](#) on page 244
- [Software requirements](#) on page 245
- [Installing](#) on page 245
- [Creating a user profile](#) on page 245

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 244 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 239.

Installing

Use one of the following methods to install or update the plug-in:

- [Installing plug-ins from an online registry](#) on page 241
- [Installing plug-ins from a local package](#) on page 241

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>
```

The plug-in uses HTTPS by default. Use the optional flag `--protocol http` to override the default with HTTP.

Note: For more information, issue the command `zowe profiles create cics --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 239.

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 246
- [Installing from a local package](#) on page 246

Installing from an online registry

If you installed Zowe CLI from **online registry**, complete the following steps:

1. Open a commandline window and issue the following command:

```
zowe plugins install @zowe/db2-for-zowe-cli@zowe-v1-lts
```

2. [Addressing the license requirement](#) on page 247 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 Xcode on MacOS

To install the Db2 CLI plug-in on MacOS, you need the command line tools, which can be obtained by installing Xcode from the [App Store](#).

Note: On some versions of MacOS, you may receive the error `xcrun: error: invalid active developer path` as shown below:

```
xcrun: error: invalid active developer path (/Library/Developer/
CommandLineTools), missing xcrun at: /Library/Developer/CommandLineTools/
usr/bin/xcrun
```

If this occurs, a manual refresh of the command line tools is required by running the following commands:

```
sudo rm -rf /Library/Developer/CommandLineTools
sudo xcode-select --install
```

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 173 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 247 to begin using the plug-in.

Addressing the license requirement

To successfully connect the Db2 CLI plug-in to a database on z/OS, a license needs to be present either on the client where the Zowe CLI is executed from, or else on z/OS. If you don't have a license configured when you execute Db2 CLI commands, you will receive an error SQL1598N, for example:

```
DB2 ODBC Driver Error: [node-ibm_db] SQL_ERROR
Error Details:
Error: [IBM][CLI Driver] SQL1598N An attempt to connect to the
database server failed because of a licensing problem.
```

Server-side license

You can execute the utility db2connectactivate on z/OS to enable a Db2 database to accept client requests. For more information, see [db2connectactivate - Server license activation utility](#). This avoids having to apply the Db2 Connect license on each database client that connects directly to the server. It is also the preferred approach to enabling users of the Zowe Db2 CLI because it avoids individual client license distribution and configuration.

Client-side license

If the utility db2connectactivate has not been executed against the Db2 database that your profile is connecting to, then it is possible to obtain the license file db2consv_zs.lic from a copy of DB2 Connect and use this for client configuration. This will need to be done separately for each client PC.

1. Locate your client copy of the Db2 license file db2consv_zs.lic.

Note: The license must be of version 11.5 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 db2consv_za.lic and place it in the following directory.

```
<zowe_home>/pluginsinstalled/lib/node_modules/@zowe/db2-for-zowe-cli/
node_modules/ibm_db/installer/clidriver/license
```

Tip: By default, <zowe_home> is set to ~/.zowe on /*UNIX Aand Mac 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 <profileName> -H <host> -P <port> -d <database> -u
<user> --pw <password>
```

Note For more information, issue the command `zowe profiles create db2-profile --help`

SQL0805N: Database BIND

To be able to run remote SQL commands against a Db2 database, you must invoke a `BIND` command against it. If the `BIND` command is not run, you will see an error that contains SQL0805N similar to the log below:

```
Command Error:
DB2 ODBC Driver Error: [node-ibm_db] Error in ODBCConnection::QuerySync
while executing query.Error Details:
Error: [IBM][CLI Driver][DB2] SQL0805N Package
"DSNV112E.NULLID.SYSSH200.5359534C564C3031" was not found.
```

If you receive this error, a user with DBADM authority must run the `BIND` command. This will typically be done by a Db2 System Programmer. More information can be found in the [Db2 product documentation](#) and [The Bind process](#).

IBM® z/OS FTP Plug-in for Zowe CLI

The IBM® z/OS FTP Plug-in for Zowe™ CLI lets you extend Zowe CLI to access z/OS datasets, USS files, and submit JCL. The plug-in uses the z/OS FTP service to achieve the interaction with z/OS.

- [Use cases](#) on page 248
- [Commands](#) on page 248
- [Software requirements](#) on page 249
- [Installing](#) on page 249
- [Creating a user profile](#) on page 249

Use cases

As a z/OS user, you can use the plug-in to perform the following tasks:

- List, view, rename, and download z/OS datasets or USS files.
- Upload local files or `stdin` to z/OS datasets or USS files.
- List, view, and download job status or job spool files.
- Delete a z/OS dataset, USS file, or job.

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 239.

Installing

Use one of the following methods to install or update the plug-in:

- [Installing plug-ins from an online registry](#) on page 241
- [Installing plug-ins from a local package](#) on page 241

Creating a user profile

You can create a `zftp` user profile to avoid typing your connection details on every command. A `zftp` profile contains the host, port, username, and password for the z/OS instance to which you want to connect. You can create multiple profiles and switch between them as needed.

Issue the following command:

```
```
zowe profiles create zftp <profile name> -H <host> -u <user> -p
<password> -P <port>
```
```

The result of the command displays as a success or failure message. You can use your profile when you issue commands in the `zftp` command group.

Note: There is an option named `--secure-ftp` that is set to `true` by default. If FTPS (FTP over SSL) is not enabled in z/OS FTP service, we recommend using `--secure-ftp false`. FTPS is not equivalent to SFTP (FTP over SSH).

Note: For more information about the syntax, actions, and options, for a profiles create command, open Zowe CLI and issue the following command:

```
zowe profiles create zftp -h
```

IBM® IMS™ Plug-in for Zowe CLI

The IBM IMS Plug-in for Zowe CLI lets you extend Zowe CLI such that it can interact with IMS resources (regions, programs and transactions). You can use the plug-in to start, stop, and query regions and start, stop, query, and update programs and transactions.

Note: For more information about IMS, see [IBM Information Management System \(IMS\)](#) on the IBM Knowledge Center.

[]

Use cases

As an application developer or DevOps administrator, you can use IBM IMS Plug-in for Zowe CLI to perform the following tasks:

- Refresh IMS transactions, programs, and dependent IMS regions.
- Deploy application code into IMS production or test systems.
- Write scripts to automate IMS actions that you traditionally perform using ISPF editors, TSO, and SPOC.

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 239.

Installing

Use one of the following methods to install or update the plug-in:

- [Installing plug-ins from an online registry](#) on page 241
- [Installing plug-ins from a local package](#) on page 241

Creating user profiles

You can set up an ims profile to retain your credentials, host, and port name. You can create multiple profiles and switch between them as needed. Issue the following command to create an ims profile:

```
zowe profiles create ims-profile <profileName> --host <hostname> --port
<portnumber> --ims-connect-host <ims-hostname> --ims-connect-port <ims-
portnumber> --user <username> --password <password>
```

Example: Setting up an IMS profile

The following example creates an ims profile named 'ims123' to connect to IMS APIs at host zos123 and port 1490. The name of the IMS plex in this example is 'PLEX1' and the IMS region we want to communicate with has a host of zos124 and a port of 1491:

```
zowe profiles create ims-profile ims123 --host zos123 --port 1490 --user
ibmuser --password myp4ss --plex PLEX1 --ich zos124 --icp 1491
```

Note: For more information, issue the command `zowe profiles create ims-profile --help`.

IBM® MQ Plug-in for Zowe CLI

The IBM MQ Plug-in for Zowe CLI lets you issue MQSC commands to a queue manager. MQSC commands let you to perform administration tasks. For example, you can define, alter, or delete a local queue object.

Note: For more information about MQSC commands and the corresponding syntax, see [MQSC commands](#) on the IBM Knowledge Center.

[]

Use cases

You can use the plug-in to execute MQSC Commands. With MQSC commands you can manage queue manager objects (including the queue manager itself), queues, process definitions, channels, client connection channels, listeners, services, namelists, clusters, and authentication information objects.

Using IBM MQ plug-in 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 239.

Installing

Use one of the following methods to install or update the plug-in:

- [Installing plug-ins from an online registry](#) on page 241
- [Installing plug-ins from a local package](#) on page 241

Creating a user profile

You can create an mq user profile to avoid typing your connection details on every command. An mq profile contains the host, port, username, and password for the MQ Rest API server of your choice. You can create multiple profiles and switch between them as needed.

Follow these steps:

1. Create an mq profile:

```
zowe profiles create mq-profile <profileName> --host <hostname> --port
<portnumber> --user <username> --password <password>
```

The result of the command displays as a success or failure message. You can use your profile when you issue commands in the mq command group.

Tip: For more information about the syntax, actions, and options, for a profiles create command, open Zowe CLI and issue the following command:

```
zowe profiles create mq-profile -h
```

Secure Credential Store Plug-in for Zowe CLI

The Secure Credential Store (SCS) Plug-in for Zowe CLI lets you store your credentials securely in the credential manager of your operating system. The plug-in invokes a native Node module, [keytar](#), that manages user IDs and passwords in a credential manager.

- [Use Cases](#) on page 251
- [Commands](#) on page 251
- [Software requirements](#) on page 252
- [Installing](#) on page 252
- [Using](#) on page 252

Use Cases

Zowe CLI stores credentials (mainframe username and password) in plaintext on your computer by default. You can use the SCS plug-in to store credentials more securely and prevent your credentials from being compromised as a result of a malware attack or unlawful actions by others.

Commands

For detailed command, actions, and option documentation for this plug-in, see our Web Help (available online or as PDF or ZIP):

- [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 239.

Installing

Use one of the following methods to install or update the plug-in:

- [Installing plug-ins from an online registry](#) on page 241
- [Installing plug-ins from a local package](#) on page 241

Note: Existing user profiles are *not* automatically updated to securely store credentials.

Using

The plug-in introduces a new command group, `zowe scs`, that lets you update existing user profiles and enable/disable the plug-in.

Securing your credentials

User profiles that you create *after* installing the plug-in will automatically store your credentials securely.

To secure credentials in existing user profiles (profiles that you created prior to installing the SCS plug-in), issue the following command:

```
zowe scs update
```

Profiles are updated with secured credentials.

Example: Secure credentials

The following is an example of securely stored credentials in a user profile configuration file:

```
type: zosmf
host: test
port: 1234
user: 'managed by @zowe/secure-credential-store-for-zowe-cli'
password: 'managed by @zowe/secure-credential-store-for-zowe-cli'
rejectUnauthorized: false
```

Example: Default credential management

The following is an example of credentials that are stored with the *default* credential manager:

```
type: zosmf
host: test
port: 1234
user: USERNAME
password: PASSWORD
rejectUnauthorized: false
```

Deactivating the plug-in

If you do not want to use the SCS Plug-in for Zowe CLI, choose one of the following methods to deactivate the plug-in:

Uninstall the Plug-in

Issue the `zowe plugins uninstall @zowe/secure-credential-store-for-zowe-cli` command to delete the plug-in from your computer.

When you uninstall the plug-in, existing profiles become invalid and you must recreate them. For more information, see [Using profiles](#).

Reset the Configuration of Credential Manager

Issue the `zowe config reset CredentialManager` command to reset the value of the credential manager configuration to default, which deactivates the plug-in.

Zowe Explorer

Installing Zowe Explorer



The Zowe Explorer extension for Visual Studio Code (VSCode) modernizes the way developers and system administrators interact with z/OS mainframes, and lets you interact with data sets, USS files and jobs. Install the extension directly to [VSCode](#) to enable the extension within the GUI. Working with data sets and USS files from VSCode can be more convenient than using 3270 emulators, and complements your Zowe CLI experience. The extension provides the following benefits:

- Enables you to create, modify, rename, copy, and upload data sets directly to a z/OS mainframe.
- Enables you to create, modify, rename, and upload USS files directly to a z/OS mainframe.
- Provides a more streamlined way to access data sets, USS files and jobs.
- Lets you create, edit, and delete Zowe CLI `zosmf` compatible profiles.
- Lets you use the Secure Credential Store plug-in to store your credentials securely in the settings.

Note: Zowe Explorer is a subcomponent of [Zowe](#). The extension demonstrates the potential for plug-ins powered by Zowe.

- [Software Requirements](#) on page 253
- [Installing](#) on page 253
- [Configuration](#) on page 254
- [Relevant Information](#) on page 256

Software Requirements

Ensure that you meet the following prerequisites before you use the extension:

- Get access to z/OSMF.
- Install [Node.js](#) v8.0 or later.
- Install [VSCode](#).
- Configure TSO/E address space services, z/OS data set, file REST interface, and z/OS jobs REST interface. For more information, see [z/OS Requirements](#).
- Create one Zowe CLI `zosmf` profile so that the extension can communicate with the mainframe.

Notes:

- i. You can use your existing Zowe CLI `zosmf` profiles that are created with the Zowe CLI v.2.0.0 or later.
- ii. Zowe CLI `zosmf` profiles that are created in Zowe Explorer can be interchangeably used in the Zowe CLI.

Installing

1. Address [Software Requirements](#) on page 253.
2. Open VSCode, and navigate to the **Extensions** tab on the left-hand side of the UI.
3. Type **Zowe Explorer** in the search field.

Zowe Explorer appears in the list of extensions in the left-hand panel.

4. Click the green **Install** button to install the extension.

5. Restart VSCode.

The extension is now installed and available for use.

- **Note:** For information about how to install the extension from a VSIX file and run system tests on the extension, see the [Developer README](#).

You can also watch the following videos to learn how to get started with Zowe Explorer, and work with data sets.

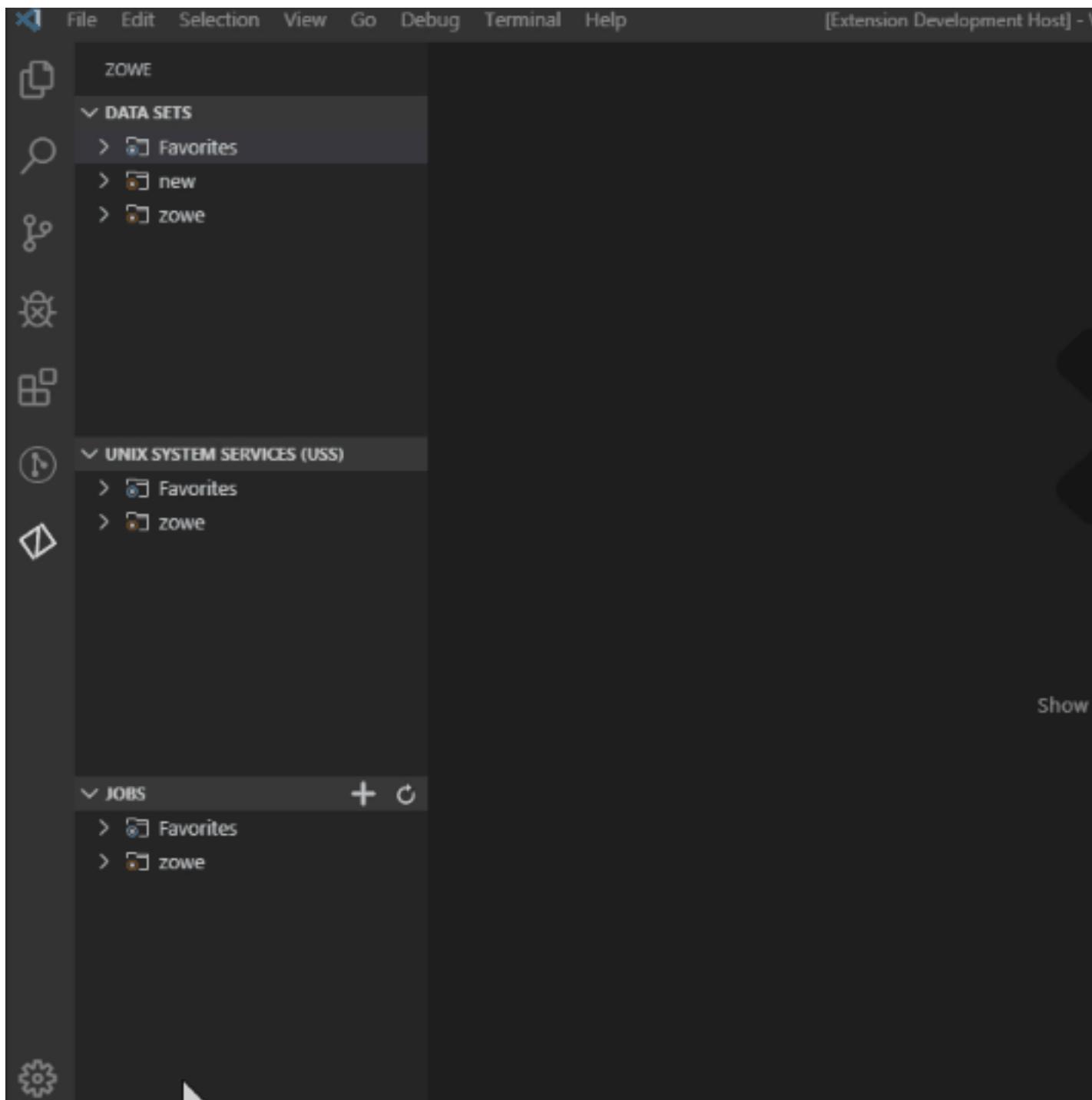
Configuration

Configure Zowe Explorer in the settings file of the extension. To access the extension settings, navigate to **Manage (the gear icon) > Settings**, then select **Extensions > Zowe Explorer Settings**. For example, you can modify the following settings:

- **Data set creation settings:** You can change the default creation settings for various data set types.

Follow these steps:

1. Click the **Edit in settings.json** button under the Data Set, USS or JOBS settings that you want to edit.
2. Edit the settings as needed.
3. Save the settings.



- **Set the Temporary Folder Location:** You can change the default folder location where temporary files are stored.

Follow these steps:

1. Click the **Edit in settings.json** button under the Data Set, USS or JOBS settings that you want to edit.
2. Modify the following definition:

```
"Zowe-Temp-Folder-Location": {  
    "folderPath": "/path/to/directory"
```

```
}
```

where `/path/to/directory` is the folder location that you specify.

1. Save the settings.

Relevant Information

In this section you can find useful links and other relevant to Zowe Explorer information that can improve your experience with the extension. Check the following links:

- For information about how to develop for Eclipse Theia, see [Theia README](#).
- For information about how to create a VSCode extension for Zowe Explorer, see [VSCode extensions for Zowe Explorer](#).
- Visit the [#zowe-explorer](#) channel on [Slack](#) for questions and general guidance.

Zowe Explorer Profiles

After you install Zowe Explorer, you need to have a Zowe Explorer profile to use all functions of the extension. You can optionally activate the Secure Credential Store plug-in to securely store your credentials.

Working with Zowe Explorer profiles

You must have a zosmf compatible profile before you can use Zowe Explorer. You can set up a profile to retain your credentials, host, and port name. In addition, you can create multiple profiles and use them simultaneously.

Follow these steps:

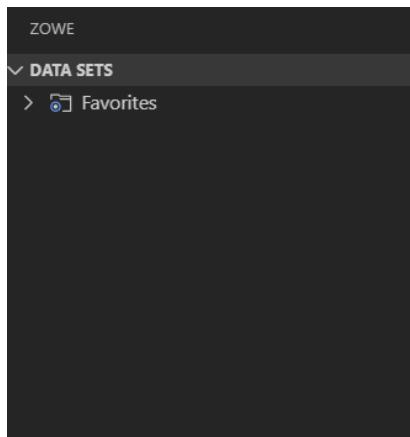
1. Navigate to the explorer tree.
2. Click the + button next to the **DATA SETS**, **USS** or **JOBS** bar.

Note: If you already have a profile, select it from the drop-down menu.

3. Select the **Create a New Connection to z/OS** option.

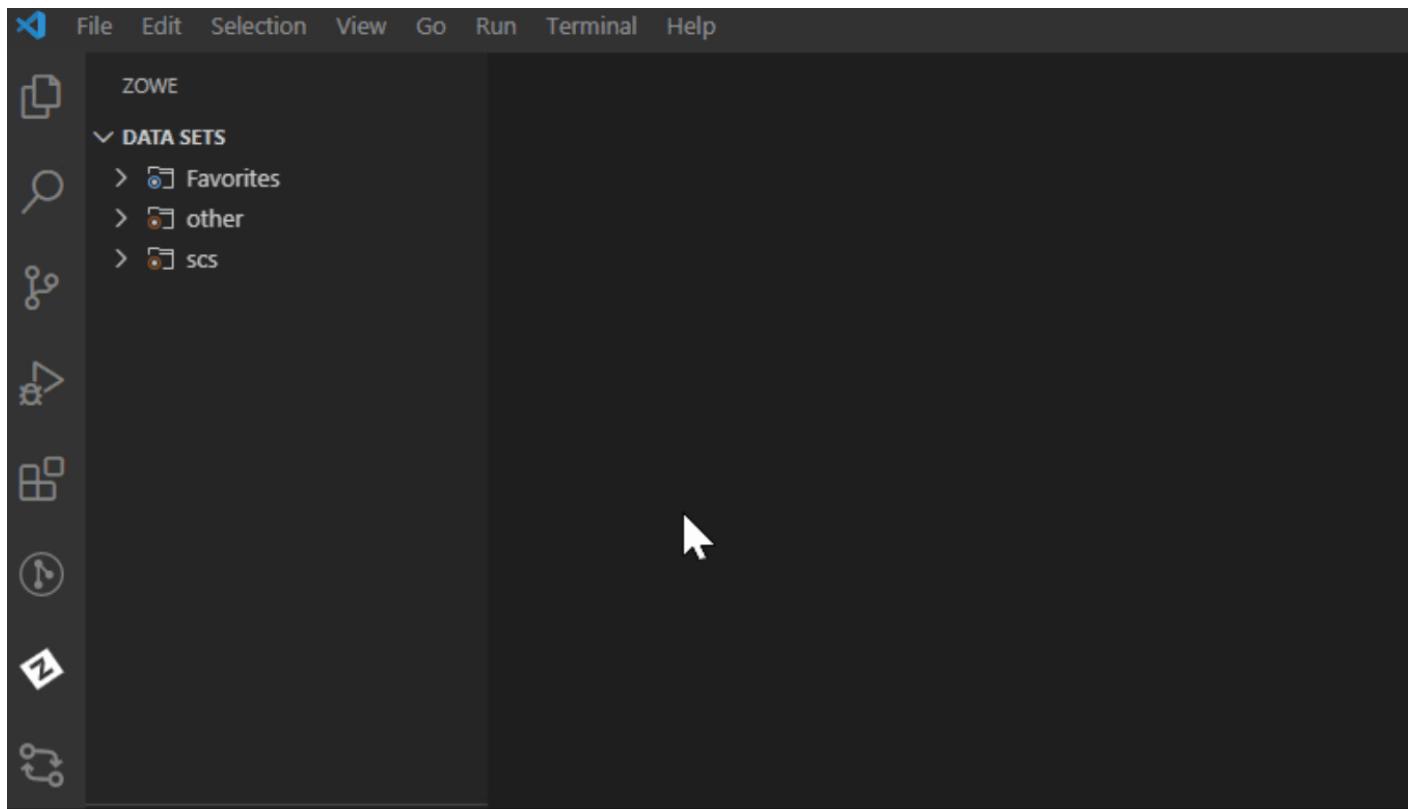
Note: When you create a new profile, user name and password fields are optional. However, the system will prompt you to specify your credentials when you use the new profile for the first time.

4. Follow the instructions, and enter all required information to complete the profile creation.



You successfully created a Zowe CLI zosmf profile. Now you can use all the functionalities of the extension.

If you need to edit a profile, click the **Update Profile** button next to the corresponding profile.



In addition, you can hide a profile from the explorer tree, and permanently delete a profile. When you delete your profile permanently, the extension erases the profile from the .zowe folder. To hide or delete a profile, right-click the profile and choose one of the respective options from the list.

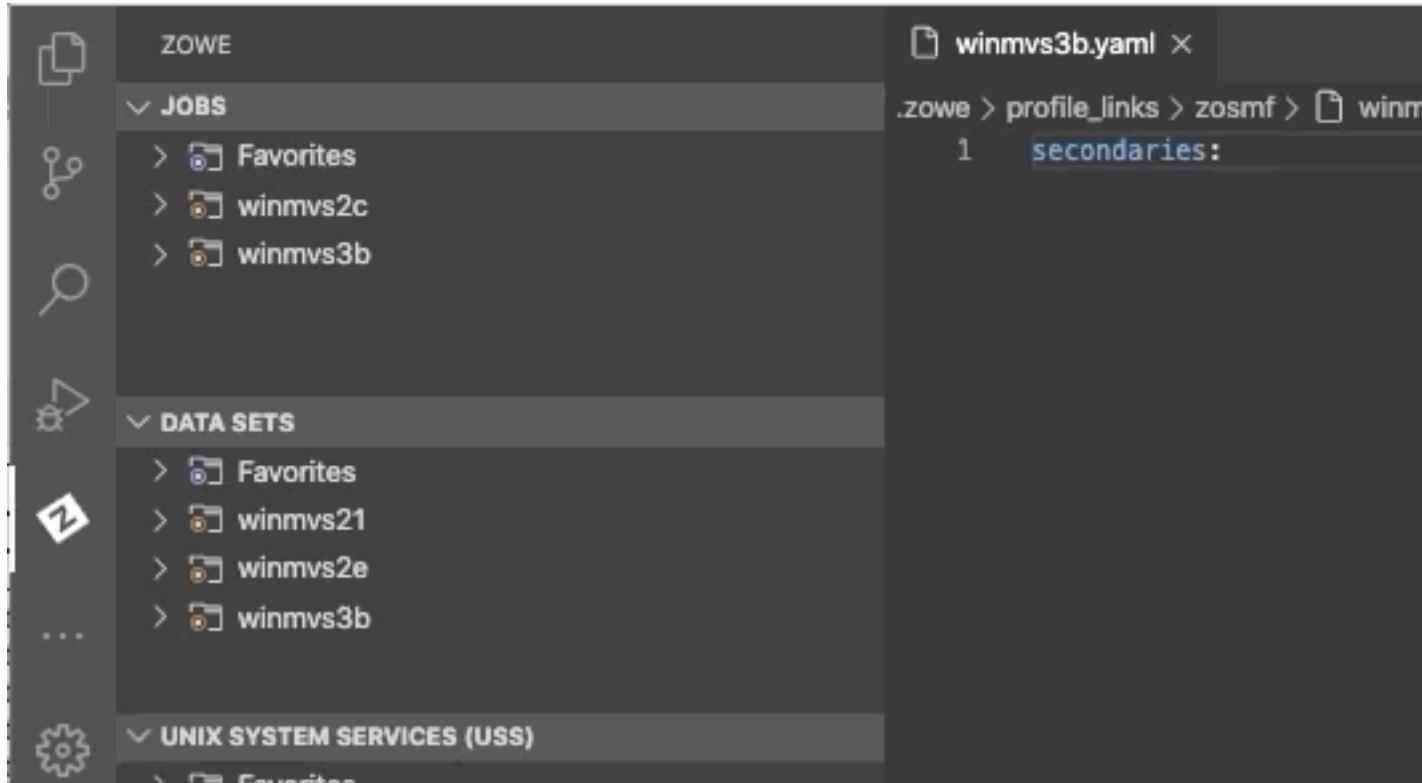
Associate Profile

Extensions built on Zowe Explorer provide users with additional functionality through unique profile types, such as RSE or FTP. The "associate profiles" function allows users to link, or associate, these extension profile types with an existing zOSMF profile. Every Zowe Explorer profile has two types of association: primary and secondary. While the primary association is zosmf, the secondary association includes Zowe CLI plug-ins or other services you might have. Within the scope of their own extender package, associated profiles can be used to access a relevant REST API that is available to the extender.

Create a secondary association for your Zowe Explorer profiles.

Follow these steps:

1. Navigate to the explorer tree.
2. Click the associate profiles button.
3. Select a secondary profile type.
4. Select a file name you want to create an association for.



You have successfully created the secondary association for your Zowe Explorer profile.

For more information, see [Associating Zowe CLI Profiles](#).

Enabling Secure Credential Store with Zowe Explorer

Store your credentials securely by using the Secure Credential Store (SCS) plug-in in Zowe Explorer. By default, your credentials are stored in plain text.

Activate the SCS plug-in in Zowe Explorer.

Follow these steps:

1. Open Zowe Explorer.
2. Navigate to the VSCode settings.
3. Open Zowe Explorer Settings.
4. Add the **Zowe-Plugin** value to the Zowe Security: Credential Key entry field.
5. Restart VSCode.
6. Create a profile.

Your Zowe Explorer credentials are now stored securely.

For Zowe CLI users

Ensure that you install the SCS plug-in for Zowe CLI before activating SCS in Zowe Explorer. For more information about the SCS plug-in for Zowe CLI, see [Secure Credential Store Plug-in for Zowe CLI](#) on page 251.

Important: If you did not install the SCS plug-in for Zowe CLI and try to activate SCS in the extension, you will not be able to use your existing profiles, and will have to recreate them.

Activate the SCS plug-in in Zowe Explorer.

1. Open Zowe CLI and issue the following command:

```
zowe scs u
```

2. Open Zowe Explorer.
3. Navigate to the VSCode settings.
4. Open Zowe Explorer Settings.
5. Add the **Zowe-Plugin** value to the Zowe Security: Credential Key entry field.
6. Restart VSCode.

The credentials of your newly created or existing profiles are now stored securely.

Use Base Profile and Token with Existing Profiles

As a Zowe user, you can leverage the base profile functionality to access multiple services through Single Sign-on. Base profiles enable you to authenticate via Zowe API Mediation Layer. You can use base profiles with more than one service profile. For more information, see [Base Profiles](#).

Before you log in and connect your service profile, ensure that you have [Zowe CLI](#) v6.16 or higher installed.

Access services through API ML with SSO

Connect your service profile with a base profile and token.

Follow these steps:

1. Open Zowe CLI and run the following command: `zowe auth login apiml`.
2. Follow the instructions to complete the login process.

A local base profile is created that contains your token. For more information about the process, see [Token Management](#).

3. Run Zowe Explorer and click the + icon.
4. Select the profile you use with your base profile with the token.

The profile appears in the tree and you can now use this profile to access z/OSMF via the API Mediation Layer.

For more information, see [Integrating with API Mediation Layer](#).

Log in to the Authentication Service

If the token for your base profile is no longer valid, you can log in again to get a new token with the **Log in to Authentication Service** feature.

Notes:

- The feature is only available for base profiles.
- The feature supports only API Mediation Layer at the moment. Other extenders may use a different authentication service.

Follow these steps:

1. Open Zowe Explorer.
2. Right-click your profile.
3. Select the **Log in to Authentication Service** option.

You will be prompted to enter your username and password beforehand.

The token is stored in the default base profile .yaml file.

If you do not want to store your token, request from the server to end the session of your token. Use the **Log out from Authentication Service** feature to invalidate the token.

Follow these steps:

1. Open Zowe Explorer.
2. Right-click your profile.
3. Select the **Log out from Authentication Service** option.

Your token has been successfully invalidated.

Using Zowe Explorer

Review this section to familiarize yourself with the extension and make the best use of available options and features. The section contains usage tips and sample use cases for data sets, USS files, JOBS, and TSO commands.

Usage Tips

Make the best use of the extension with the following tips:

- **Data set, USS and jobs persistence settings:** You can store any of data sets, USS files, or jobs permanently in the **Favorites** tab. Right-click on a data set, USS file, or job and click **Add Favorite**.
- **Identify syntax errors with a syntax highlighter:** Zowe Explorer supports a syntax hightlighter for data sets. To enhance the experience of using the extension, download an extension that highlights syntax, such as [COBOL Language Support](#) or [HLLAS Language Support](#).
- **Edit a profile:** You can edit existing profiles by clicking the **pencil** button next to the **magnifying glass** button in the explorer tree. The feature lets you modify the information inside your profile.
- **Delete a profile:** You can permanently delete profiles by right-clicking the profile and selecting the **Delete Profile** option. The feature deletes a profile from your .zowe folder. Alternatively, you can delete a profile from the VSCode Command Palette.

Follow these steps:

1. Press **F1** on your keyboard.
 2. Select the **Zowe: Delete a Profile Permanently** option.
- **Hide a profile:** You can hide a profile from profile tree by right-clicking the profile and selecting the **Hide Profile** option. If necessary, add the profile back by clicking the + button from the explorer tree.

Sample Use Cases

Review the following use cases to understand how to use Zowe Explorer.

- [Work with Data Sets](#) on page 260
- [Work with USS Files](#) on page 272
- [Work with jobs](#) on page 275
- [MVS/TSO Commands](#)

Work with Data Sets

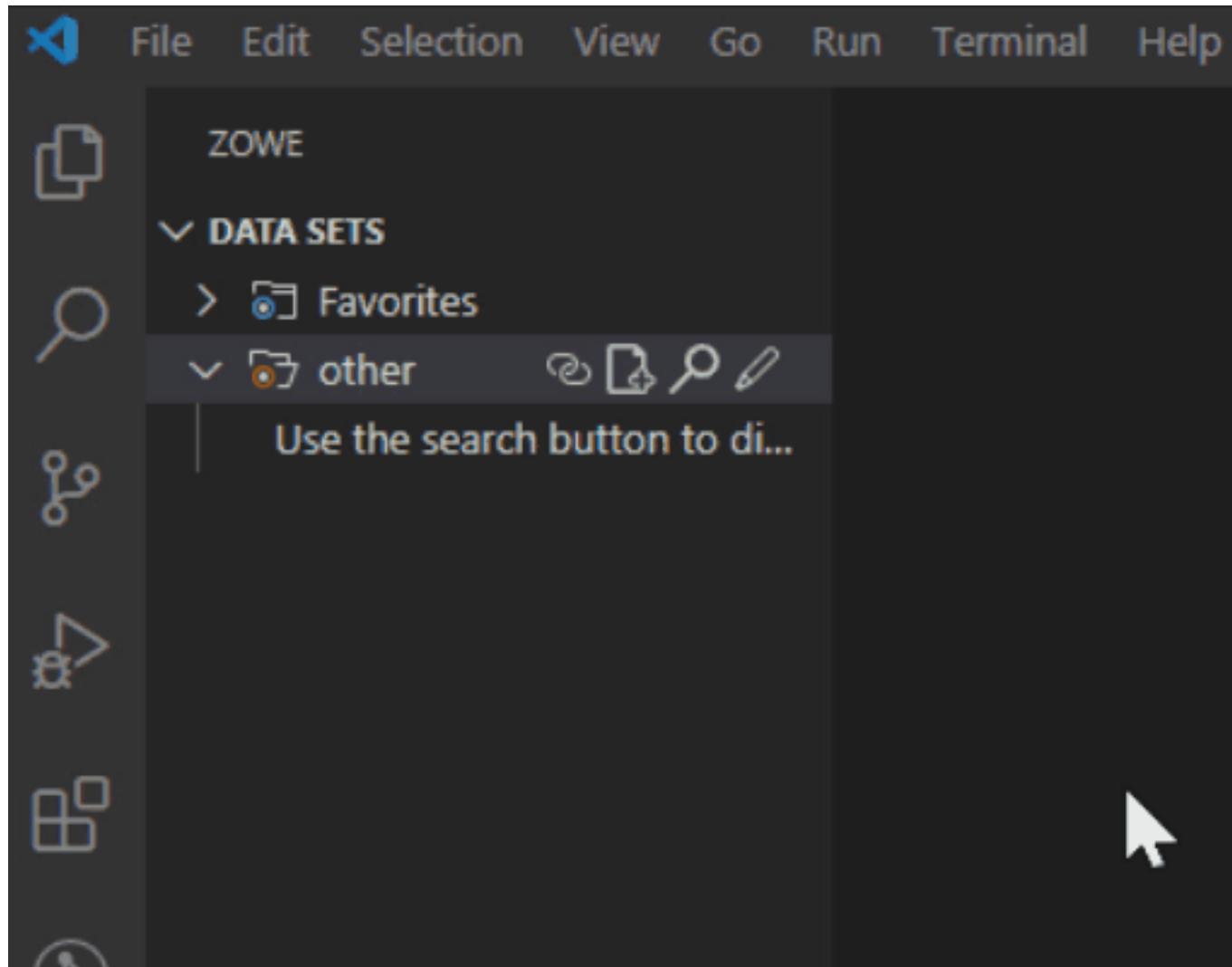
You can use the following functionalities when interacting with data sets:

- **View data sets and use multiple filters:** You can view multiple data sets simultaneously and apply filters to show specified data sets.
- **Rename data sets:** You can rename specified data sets.
- **Copy data sets:** You can copy a specified data sets and members.
- **Download, edit, and upload existing PDS members:** You can instantly pull data sets and data set members from the mainframe, edit them and upload back.
- **Create and delete data sets and data set members:** Enables you to easily create and delete both data sets and their members.
- **View and access multiple profiles simultaneously:** Enables you to work with data sets from multiple profiles.
- **Submit a JCL:** You can submit a jcl from a chosen data set.
- **Allocate Like:** You can create a copy of a chosen data set with the same parameters.

View data sets and use multiple filters

1. Navigate to the explorer tree.
2. Open the **DATA SETS** bar.
3. Select the profile that you want to filter.
4. Click the **Search Data Sets by Entering Patterns** magnifying glass.
5. From the drop-down, enter the patterns that you want to filter. The data sets that match your pattern(s) display in the explorer tree.

Tip: To provide multiple filters, separate entries with a comma. You can append or postpend any filter with an *, which indicates wildcard searching. You cannot enter an * as the entire pattern.

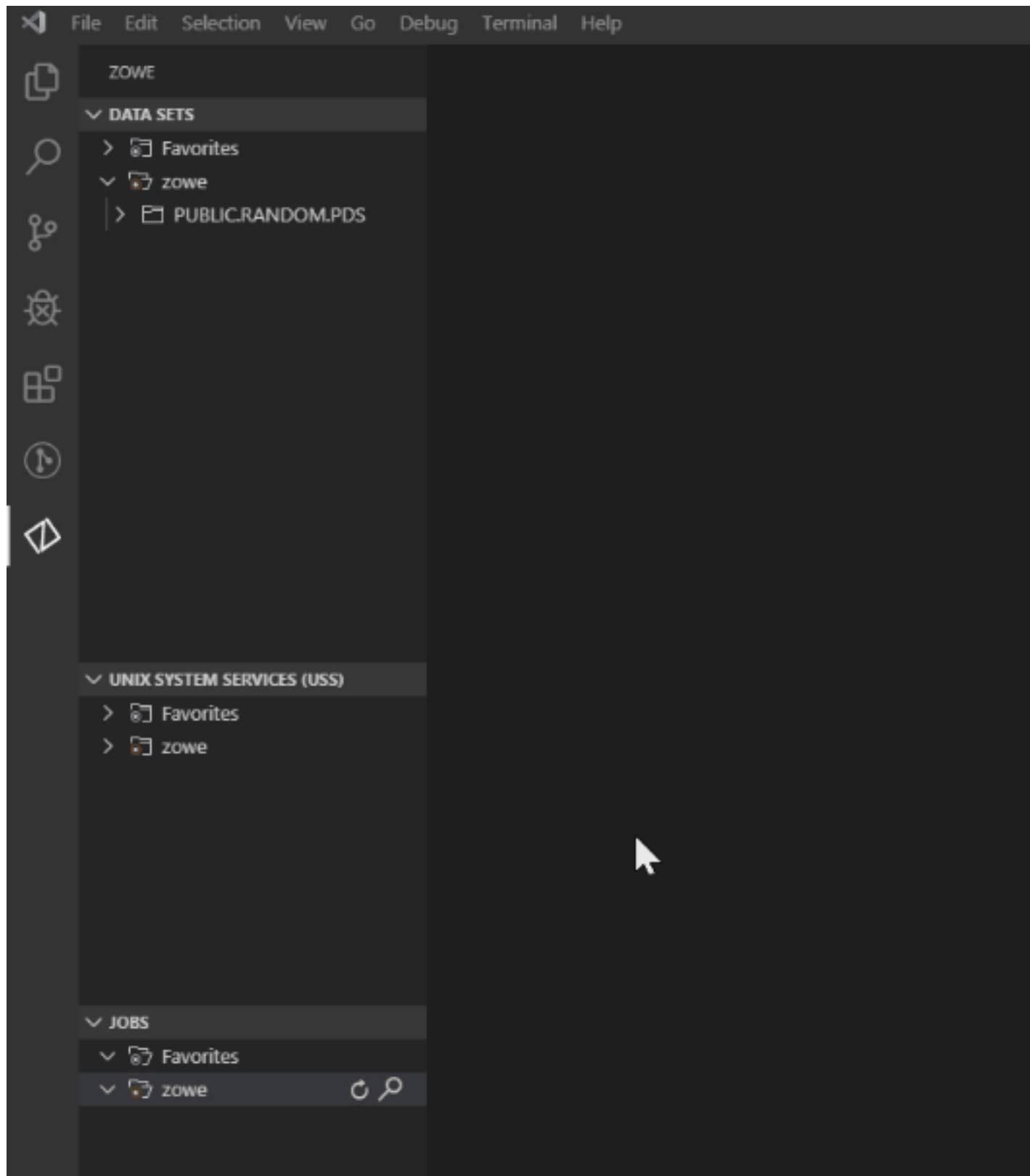


Refresh the list of data sets

1. Navigate to the explorer tree.
2. Click **Refresh All** button on the right of the **DATA SETS** explorer bar.

Rename data sets

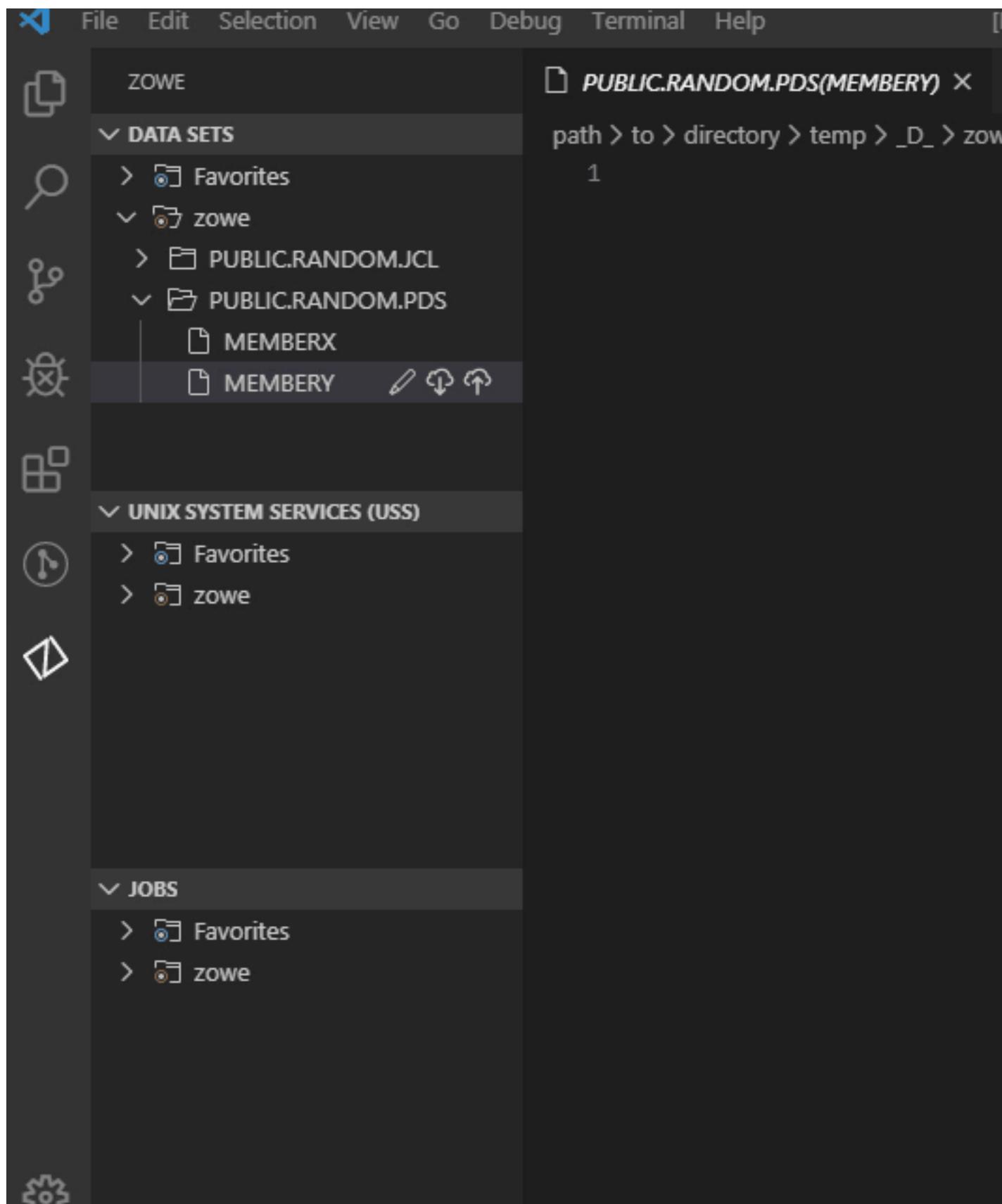
1. Navigate to the explorer tree.
2. Open the **DATA SETS** bar.
3. Select a data set you want to rename.
4. Right-click the data set and select the **Rename Data Set** option.
5. Change the name of the data set.



Copy data sets

1. Navigate to the explorer tree.
2. Open the **DATA SETS** bar.
3. Select a member you want to copy.

4. Right-click the member and select the **Copy Data Set** option.
5. Right-click the data set where the member belongs and select the **Paste Data Set** option.
6. Enter the name of the copied member.



Download, edit, and upload existing PDS members

1. Navigate to the explorer tree.

2. Open the **DATA SETS** bar.
3. Open a profile.
4. Select the PDS member (or PS) that you want to download.

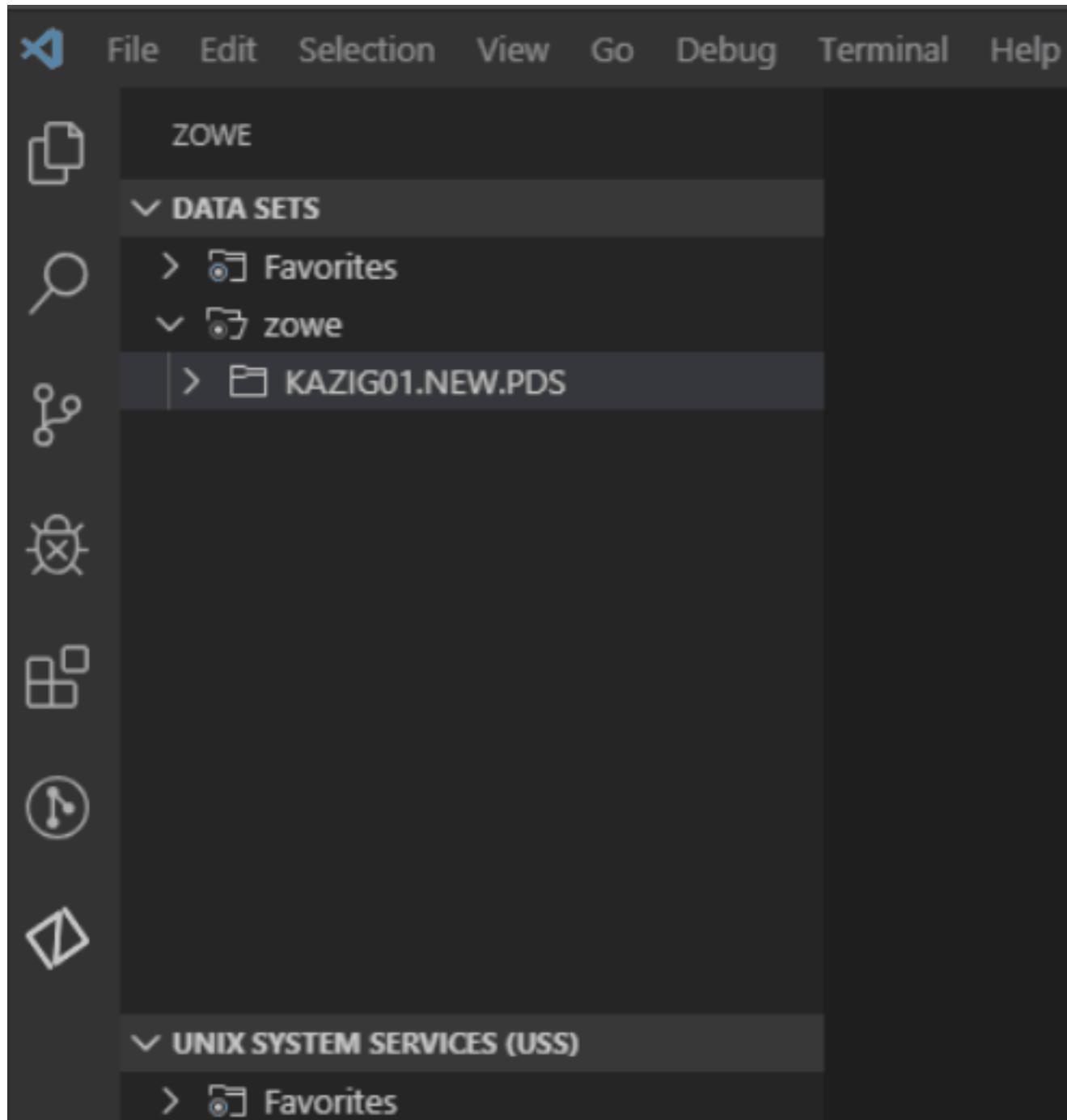
Note: To view the members of a PDS, click on the PDS to expand the tree.

The PDS member displays in the text editor window of VSC.

5. Edit the document.
6. Navigate back to the PDS member (or PS) in the explorer tree, and click the **Save** button.

Your PDS member (or PS) is uploaded.

Note: If someone else has made changes to the PDS member (or PS) while you were editing it, you can merge your conflicts before uploading to the mainframe.



Use the save option to prevent merge conflicts

1. Navigate to the explorer tree.
2. Open the **DATA SETS** bar.
3. Open a member of a data set you want to edit.
4. Edit the member.
5. Press Ctrl+S or Command+S (OSx) to save your changes.
6. (Optional) Resolve merge conflicts if necessary.

The screenshot shows a Zowe interface with a dark-themed sidebar and a main content area.

Left Sidebar:

- ZOWE**: A folder icon.
- DATA SETS**:
 - > Favorites
 - > other
 - > PUBLIC.DATASET.NEW
- UNIX SYSTEM SERVICES (USS)**:
 - > Favorites
 - > zowe
- JOB**:
 - > Favorites
 - > demo
 - > zowe

Main Content Area:

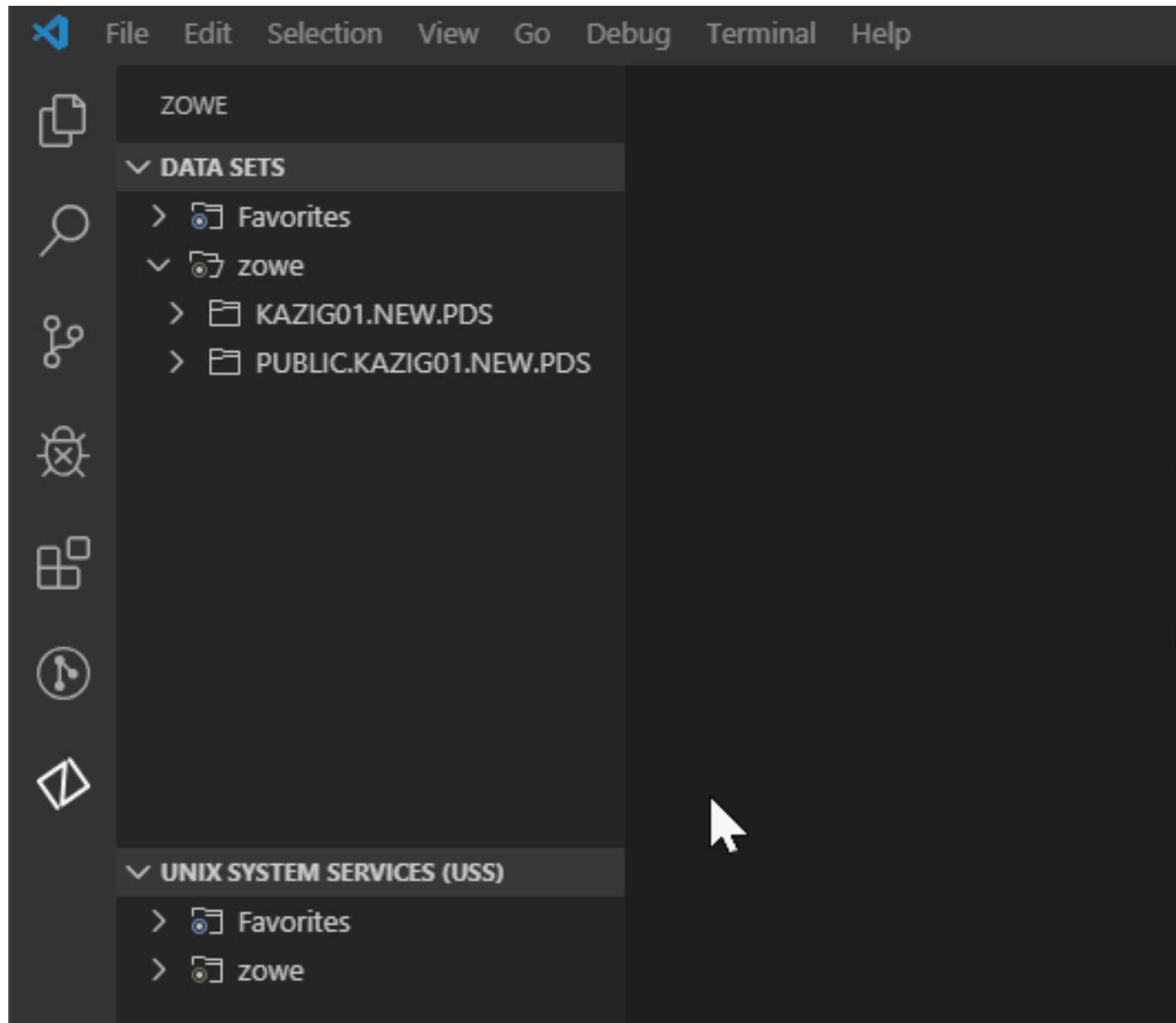
A file viewer window is open, showing the following details:

- Title:** PUBLIC.DATASET.NEW(MEMBER1) X
- Path:** path > to > directory > temp > _D_ > other > PUBLIC.DATASET.NEW(MEMBER1)
- Content:**

```
1 Hello, Zowe!
2
3
```

Create a new PDS and a PDS member

1. Navigate to the explorer tree.
2. Open the **DATA SETS** bar.
3. Click the **Create New Data Set** button to specify the profile that you want to create the data set with.
4. From the drop-down menu, select the type of PDS that you want to create.
5. Enter a name for the PDS. The PDS is created.
6. To create a member, right-click the PDS and select **Create New Member**.
7. Enter a name for the member. The member is created.



Delete a PDS member and PDS

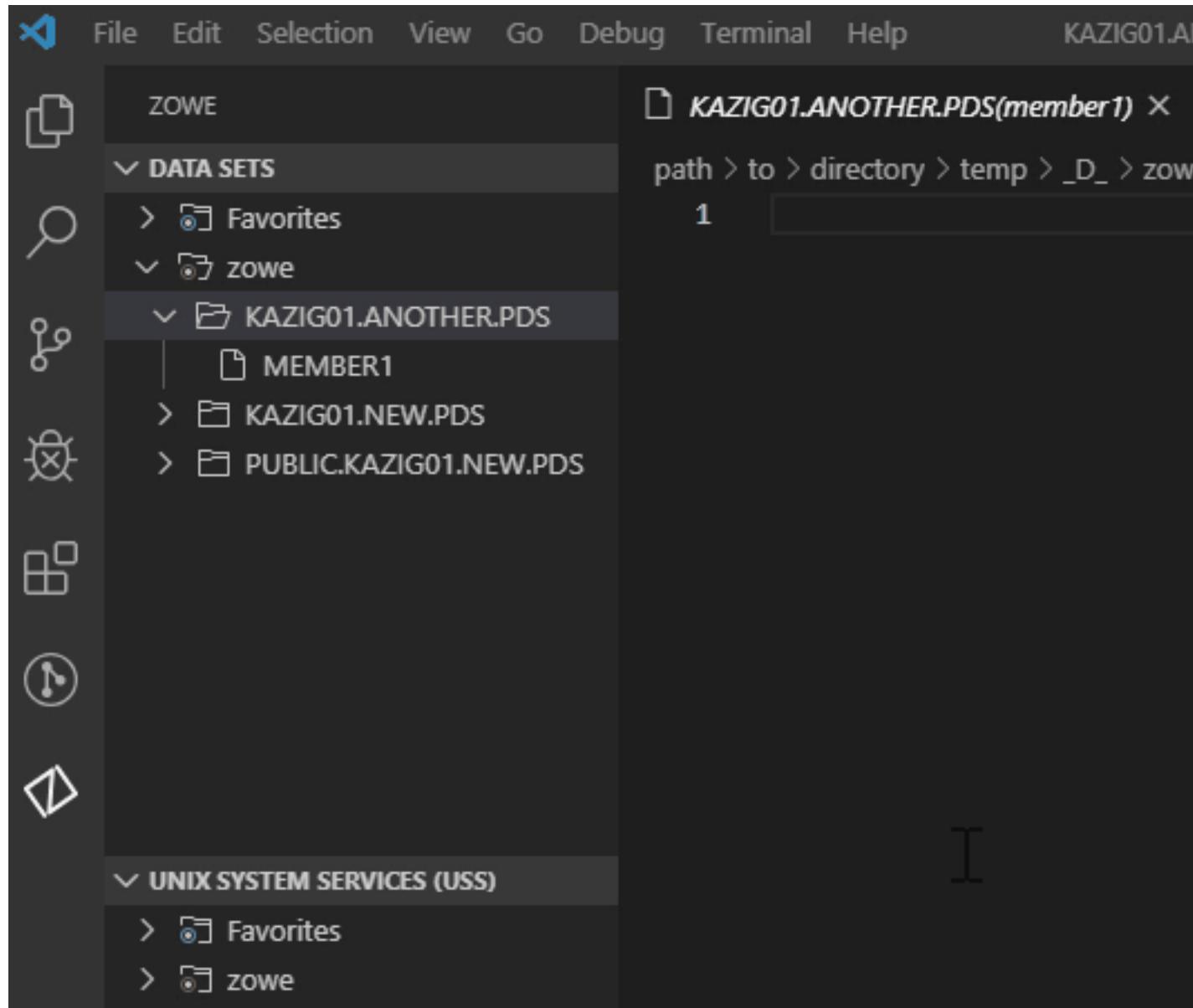
1. Navigate to the explorer tree.
2. Open the **DATA SETS** bar.
3. Open the profile and PDS containing the member.
4. Right-click on the PDS member that you want to delete and select **Delete Member**.

5. Confirm the deletion by clicking **Yes** on the drop-down menu.

Note: Alternatively, you can select 'No' to cancel the deletion.

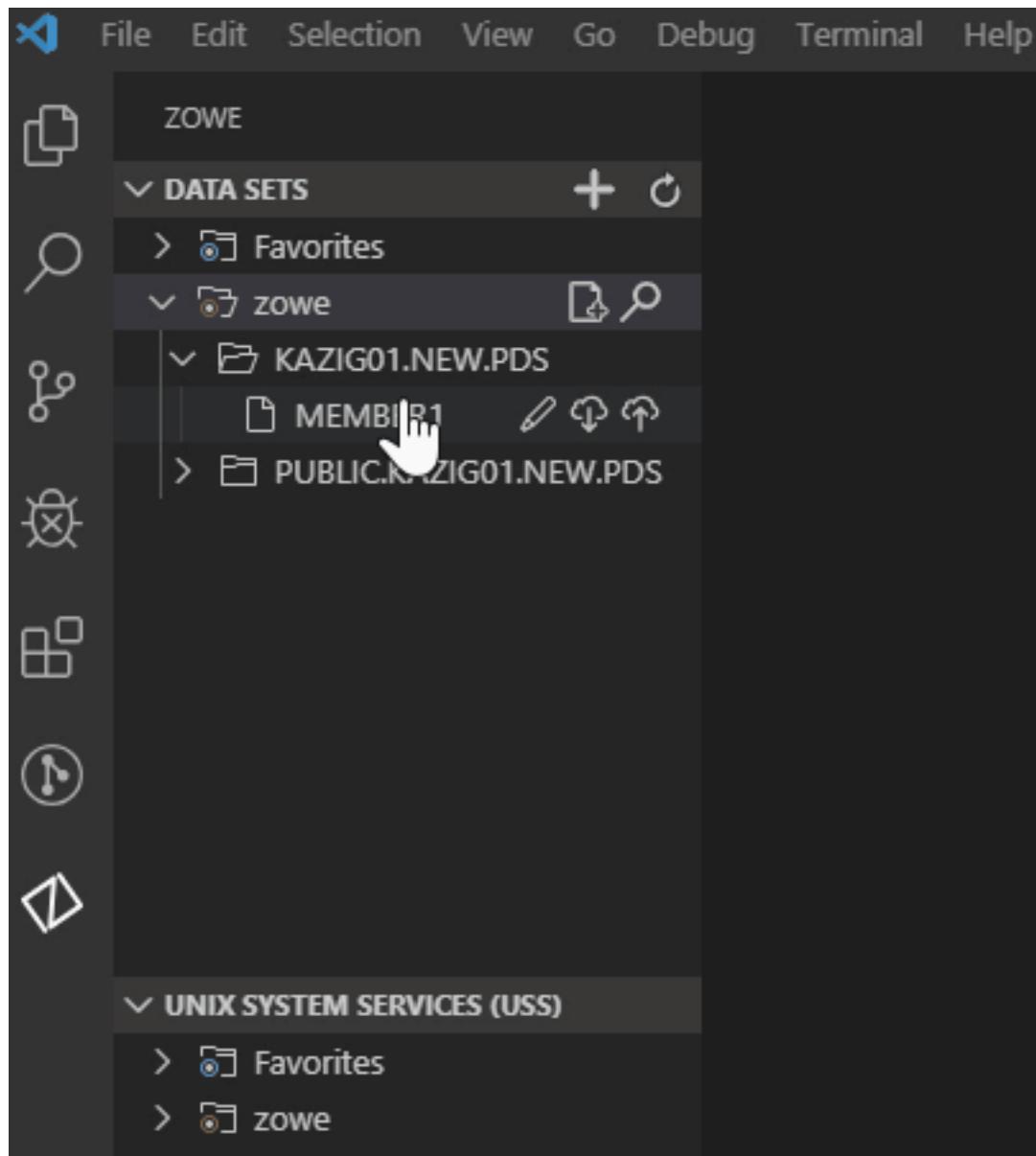
6. To delete a PDS, right-click the PDS and click **Delete PDS**, then confirm the deletion.

Note: You can delete a PDS before you delete its members.



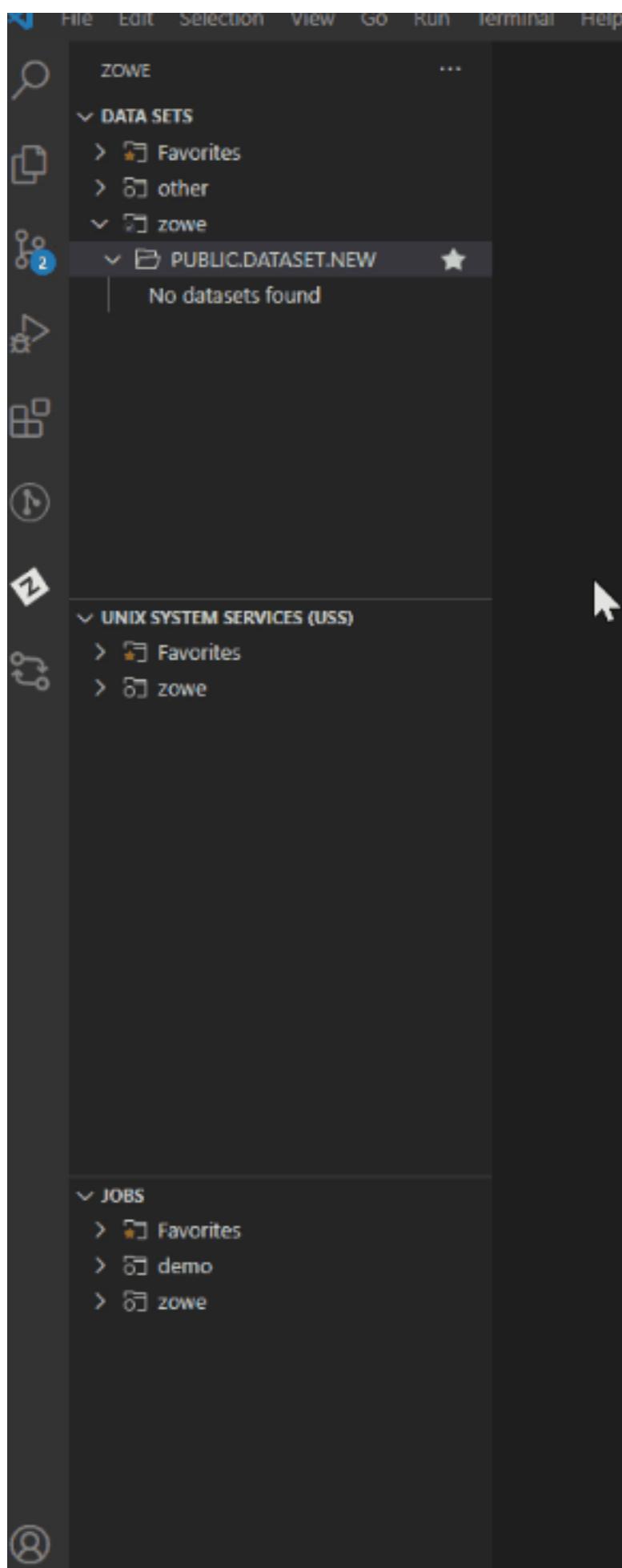
View and access multiple profiles simultaneously

1. Navigate to the explorer tree.
2. Open the **DATA SETS** bar.
3. Click the **Add Profile** button on the right of the **DATA SET** explorer bar.
4. Select the profile that you want to add to the view as illustrated by the following screen.



Allocate Like

1. Navigate to the explorer tree.
2. Open the **DATA SETS** bar.
3. Right-click the data set and select the **Allocate Like (New File with Same Attributes)** option.
4. Enter the new data set name.



Work with USS Files

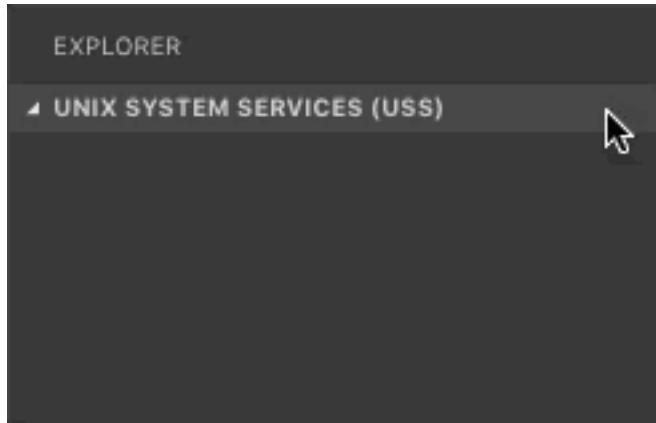
You can use the following functionalities when interacting with USS files:

- **View Unix System Services (USS) files:** You can view multiple USS files simultaneously.
- **Rename USS files:** You can rename specified USS files.
- **Download, edit, and upload existing USS files:** You can instantly pull USS files from the mainframe, edit them and upload back.
- **Create and delete USS files and directories:** Enables you to easily create and delete both USS files and directories.
- **View and access multiple profiles simultaneously:** Enables you to work with USS files from multiple profiles.

View Unix System Services (USS) files

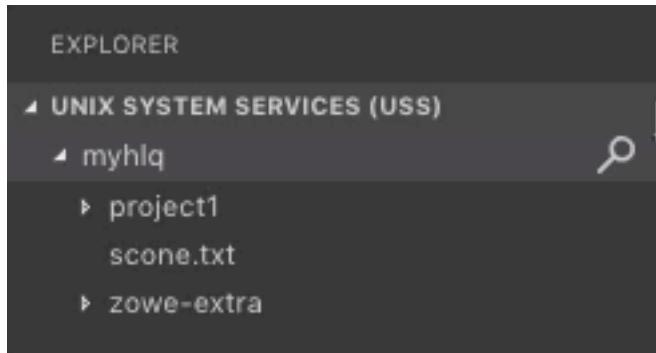
1. Navigate to the explorer tree.
2. Open the **Unix System Services (USS)** bar.
3. Select the profile that you want to filter.
4. Click the **Search Unix System Services (USS) by Entering a Path** magnifying glass.
5. From the drop-down, enter the path that you want as the root of your displayed tree. All child files and directories of that root file are displayed in the explorer tree.

Note: You will not be able to expand directories or files that you are not authorised for.



Refresh the list of files

1. Navigate to the explorer tree.
2. Click **Refresh All** button on the right of the **Unix System Services (USS)** explorer bar as illustrated by the following screen:



Rename USS files

1. Navigate to the explorer tree.
2. Open the **USS** bar.
3. Select a USS file you want to rename.

4. Right-click the USS file and select the **Rename USS file** option.
5. Change the name of the USS file.

Download, edit, and upload an existing file

1. Click the file that you want to download.

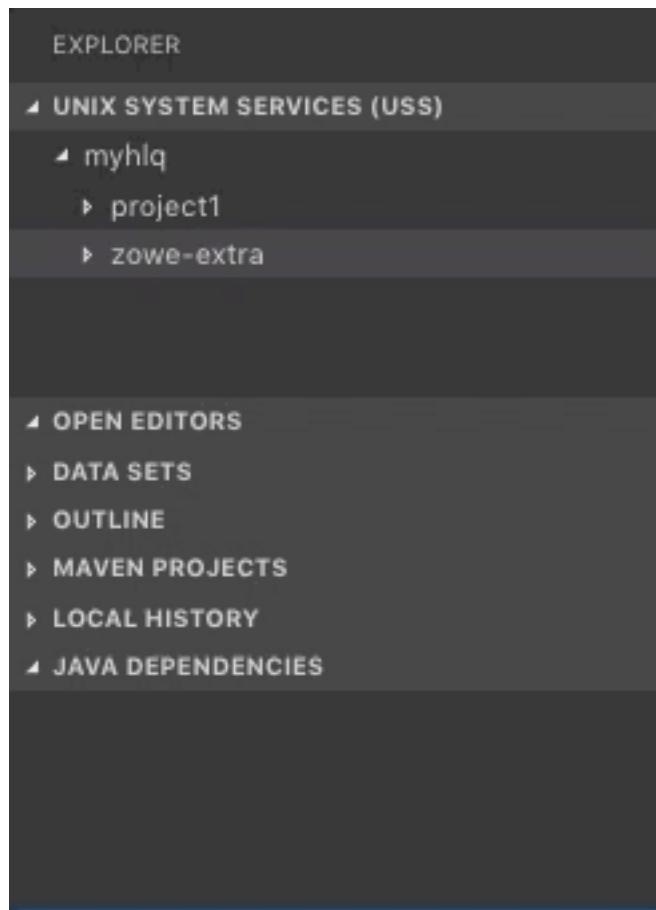
Note: To view the files within a directory, click the directory to expand the tree.

The file is displayed in the text editor window of VSC.

Note: If you define file associations with syntax coloring, the suffix of your file will be marked up.

2. Edit the document.
3. Press Ctrl+S or Command+S (OSx) to save the file

Your file is uploaded.



Creating and deleting files and directories

Create a directory

1. Navigate to the explorer tree.
2. Open the **Unix System Services (USS)** bar.
3. Select a directory where you want to add the new directory.
4. Select the **Create directory** button and specify the directory name. The directory is created.

Create a file

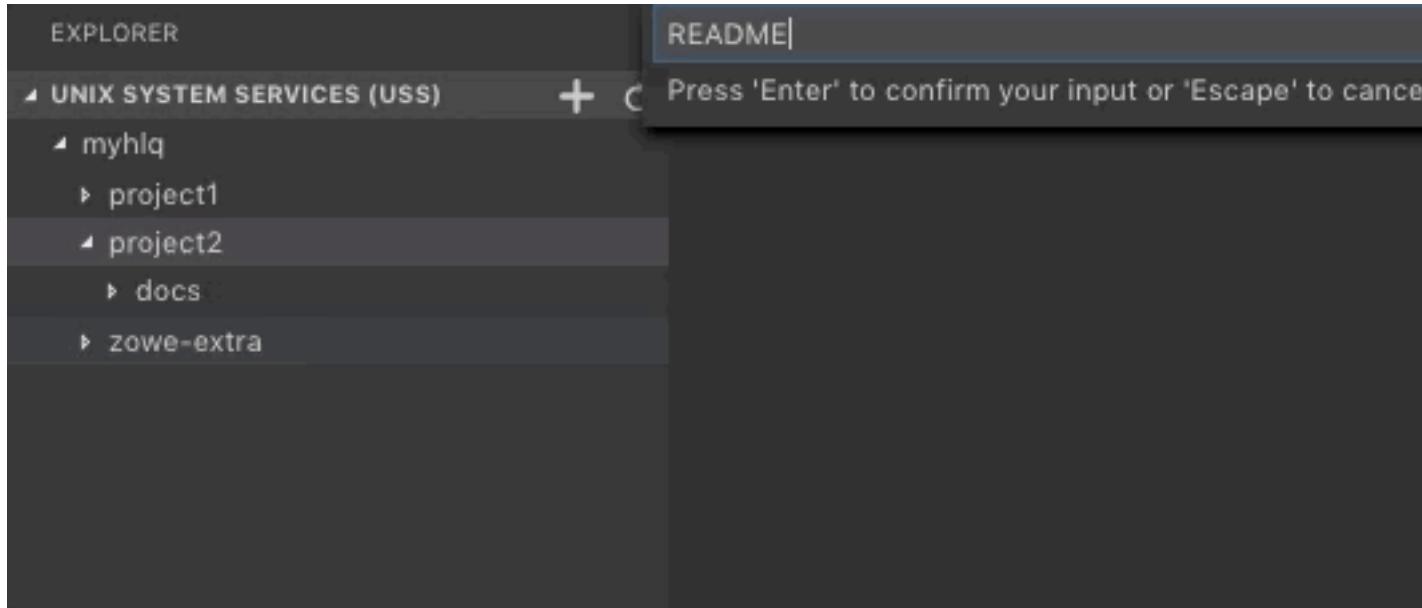
1. Navigate to the explorer tree.
2. Open the **Unix System Services (USS)** bar.
3. Select a directory where you want to add the new file to.
4. Select the **Create file** button and specify the file name. The file is created.

Delete a file

1. Navigate to the explorer tree.
2. Open the **Unix System Services (USS)** bar.
3. Select a file you want to remove.
4. Select the **Delete** button and click **Yes*** to confirm. The file is deleted.

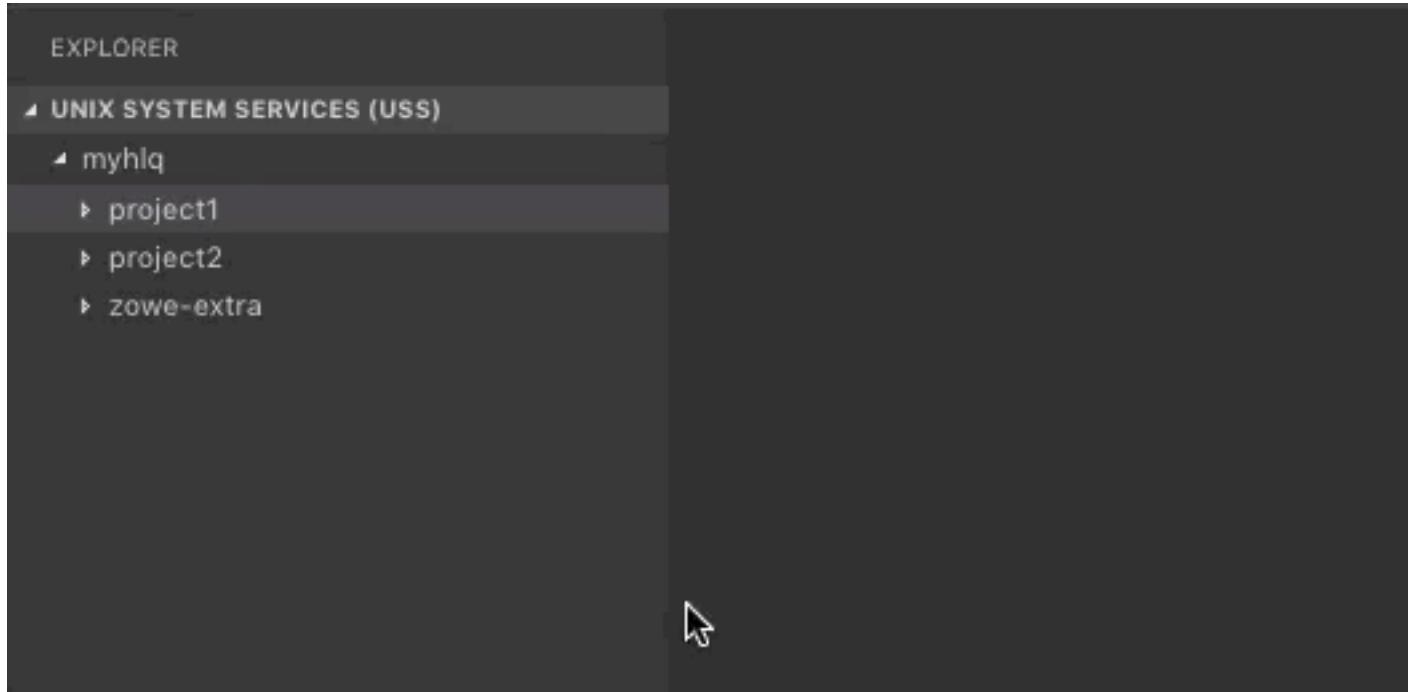
Delete a directory

1. Navigate to the explorer tree.
2. Open the **Unix System Services (USS)** bar.
3. Select a directory you want to remove.
4. Select the **Delete** button and click **Yes** to confirm. The directory and all child files and directories are deleted.



View and access multiple USS profiles simultaneously

1. Navigate to the explorer tree.
2. Open the **Unix System Services (USS)** bar.
3. Click the **Add Session** button on the right of the **Unix System Services (USS)** explorer bar.
4. Select the profile that you want to add to the view as illustrated by the following screen.



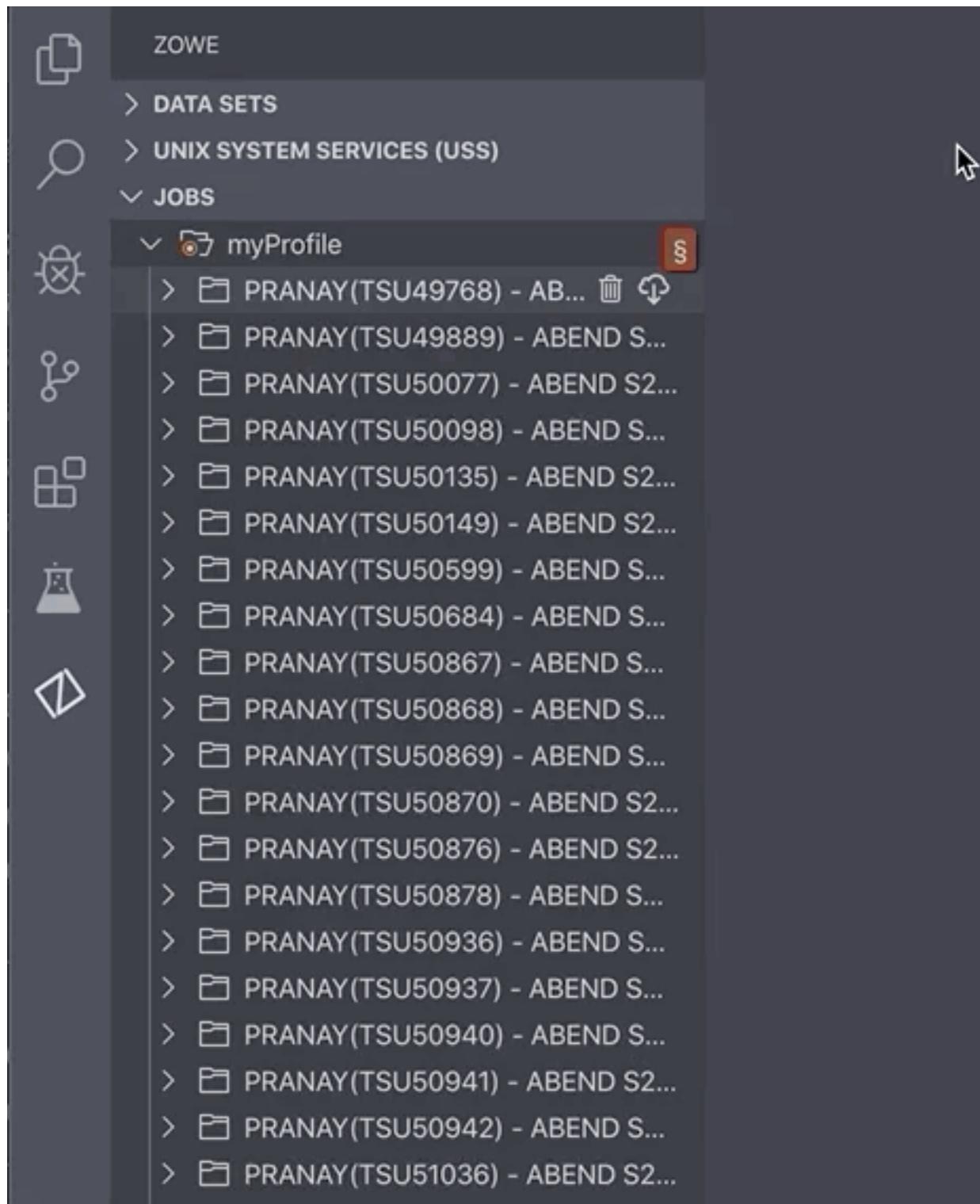
Work with jobs

You can use the following functionalities when interacting with jobs:

- **View a job:** You can view multiple jobs simultaneously.
- **Download spool content:** You can download spool content on your computer.

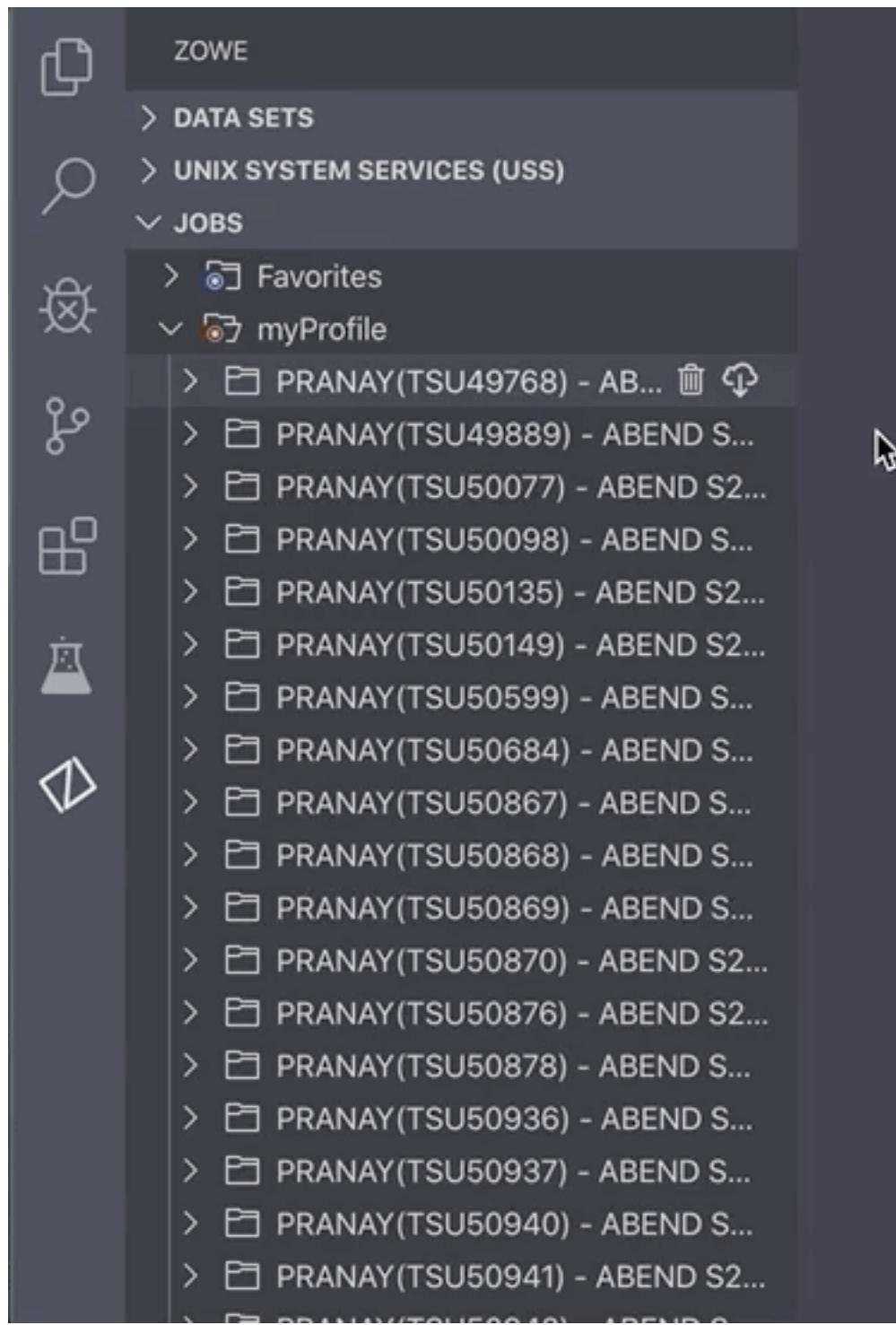
View a job

1. Navigate to the explorer tree.
2. Open the **JOBS** bar.
3. Select a directory with JCL files.
4. Right-click on the JCL you want to view, and click **Get JCL**.



Download spool content

1. Navigate to the explorer tree.
2. Open the **JOBS** bar.
3. Select a directory with JCL files.
4. Click the **Download** icon next to a folder with the spool content.
5. Save the file on your computer.

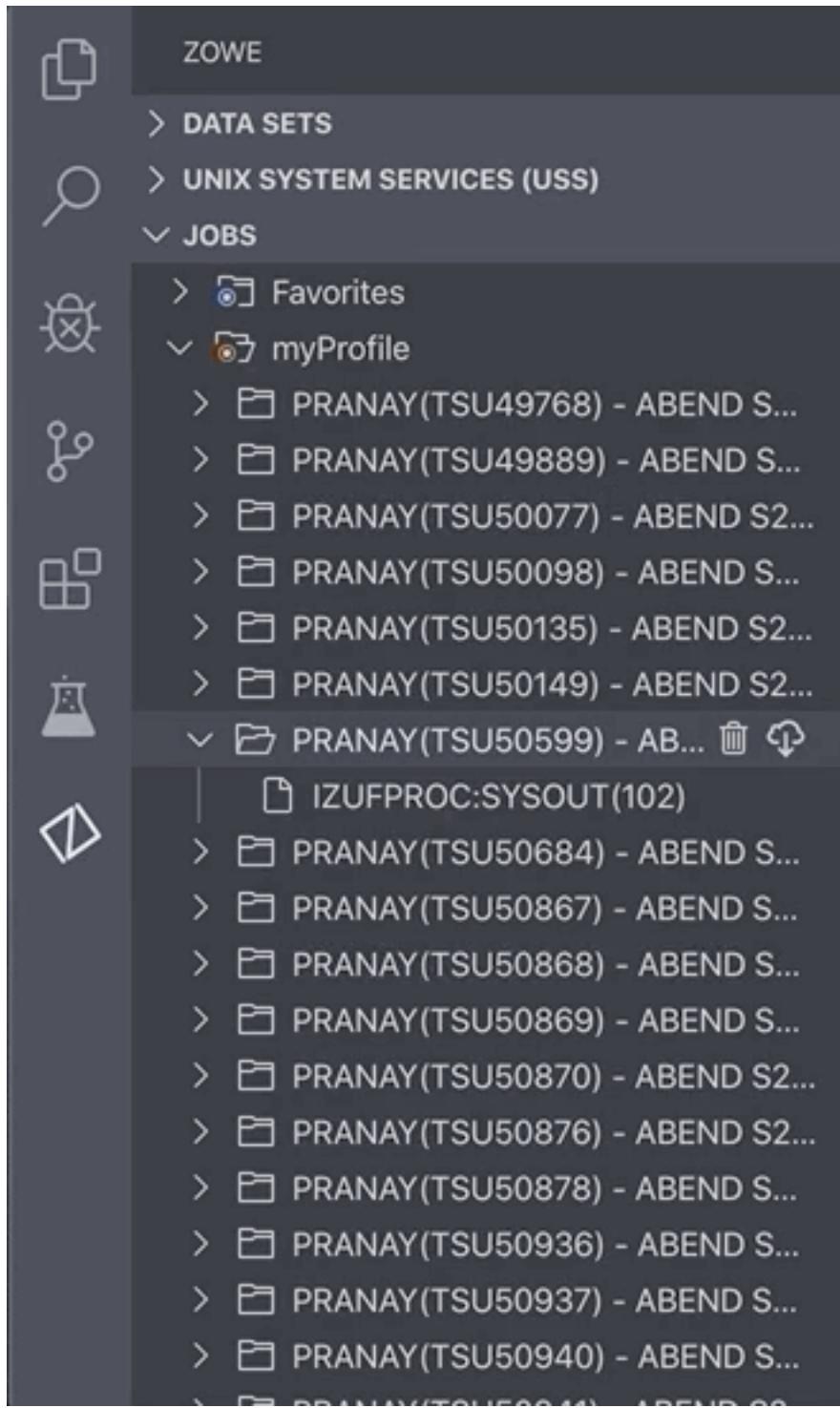


MVS/TSO Commands

Issue MVS commands

Zowe Explorer also enables you to issue MVS commands. You can issue such commands as Allocate or Exec against a profile.

1. Press the **F1** key on your keyboard.
2. Select the **Zowe:Issue MVS Command** option.
3. Select your profile.
4. Issue an MVS command.



Extending Zowe Explorer

You can extend the possibilities of Zowe Explorer by creating your own extensions. For more information on how to create your own Zowe Explorer extension, see [Extensions for Zowe Explorer](#).

Chapter

3

Extending

Topics:

- Overview
- Developing for Zowe CLI
- Developing for Zowe API
Mediation Layer
- Developing for Zowe Application
Framework
- Developing for Zowe SDKs
- Zowe Conformance Program

Overview

Extending Zowe

Zowe is designed as an extensible tools platform. One of the Zowe architecture goals is to provide consistent interoperability between all Zowe components including extensions. The Zowe Conformance Program defines the criteria to help accomplish the aforementioned goal. By satisfying the Zowe Conformance Program criteria, extension providers are assured that their software remains functional throughout the Zowe release cycle. For more information, see the [Zowe Conformance Program](#) on page 397.

Zowe can be extended in the following ways:

- [Extend Zowe CLI](#) on page 280
- [Extend Zowe API Mediation Layer](#) on page 280
 - [Dynamic API registration](#) on page 280
 - [Static API registration](#) on page 280
- [Add a plug-in to the Zowe Desktop](#) on page 281
- [Sample extensions](#) on page 281

Note: For more information on the architecture of Zowe, see [Zowe architecture](#) on page 13.

Extend Zowe CLI

Zowe CLI extenders can build plug-ins that provide new commands. Zowe CLI is built using Node.js and is typically run on a machine other than z/OS, such as a PC, where the CLI can be driven through a terminal or command prompt, or on an automation machine such as a DevOps pipeline orchestrator.

For more information about extending the Zowe CLI, see [Developing a new plug-in](#) on page 296. This article includes a sample plug-in that is provided with the tutorial; see [Installing the sample plug-in](#) on page 291.

Extend Zowe API Mediation Layer

Zowe API Mediation Layer extenders can build and onboard additional API services to the API ML microservices ecosystem. REST APIs can register with the API Mediation Layer, which makes them available in the API Catalog and for routing through the API Gateway.

To register a z/OS service with the API Mediation Layer, there are two approaches:

- [Dynamic API registration](#) on page 280
- [Static API registration](#) on page 280

For information about how to onboard REST APIs, see the [Onboarding Overview](#) on page 302.

Dynamic API registration

Registration of a REST API service to the API ML is performed through a call to the Discovery Service by sending registration data and metadata for the service being registered. Registration requires that the z/OS service must know the web address of the API ML Discovery Service. When Dynamic registration is performed, the service that performs the registration must periodically send heartbeat requests to the Discovery Service for each registered service instance. These heartbeat requests serve to renew the corresponding service instance registration with API ML. These requests enable the Discovery Service to monitor the availability of registered service instances. Services that are registered dynamically display the status of the service in the API Catalog after initial service registration.

For more information about how to build a service which is able to register, see the [Onboarding Overview](#) on page 302.

Static API registration

For services that cannot be modified to be dynamically discoverable, it is possible onboard them to the API ML by providing the API ML a static definition file with API service details. This registration method does not require modifications to the existing API service code. For more information, see [Onboard a REST API without code changes](#)

[required](#) on page 317. Unlike services that use Dynamic API registration, the status of services onboarded through Static API registration is not displayed in the API Catalog.

Add a plug-in to the Zowe Desktop

The Zowe Desktop allows a user to interact with z/OS applications through a web browser. The Desktop is served by the Zowe Application Framework Server on z/OS, also known as Z Lightweight User Experience (ZLUX). The Zowe desktop comes with a set of default applications. You can extend it to add new applications. For more information, see [Overview](#) on page 351.

The Zowe Desktop is an angular application that allows native plug-ins to be built that provide for a high level of interoperability with other desktop components. The React JavaScript toolkit is also supported. Additionally, you can include an existing web application in the Zowe Desktop using an iframe.

Notes: For more information, see the following samples:

- [Sample Iframe App](#) on page 352.
- [Sample Angular App](#) on page 352.
- [Sample React App](#) on page 352.

Sample extensions

Sample Zowe API and API Catalog extension

The repository <https://github.com/zowe/sample-node-api> contains a sample Zowe extension with a node server providing sample APIs for looking at cars in a dealership. For more information, see [sample-node-api](#).

Sample Zowe Desktop extension

The repository <https://github.com/zowe/sample-trial-app> contains a sample Zowe extension with a node server providing a web page that gives a user interface to the APIs included with the API sample above.

Packaging z/OS extensions

You can extend Zowe in multiple ways. You may extend Zowe with "microservices", which may start a new service within Zowe. You can also create Zowe App Framework plug-ins to provide UI to users.

Before you start, review the following terms first.

- **component:** is the most generic way to describe a program which can work within Zowe. It can be a microservice, a Zowe App Framework plug-in, or even just a shared program to be used by other Zowe components. This is also the generic word when referring to both Zowe core components and extensions. In most of the cases described in this topic, this terminology doesn't include programs running on the client side, like Zowe CLI plug-in or Zowe Explorer (VSCode extension).
- **extension:** is similar to **component** but excludes Zowe core components. It is recommended that you install all Zowe extensions into a shared extension directory.

Zowe server component package format

You can package Zowe components (extensions) into various formats. You can package them as a stand-alone PAX, ZIP, or TAR file. You can also bundle and ship your Zowe extension(s) within another product.

A typical component package, for example, `jobs-api-package-1.0.4.zip`, consists of these files and directories:

```

+-- manifest.yaml
-- apiml-static-registration.yaml.template
-- bin/
    |-- configure.sh
    |-- jobs-api-server-1.0.4-boot.jar
    |-- start.sh
    |-- validate.sh

```

where,

- `manifest.yaml`: This is the Zowe component manifest file. You can find detailed definition of manifest in [Zowe component manifest](#) on page 282.
- `apiml-static-registration.yaml.template`: This is a supporting file that instructs the Zowe launch script how to register this extension service to the API Mediation Layer Discovery service. In this case, this file is referred in the `manifest.yaml apimlServices.static[0].file` field. This file is optional depending on the function of the component and you can change and customize the file name in the manifest file.
- `bin/(configure|start|validate).sh`: These are Zowe component lifecycle scripts. You may not need these files depending on the function of the component. You can find detailed definition of lifecycle scripts in [Zowe lifecycle](#).

It is also suggested that you put the following files into the package:

- `README.md`: This file is a brief introduction to your extension in Markdown format, including how it should be installed, configured, verified, and so on.
- `LICENSE`: This is the full license text file.

If you decide to bundle and ship Zowe extensions within another product, you can put the whole directory structure above into your product package as subdirectories. Take the following structure as an example.

```
+-- <my-product-root>
    |-- <other-directories-and-files>
    |-- zowe-extension-A
        |-- manifest.yaml
        |-- bin/
            |-- start.sh
    |-- zowe-extension-B
        |-- manifest.yaml
        |-- bin/
            |-- start.sh
```

Zowe component manifest

Note: this feature is added with Zowe v1.19.0 release.

Zowe extensions, as well as core components, can use a manifest file to describe itself. The manifest file defines the name and purpose of the component. It also provides information about how this component should be installed, configured, started, and tested. It can be named as `manifest.yaml`, `manifest.yml`, or `manifest.json` and should locate in the root directory of the component. Currently, only YAML or JSON format is supported.

The manifest file contains the following properties:

- **name**: (Required) Defines a short, computer-readable name of the component. This component name will be used as directory name after it is installed. The allowed characters in the name are alphabets, numbers, hyphen (-) and underscore (_). For example, `explorer-jes` is a valid extension name.
- **id**: (Optional) Defines a long, computer-readable identifier of the component. If the component is hosted as one of the projects in [Open Mainframe Project](#), the identifier also matches the component path in the Zowe Artifactory. For example, `org.zowe.explorer-jes` is a valid identifier. You can locate the component's official releases by looking into the `libs-release-local/org/zowe/explorer-jes/` directory in the [Zowe Artifactory](#).
- **version**: (Optional but recommended) This is the current version of the component without the prefix of v. For example, `1.0.4` is a valid version value.
- **title**: (Optional) Defines a short human-readable name for this component. This value will also be used as the default title for API Catalog tile, or App Framework plug-in title. For example, `JES Explorer` is a valid title for the `explorer-jes` component.
- **description**: (Optional) Defines a long human-readable description of this component. There is no restriction on what you can put in the field.
- **license**: (Optional but recommended) Defines the license code of the component. For example, Zowe core components have `EPL-2.0` value in this field.

- **build:** (Optional but strongly recommended) Defines the build information of the current package, including git commit hash, and so on. When Zowe core components define manifest file, these fields are left as template variables. The template will be updated when a publishable package is created. It supports the following subfields:
 - **branch:** It tells the user which branch this package is built from.
 - **number:** You may create multiple packages in the same branch. This is the sequential number of the current package.
 - **commitHash:** This is the commit hash of the package that can be used to match the exact source code in the repository. Zowe core components usually use `git rev-parse --verify HEAD` to retrieve the commit hash.
 - **timestamp:** This is the UNIX timestamp when the package is created.
- **commands:** This defines actions that should be taken when the component is installed, configured, started, or tested. You must issue this command with one or more subfields as listed below. For example, `commands.install`. All subfields are optional and usually should point to a USS command or script.
 - **install:** This defines extra steps when installing this component. It will be automatically executed if you install your component with the `<RUNTIME_DIR>/bin/zowe-install-component.sh` utility tool.
 - **configureInstance:** This defines extra steps when configuring the component for a Zowe instance. It will be automatically executed if you configure your component with the `<RUNTIME_DIR>/bin/zowe-configure-component.sh` utility tool.
 - **validate:** This defines extra validations that the component requires other than global validations. It is for runtime purpose, and will be automatically executed each time Zowe is started.
 - **configure:** This defines extra configuration steps before starting the component. It is for runtime purpose, and will be automatically executed each time Zowe is started.
 - **start:** This tells the Zowe launch script how to start the component. It is for runtime purpose, and will be automatically executed each time Zowe is started.
- **apimlServices:** This section defines how the component will be registered to the API Mediation Layer Discovery Service. All subfields are optional.
 - **dynamic:** Array of objects. This information will tell Zowe and users what services you will register under the Discovery service.
 - **serviceId:** This defines the service ID registered to the Discovery service.
 - **static:** Array of objects. When the component is statically registered under the Discovery service, this tells Zowe where to find these static definitions. This information is for the Zowe runtime. When Zowe is started, the launch script will check this field and put the parse static definition file into the directory defined as `STATIC_DEF_CONFIG_DIR` in the Zowe instance.
 - **file:** Defines the path to the static definition file. This file is supposed to be a template.
- **appfwPlugins:** Array of objects. This section defines how the component will be registered to the App Framework plug-in. All subfields are optional.
 - **path:** This points to the directory where App Framework `pluginDefinition.json` file is located. If you use the `<RUNTIME_DIR>/bin/zowe-configure-component.sh` utility tool to configure this component for an instance, the script will automatically execute `<INSTANCE_DIR>/bin/install-app.sh` with this path.

Please note: All paths of directories or files mentioned above should be relative paths to the root directory where manifest is located.

Sample manifests

There are many examples you may find across Zowe GitHub repositories, for example:

- [API Catalog manifest.yaml](#)
- [Jobs API manifest.yaml](#)
- [Sample Node API and API Catalog extension manifest.yaml](#)
- [Sample Zowe App Framework extension manifest.yaml](#)

Install and configure Zowe server component

Learn how to install and configure the Zowe server components or extensions manually or by using scripts `zowe-install-component.sh` and `zowe-configure-component.sh`.

Install with `zowe-install-component.sh` (Technical Preview)

Note: This section is for technical preview and we are happy to hear any feedback. Content in this section may be changed or improved in the future.

Note: this feature is added with Zowe v1.19.0 release.

From Zowe v1.19.0, Zowe ships a `bin/zowe-install-component.sh` tool to help you install any Zowe server component (extension). Zowe core components are also installed with this tool. In order to be compatible with the tool, we recommend components follow [Zowe server component package format](#) on page 281.

The tool can be executed from z/OS USS, and it takes these command line parameters:

- **-o | --component-file:** (Required) Defines the path to the component package or directory.
- **-c | --component-name:** Specifies the name of the component. This parameter is optional if `NODE_HOME` is defined and the component has the manifest file. Otherwise, it's required.
- **-i | --instance-dir:** (Optional) Defines the path to the Zowe instance directory. When a value is defined, the script will also execute `bin/zowe-configure-component.sh` once it's installed.
- **-d | --target-dir:** (Optional) Defines the path to the installation target directory. For native components, the default value is `${ZOWE_ROOT_DIR} /components`. For non-native components, the script will check the value of the `ZWE_EXTENSION_DIR` variable. If the value is also empty, the script will fall back to the default target directory `/global/zowe/extensions`.
- **-e | --auto-encoding:** (Optional) Defines whether to automatically tag the encoding of the files that are shipped with the component. The default value is `auto`, which indicates that the script will determine whether the automatic tagging is needed or not. Note that the automatic tagging process is opinionated about which file extensions should be in which encoding. If this does not fit in your needs, a `pax` format is recommended to include the tagging information into your package. This option is only applicable for z/OS. Allowed values include:
 - `yes`: automatically tag the encoding of the files.
 - `no`: do not automatically tag encoding of the files.
 - `auto`: tag only when manifest is in ISO8859-1 encoding.
- **-k | --core:** This is an optional boolean. It defines whether this component is bundled into the Zowe package as a core component.
- **-l | --logs-dir:** (Optional) Specifies the path to the log directory.
- **-f | --log-file:** (Optional) Instead of writing independent log to a directory, you have option to append log to this log file specified.

Examples

This command installs `my-zowe-component-1.2.3.pax` into `/global/zowe/extensions`.

```
$ zowe-install-component.sh -o /path/to/my-zowe-component-1.2.3.pax
```

This command installs `my-zowe-component-1.2.3.zip` into `/var/zowe/my-extensions` and also configures this component for the Zowe instance located at `/var/zowe/instance`. The installation and configuration logs will be written into `/var/zowe/instance/logs`.

```
$ zowe-install-component.sh \
  -o /path/to/my-zowe-component-1.2.3.zip \
  -d /var/zowe/my-extensions \
  -i /var/zowe/instance \
  -l /var/zowe/instance/logs
```

Configure with `zowe-configure-component.sh` (Technical Preview)

Note: This section is for technical preview and we are happy to hear any feedback. Content in this section may be changed or improved in the future.

Note: This feature is added with Zowe v1.19.0 release.

From Zowe v1.19.0, Zowe ships a `bin/zowe-configure-components.sh` tool to help you configure an installed Zowe server component (extension) for a Zowe instance. Zowe core components are also configured with this tool. In order to be compatible with the tool, we recommend components follow [Zowe server component package format](#) on page 281.

You may not need to run this script directly if you have supplied `-i|--instance-dir` when you run `zowe-install-component.sh`.

The tool can be executed from z/OS USS, and it takes these command line parameters:

- `-c|--component-name`: (Required) Specifies the name of the component.
- `-i|--instance-dir`: (Required) Defines the path to the Zowe instance directory.
- `-d|--target-dir`: (Optional) Defines the directory where the component is installed. For native components, the default value is `${ZOWE_ROOT_DIR} /components`. For non-native components, the script will check `ZWE_EXTENSION_DIR` if possible. Otherwise, it will fall back to the default target directory `/global/zowe/extensions`.
- `-k|--core`: This is an optional Boolean. It defines whether this component is bundled into the Zowe package as a core component.
- `-l|--logs-dir`: (Optional) Defines the path to the log directory.
- `-f|--log-file`: (Optional) Instead of writing independent log to a directory, you have option to append log to this log file specified.

Examples

This command configures `my-zowe-component` installed in `/global/zowe/extensions` for the Zowe instance located at `/var/zowe/instance`.

```
$ zowe-configure-component.sh \
  -c my-zowe-component \
  -i /var/zowe/instance
```

This command configures `my-zowe-component` installed in `/var/zowe/my-extensions` for the Zowe instance located at `/var/zowe/instance`. The configuration logs will be written into `/var/zowe/instance/logs`.

```
$ zowe-configure-component.sh \
  -c my-zowe-component \
  -i /var/zowe/instance \
  -d /var/zowe/my-extensions \
  -l /var/zowe/instance/logs
```

Install and configure manually

Zowe core components

The Zowe runtime directory delivers its core components in the `<RUNTIME_DIR>/components/` directory. A typical components directory looks like this:

```
<RUNTIME_DIR>/components/
  /discovery
  /gateway
  /app-server
  /explorer-jes
  /explorer-mvs
  /files-api
```

```
/jobs-api
/...
```

You can configure whether to start the Zowe core component or not with the `LAUNCH_COMPONENTS` variable defined in `<INSTANCE_DIR>/instance.env`. The value of this variable can be a comma-separated list of core component names.

Note: You can also use full USS path to the component bin directory which contains lifecycle scripts, but this behavior will be deprecated in next major release.

Zowe extensions

It is suggested that you install all Zowe extension runtime programs into a single location. The suggested default extension directory is `/global/zowe/extensions`. Each extension should be represented with the extension name in this directory, either a directory or a symbolic link. This location should be defined as `ZWE_EXTENSION_DIR` in `<INSTANCE_DIR>/instance.env`.

The Zowe launch script reads `EXTERNAL_COMPONENTS` defined in `<INSTANCE_DIR>/instance.env` to decide whether to start an extension. The value of `EXTERNAL_COMPONENTS` is a comma-separated list of extension names.

Note: You can also use full USS path to the extension bin directory which contains lifecycle scripts, but this behavior will be deprecated in next major release.

Example

The vendor MYVENDOR has a product named MYAPP that installs into `/usr/lpp/myvendor/myapp`. There is one Zowe extension shipped within it in the directory `/usr/lpp/myvendor/myapp/zowe-ext`. This subdirectory is a Zowe extension so they want it to be started and stopped with Zowe and run as an address space under the `ZWESVSTC` started task in the Zowe USS shell.

The directory `/usr/lpp/myvendor/myapp/zowe-ext` should include a `manifest.yaml` file to describe the extension. The script `/usr/lpp/myvendor/myapp/zowe-ext/bin/validate.sh` checks that the environment has been configured correctly and the script `/usr/lpp/myvendor/myapp/zowe-ext/bin/start.sh` starts the vendor application. The `/usr/lpp/myvendor/myapp/zowe-ext/manifest.yaml` should look like this:

```
name: myapp
id: com.myvendor.myapp
title: My Zowe Extension
commands:
  validate: bin/validate.sh
  start: bin/start.sh
```

Because MYAPP is shipped within another product, the installation should create a symbolic link in `ZWE_EXTENSION_DIR` directory.

```
$ ls -l /global/zowe/extensions
total 16
lrwxrwxrwx  1 <USER> <GROUP>          23 Nov 11  2019 myapp -> /usr/lpp/
myvendor/myapp/zowe-ext
```

Also `myapp` will be added into the value of the `EXTERNAL_COMPONENTS` variable in `<INSTANCE_DIR>/instance.env`.

```
ZWE_EXTENSION_DIR=/global/zowe/extensions
EXTERNAL_COMPONENTS=some-other-extensions,myapp
```

You might need to manually run the script `<INSTANCE_DIR>/bin/install-app.sh` if your component is a Desktop plug-in. Or you can choose to add this step to [Configure](#) on page 288.

When the Zowe instance is launched by running <INSTANCE_DIR>/bin/zowe-start.sh, it will read manifest commands instructions and call the /usr/lpp/myvendor/myapp/zowe-ext/bin/start.sh script. The started task will create an address space under ZWESVSTC for the vendor component. When the Zowe instance is stopped, the address space is terminated.

Zowe server component runtime lifecycle

Zowe runtime lifecycle

This topic describes the runtime lifecycle of Zowe core components and how an offering that provides a Zowe extension can set up runtime lifecycle for their component.

The Zowe UNIX System Services (USS) components are run as part of the started task ZWESVSTC. For more information, see [Option 1: Starting Zowe from a USS shell](#) on page 156. There are two key USS directories that play different roles when launching Zowe.

- The Zowe runtime directory <RUNTIME_DIR> that contains the executable files is an immutable set of directories and files that are replaced each time a new release is applied. The initial release or an upgrade is installed either with UNIX shell scripts (see [Installing Zowe runtime from a convenience build](#) on page 102), or SMP/E where the runtime directory is laid down initially as FMID AZWE001 and then upgraded through rollup PTF builds (see [Installing Zowe SMP/E](#) on page 108). The Zowe runtime directory is not altered during operation of Zowe, so no data is written to it and no customization is performed on its contents.
- The Zowe instance directory <INSTANCE_DIR> contains information that is specific to a launch of Zowe. It contains configuration settings that determine how an instance of the Zowe server is started, such as ports that are used or paths to dependent Java and Node.js runtimes. The instance directory also contains log directory where different 'microservices' write trace data for diagnosis, as well as a workspace and shell scripts to start and stop Zowe. More than one Zowe instance directory can be created to allow multiple launches of a Zowe runtime, each one isolated from each other and starting Zowe depending on how the instance directory has been configured. For more information, see [Creating and configuring the Zowe instance directory](#) on page 150.

To start Zowe, the script <INSTANCE_DIR>/bin/zowe-start.sh is run from a USS shell. This uses a REXX program to launch the started task ZWESVSTC, passing the instance directory path as a parameter. It is the equivalent of using the TSO command /S ZWESVSTC, INSTANCE='<INSTANCE_DIR>', JOBNAME='<JOBNAME>'. The ZWESVSTC PROCLIB uses the program that creates a USS process and starts the script <INSTANCE_DIR>/bin/internal/run-zowe.sh. By using BPXBATSL to start the USS process, all of the address spaces started under this shell are managed by SDSF. If the zowe-start.sh run run-zowe.sh directly, the USS processes will not run as a started task and will run under the user ID of whoever ran the run-zowe.sh script rather than the Zowe user ID of ZWESVUSR, likely leading to permission errors accessing the contents of the <RUNTIME_DIR> as well as the Zowe certificate. For these reasons, the zowe-start.sh script launches Zowe's USS process beneath the started task ZWESVSTC.

When run-zowe.sh is run in the USS shell that BPXBATSL creates, it executes the file <INSTANCE_DIR>/instance.env. This file sets a number of shell variables, such as ROOT_DIR that points to the directory with the <RUNTIME_DIR>, variables for all of the ports used by the Zowe components, and other configuration data. For more information, see [Reviewing the instance.env file](#).

Note:

The scripts of core Zowe components and some extensions use the helper library <RUNTIME_DIR>/bin/utils. Currently, these are not publicly supported. Future releases of Zowe might provide these as supported system programming interfaces (SPIs) and include their usage in the Zowe documentation.

Zowe component runtime lifecycle

Each Zowe component will be installed with its own USS directory, which contains its executable files. Within each component's USS directory, a bin directory is recommended to contain scripts that are used for the lifecycle of the component. When Zowe is started, it identifies the components that are configured to launch and then execute the scripts of those components in the cycle of [Validate](#) on page 288, [Configure](#) on page 288, and [Start](#) on page 288. All components are validated, then all are configured, and finally all are started. This technique is used as follows:

- Used for the base Zowe components that are included with the core Zowe runtime.
- Applies to extensions to allow vendor offerings to be able to have the lifecycle of their 'microservices' within the Zowe USS shell and be included as address spaces under the ZWESVSTC started task.

Validate

Each component can optionally instruct Zowe runtime to validate itself with a USS command defined in manifest commands .validate. If this is not defined, for backward compatible purpose, a call to its /bin/validate.sh script will be executed if it exists.

If present, the validate script performs tasks such as:

- Check that the shell has the correct prerequisites.
- Validate that ports are available.
- Perform other steps to ensure that the component is able to be launched successfully.

During execution of the validate script, if an error is detected, then a component should echo a message that contains information to assist a user diagnosing the problem.

Configure

Each component can optionally instruct Zowe runtime to configure itself with a USS command defined in manifest commands .configure. If this is not defined, for backward compatible purpose, a call to its /bin/configure.sh script will be executed if it exists.

If the component has manifest defined, some configure actions will be performed automatically based on manifest definition:

- apimlServices.static: Zowe runtime will automatically parse and add your static definition to API Mediation Layer.

For backward compatible purpose, you can choose to configure component by yourself with /bin/configure.sh. An example configuration step is if a component wants to install applications into the Zowe desktop as iframes, or add API endpoints statically into the API Mediation Layer. Because a component's configure.sh script is run inside the USS shell that the instance.env has initialized, it will have all of the shell variables for prerequisites set, so the configure step can be used to query these in order to prepare the component ready for launch.

Start

Each component can optionally instruct Zowe runtime to start itself with a USS command defined in manifest commands .start. If this is not defined, for backward compatible purpose, a call to its /bin/start.sh script will be executed if it exists. If your component is not supposed to be started by itself, for example, the component is a shared library, you can skip this instruction.

It is up to each component to start itself based on how it has been written. We recommend that any variables that someone who configure Zowe may need to vary, such as timeout values, port numbers, or similar, are specified as variables in the instance.env file and then referenced as shell variables in the start.sh script to be passed into the component runtime.

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 290](#) - Clone the project and prepare your local environment.
2. [Installing the sample plug-in on page 291](#) - Install the sample plug-in to Zowe CLI and run as-is.
3. [Extending a plug-in on page 293](#) - Extend the sample plug-in with a new by creating a programmatic API, definition, and handler.
4. [Developing a new plug-in on page 296](#) - 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 301](#) - 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 |

| For more information about ... | See: |
|---|--|
| 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.

Prerequisites

[Methods to install Zowe CLI](#) on page 173.

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 `@latest` or `@zowe-v1-lts` 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 `master` branch of Sample Plug-in is compatible with the `@zowe-v1-lts` version of core CLI (Zowe LTS release).

The `master` branch of Sample Plug-in is also compatible with the `@latest` version of core CLI (Zowe Active Development release) at this time.

For more information about the versioning scheme, see [Maintainer 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 master
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 291 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 290 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:

```
$ zowe

DESCRIPTION
-----
Welcome to Zowe CLI!

Zowe CLI is a command line interface (CLI) that provides a simple and
streamlined way to interact with IBM z/OS.

For additional Zowe CLI documentation, visit https://zowe.github.io/zowe-cli-docs/

For Zowe CLI support, visit https://zowe.org.
```



```
USAGE
-----
zowe [group]

GROUPS
-----
diagnostics          Run diagnostics
plugins              Install and manage plug-ins
profiles             Create and manage configuration profiles
provisioning | pv    Perform z/OSMF provisioning tasks on P
                      Templates in the Service Catalog and P
                      Instances in the Service Registry.
zos-console | console Issue z/OS console commands and collect
zos-files | files    Manage z/OS data sets
zos-jobs | jobs      Manage z/OS jobs
zos-tso | tso         Issue TSO commands and interact with T
zosmf                Interact with z/OSMF
zowe-cli-sample | zcsp Zowe CLI sample plug-in
```

Figure 3: Installed Sample Plugin

Using the installed plug-in

To use the plug-in functionality, issue: `zowe zowe-cli-sample list directory-contents`:

```
$ zowe zowe-cli-sample list directory-contents
We just got a valid z/OSMF status response from system = [REDACTED]

mode size birthed
16822      Thu Sep 20 2018 09:52:20 GMT-0400 (Eastern Daylight
33206 297    Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight
16822      Thu Sep 20 2018 09:54:20 GMT-0400 (Eastern Daylight
33206      Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight
33206 211    Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight
33206 6855   Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight
33206 1609   Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight
16822      Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight
16822      Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight
33206 36028  Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight
16822      Thu Sep 20 2018 10:06:27 GMT-0400 (Eastern Daylight
33206 14100  Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight
[REDACTED]
```

Figure 4: 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 293 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 291 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 `@zowe/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
  "@zowe/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 `@zowe/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 "@zowe/zowe-cli-sample-plugin";
import { Session, Imperative } from "@zowe/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 "@zowe/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
    "@zowe/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 `@zowe/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:

```
$ zowe zowe-cli-sample list typicode-todos --id 4
userId:      1
id:          4
title:       et porro tempora
completed:   true
```

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 296 tutorial next to create a new plug-in for Zowe CLI.

Developing a new plug-in

Before you begin this tutorial, complete the [Extending a plug-in](#) on page 293 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.

Files changed (1) [show](#)

| | | Old → New RENAME |
|---|---|---|
| | | @@ -1,2 +1,2 @@ |
| 1 | 1 | //SWAWI03\$ JOB 105300000 |
| 2 | - | //EXEC EXEC PGM=IEFBR14 |
| | 2 | + //EXEC EXEC PGM=IEFBR15 |

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": "@zowe/files-util",
```

Issue the command `npm install` against the local repository.

Adjusting Imperative CLI Framework configuration

Change `imperative.ts` to contain the following:

```
import { IImparativeConfig } from "@zowe/imperative";

const config: IImparativeConfig = {
  commandModuleGlobs: ["**/cli/**/*definition!(.d).*s"],
  rootCommandDescription: "Files utility 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 "@zowe/imperative";
import { Download, IDownloadOptions, IZosFilesResponse } from "@zowe/cli";
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 "@zowe/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 "@zowe/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
    "@zowe/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 "████████.cntl(iefbr14)" "████████.cntl(iefbr15)"
```

The raw diff output is displayed as a command response:

```

$ zowe files-util diff data-sets "████████.cntl(iefbr14)" "████████.cntl(iefbr15)"
=====
--- ████████.cntl(iefbr14)          Old
+++ ████████.cntl(iefbr15)          New
@@ -1,2 +1,2 @@
 // $ 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 297.

- `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 301 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 289 and [Imperative CLI Framework Documentation](#) on page 289 to learn more.

Developing for Zowe API Mediation Layer

Onboarding Overview

As an API developer, you can onboard a REST API service to the Zowe™ API Mediation Layer (API ML). Onboarding your REST service to the Zowe™ API Mediation Layer will make your service discoverable by the API ML Discovery Service, enable routing through the API Gateway, and make service information and API documentation available through the API Catalog.

The specific method you use to onboard a REST API to the API ML depends on the programming language or framework used to build your REST service.

This Onboarding Overview article addresses the following topics:

- [Prerequisites](#) on page 302
- [Service Onboarding Guides](#) on page 303 to onboard your REST service with the API ML
- [Validating successful onboarding](#)
- Using the [Sample REST API Service](#) on page 305 to learn how to onboard a REST service to the API ML

Prerequisites

Meet the following prerequisites before you onboard your service:

- Running instance of Zowe

Note: For [Onboard a REST API without code changes required](#) on page 317, access to Zowe runtime is required to create the static service definition.

- A certificate that is trusted by Zowe and certificate(s) to trust Zowe services

Zowe uses secured communication over TLSv1.2. As such, the protocol version and the certificate is required. For more information, see [Certificate management in Zowe API Mediation Layer](#) on page 342 and [Zowe API ML TLS requirements](#).

- A REST API-enabled service that you want to onboard

If you do not have a specific REST API service, you can use the [sample service](#).

Your service should be documented in a valid OpenApi 2.0 / 3.0 Swagger JSON format.

- Access to the Zowe artifactory

Repository URL: <https://zowe.jfrog.io/zowe/libs-release>

- Either the *Gradle* or *Maven* build automation system

Service Onboarding Guides

Services can be updated to support the API Mediation Layer natively by updating the service code. Use one of the following guides to onboard your REST service to the Zowe API Mediation Layer:

Recommended guides for services using Java

- [Onboard a REST API service with the Plain Java Enabler \(PJE\)](#)
- [Onboard a Spring Boot based REST API Service](#)

Guides for Static Onboarding and Direct Call Onboarding

Use one of the following guides if your service is not built with Java, or you do not want to change your codebase or use the previously mentioned libraries:

- [Onboard a REST API without code changes required](#) on page 317
- [Onboarding a service with the Zowe API Meditation Layer without an onboarding enabler](#) on page 311

Documentation for legacy enablers

For legacy enabler documentation (version 1.2 and lower), refer to the previous version of the documentation:

- [Zowe Docs version 1.8.x](#)

Note: Enabler version 1.2 and previous versions are no longer supported.

Tip: We recommend you use the enabler version 1.3 or higher to onboard your REST API service to the Zowe API Medaition Layer.

Verify successful onboarding to the API ML

Verifying that your service was successfully onboroaded to the API ML can be done by ensuring service registration in the API ML Discovery Service or visibility of the service in the API ML Catalog.

Verifying service discovery through Discovery Service

Verify that your service is discovered by the Discovery Service with the following procedure.

Follow these steps:

- Issue a HTTP GET request to the Discovery Service endpoint /eureka/apps to get service instance information:

```
https://{{zowe-hostname}}:{{discovery-service-port}}/eureka/apps/{{serviceId}}
```

Note: The endpoint is protected by client certificate verification. A valid trusted certificate must be provided with the HTTP GET request.

- 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>
```

Tips:

- Ensure that addresses and user credentials for individual API ML components correspond to your target runtime environment.
- If you work with local installation of API ML and you use 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.

Verifying service discovery through the API Catalog

Services may not be immediately visible in the API Catalog. We recommend you wait for 2 minutes as it may take a moment for your service to be visible in the Catalog. If your service still does not appear in the Catalog, ensure that your configuration settings are correct.

Follow these steps:

1. Check to see that your API service is displayed in the API Catalog UI, and that all information including API documentation is correct.
2. Ensure that you can access your API service endpoints through the Gateway.

Sample REST API Service

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 317 (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": "pet"
    }
  ]
}
```

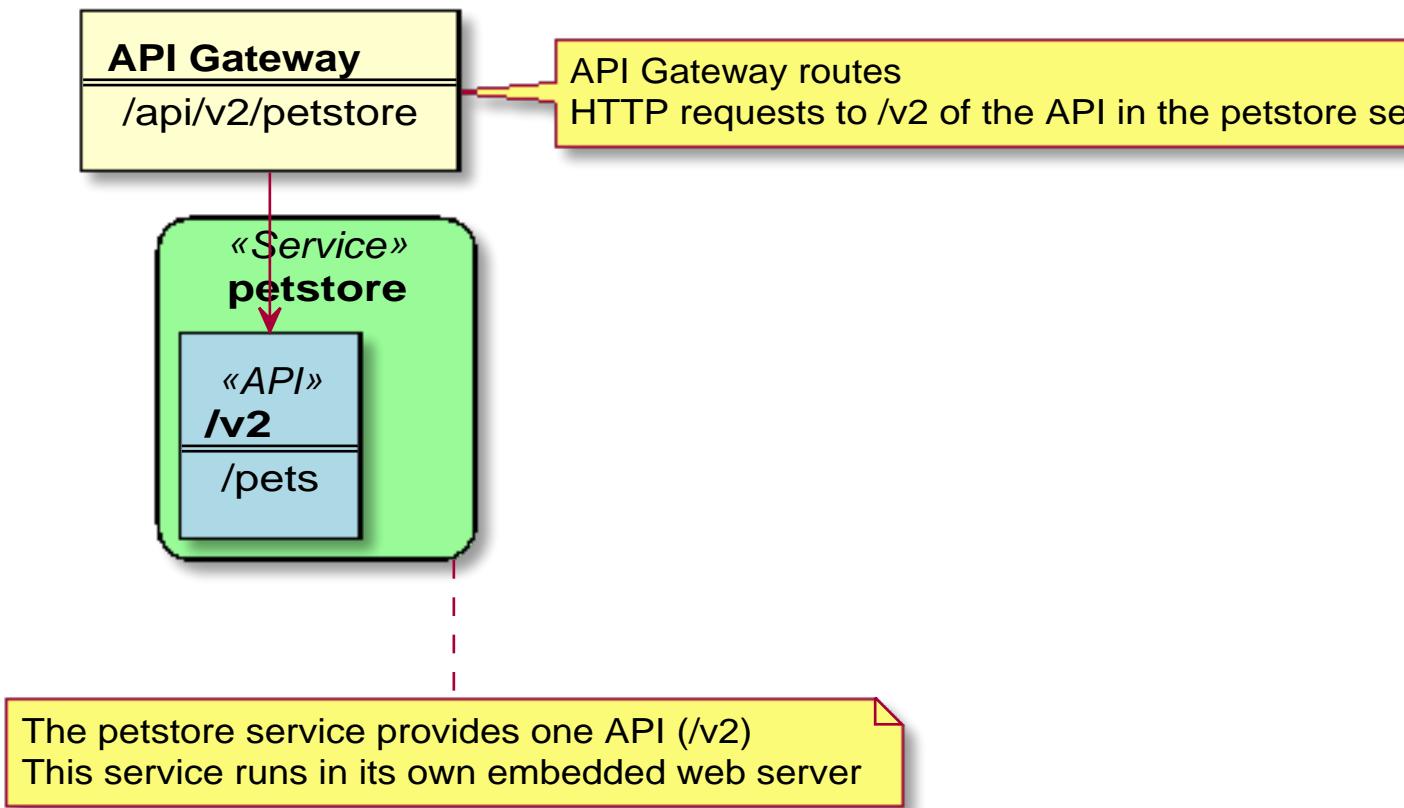
```

        "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.

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 to the API ML. You can also externalize configuration parameters for subsequent customization by a systems administrator.

- [Introduction on page 307](#)
- [Configuring a REST service for API ML onboarding on page 307](#)

- Plain Java Enabler service onboarding API on page 308
 - Automatic initialization of the onboarding configuration by a single method call on page 308
- Validating successful onboarding with the API Mediation Layer
- Loading YAML configuration files on page 309
 - Loading a single YAML configuration file on page 309
 - Loading and merging two YAML configuration files on page 309

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:

- Onboarding a service with the Zowe API Meditation Layer without an onboarding enabler on page 311
- 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.

The 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.

Validating successful onboarding with the API Mediation Layer

Ensure that you successfully onboarded a service with the API Mediation Layer.

Follow these steps:

1. [Validate successful onboarding](#)
2. Check that you can access your API service endpoints through the Gateway.
3. (Optional) Check that you can access your API service endpoints directly outside of the Gateway.

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, attempts to unregister this
        service from API ML registry.
    */
    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        if (apiMediationClient != null) {
            apiMediationClient.unregister();
        }
    }
}

```

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 302
- [python-eureka-client](#)
- [eureka-js-client](#)
- [Sample REST API Service](#) on page 305

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 311
- [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 useHttps. ForHttps, 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 useHttps.

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.enableUrlEncodedCharacters>false</
apiml.enableUrlEncodedCharacters>
    <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.authentication.scheme>httpBasicPassTicket</
apiml.authentication.scheme>
    <apiml.authentication.applid>ZOWEAPPL</apiml.authentication.applid>
    <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](#)

- [Authentication 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.

- **apiml.enableUrlEncodedCharacters**

When this parameter is set to `true`, the Gateway allows encoded characters to be part of URL requests redirected through the Gateway. The default setting of `false` is the recommended setting. Change this setting to `true` only if you expect certain encoded characters in your application's requests.

Important! When the expected encoded character is an encoded slash or backslash (%2F, %5C), make sure the Gateway is also configured to allow encoded slashes. For more info see [z/OS Installation Roadmap](#) on page 99.

- **apiml.connectTimeout**

The value in milliseconds that specifies a period in which API ML should establish a single, non-managed connection with this service. If omitted, the default value specified in the API ML Gateway service configuration is used.

- **apiml.readTimeout**

The value in milliseconds that specifies maximum time of inactivity between two packets in response from this service to API ML. If omitted, the default value specified in the API ML Gateway service configuration is used.

- **apiml.connectionManagerTimeout**

HttpClient employs a special entity to manage access to HTTP connections called by HTTP connection manager. The purpose of an HTTP connection manager is to serve as a factory for new HTTP connections, to manage the life cycle of persistent connections, and to synchronize access to persistent connections. Internally, an HTTP connection manager works with managed connections, which serve as proxies for real connections. ConnectionManagerTimeout specifies a period in which managed connections with API ML should be established. The value is in milliseconds. If omitted, the default value specified in the API ML Gateway service configuration is used.

- **apiml.okToRetryOnAllOperations**

Specifies whether all operations can be retried for this service. The default value is `false`. The `false` value allows retries for only GET requests if a response code of 503 is returned. Setting this value to `true` enables retry requests for all methods, which return a 503 response code. Enabling retry can impact server resources resulting from buffering of the request body.

- **apiml.service.corsEnabled**

When this parameter is set to `true`, CORS is enabled on the service level for all service routes. The same parameter can also be set on the service level, by providing the parameter as `customMetadata` as shown in the [custom metadata.md](#).

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](#).

Authentication parameters

Authentication parameters are grouped under the prefix: `apiml.authentication`. When unspecified, the default values are used.

This parameter enables a service to accept the Zowe JWT token. The API Gateway translates the token to an authentication method supported by a service.

The following parameters define the service authentication method:

- **apiml.authentication.scheme**

This parameter specifies a service authentication scheme. The following schemes are supported by the API Gateway:

- **bypass**

This value specifies that the token is passed unchanged to the service.

Note: This is the default scheme when no authentication parameters are specified.

- **zoweJwt**

This value specifies that a service accepts the Zowe JWT token. No additional processing is done by the API Gateway.

- **httpBasicPassTicket**

This value specifies that a service accepts PassTickets in the Authorization header of the HTTP requests using the basic authentication scheme. It is necessary to provide a service APPLID in the `apiml.authentication.applid` parameter.

Tip: For more information, see [Enabling PassTicket creation for API Services that Accept PassTickets](#) on page 348.

- **zosmf**

This value specifies that a service accepts z/OSMF LTPA (Lightweight Third-Party Authentication). This scheme should only be used for a z/OSMF service used by the API Gateway Authentication Service, and other z/OSMF services that are using the same LTPA key.

Tip: For more information about z/OSMF Single Sign-on, see [Establishing a single sign-on environment](#).

- **apiml.authentication.applid**

This parameter specifies a service APPLID. This parameter is valid only for the `httpBasicPassTicket` authentication scheme.

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 orHttps 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.

- **apiml.apiInfo.{api-index}.defaultApi**

(Optional) This parameter specifies if the API is the default one shown in the API Catalog. If no API has this parameter set to `true`, or multiple APIs have it set to `true`, then the default API becomes the API with the highest major version seen in `apiml.apiInfo.{api-index}.version`.

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 Mediation 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 Mediation Layer

Ensure that you successfully onboarded a service with the API Mediation Layer.

Follow these steps:

1. [Validate successful onboarding](#)
2. Check that you can access your API service endpoints through the Gateway.
3. (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 REST API without code changes required

As a user of Zowe™, onboard an existing REST API service to the Zowe™ API Mediation Layer without changing the code of the API service. This form of onboarding is also referred to 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 Mediation Layer. For a complete list of methods to onboard a REST service natively to the API Mediation Layer, see the [Service Onboarding Guides](#) on page 303.

The following procedure outlines the steps to onboard an API service through the API Gateway in the API Mediation Layer without requiring code changes.

- [Identify the API that you want to expose](#)
- [Define your service and API in YAML format](#) on page 318
- [Route your API](#) on page 319
- [Customize configuration parameters](#) on page 320
- [Add and validate the definition in the API Mediation Layer running on your machine](#) on page 326
- [Add a definition in the API Mediation Layer in the Zowe runtime](#) on page 326
- [\(Optional\) Check the log of the API Mediation Layer](#) on page 327
- [\(Optional\) Reload the services definition after the update when the API Mediation Layer is already started](#) on page 327

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 302.

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 the base URL of the service.

Example:

In the sample service described in the [Sample REST API Service](#) on page 305, the URL of the service is:
`http://localhost:8080`.

2. Identify the API of the service that you want to expose through the API Gateway.

Example:

The API provided by the sample service is a second version of the Pet Store API. All the endpoints to be onboarded are available through `http://localhost:8080/v2/` URL. This REST API is therefore available at the path `/v2` relative to base URL of the service. There is no version 1 in this case.

3. Choose the `service ID` of your service. The `service ID` identifies the service uniquely 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 referred to as the gateway URL and is composed of the API type and the major version. The usually used types are: `api`, `ui` and `ws` but you can use any valid URL element you want.

Example:

In the sample service, we provide a REST API. The first segment is `/api` as the service provides only one REST API. To indicate that this is version 2, the second segment is `/v2`. This version is required by the Gateway. If your service does not have a version, use `v1` on the Gateway.

Define your service and API in YAML format

After you identify the APIs you want to expose, you need to define your service and API in YAML format 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. This configuration is the minimal configuration necessary for the Gateway to properly route the requests to the application and to show the Service in the Catalog UI.

Note: For more details about configuration, see [Customize configuration parameters](#) on page 320.

```

services:
  - serviceId: petstore
    catalogUiTileId: static
    instanceBaseUrls:
      - http://localhost:8080
    routes:
      - gatewayUrl: api/v2
        serviceRelativeUrl: /v2
    authentication:
      scheme: httpBasicPassTicket
      applid: ZOWEAPPL
    apiInfo:
      - apiId: io.swagger.petstore
        gatewayUrl: api/v2

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 id is used as a part of the request URL towards the Gateway. It is removed by the Gateway when forwarding the request to the service.
- The service can have one or more instances. In this case, only one instance `http://localhost:8080` is used.
- One API is provided and the requests with the relative base path `api/v2` at the API Gateway (full gateway URL: `https://gateway:port/api/v2/serviceId/...`) are routed to the relative base path `/v2` at the full URL of the service (`http://localhost:8080/v2/...`).
- The file on USS should be encoded in ASCII to be read correctly by the API Mediation Layer.

Tips:

- There are more examples of API definitions at this [link](#).
- For more details about how to use YAML format, see this [link](#).

Route 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. The configuration parameters are explained in [Customize configuration parameters](#) on page 320. Gateway URL format:

```
https://{{gatewayHost}}:{{port}}/api/v{{majorVersion}}/{{serviceId}}/{{resource}}
```

Note: The API Gateway differentiates major versions of an API.

Example:

When the configuration parameters are:

```
services:
  serviceId: petstore
  instanceBaseUrls:
    - https://localhost:8080
  routes:
    gatewayUrl: api/v2
    serviceRelativeUrl: /v2
```

To access API version 2 of the service petstore, gateway URL will be:

```
https://gateway-host:port/api/v2/petstore
```

It will be routed to:

```
https://localhost:8080/v2
```

To access resource pets of the petstore version 2 API, gateway URL will be:

```
https://gateway:port/api/v2/petstore/pets
```

It will be routed to:

```
https://localhost:8080/v2/pets
```

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.

Customize configuration parameters

This part contains a more complex example of the configuration and an explanation of all the possible parameters:

```
services:
  - serviceId: petstore
    catalogUiTileId: static
    title: Petstore Sample Service
    description: This is a sample server Petstore service
    instanceBaseUrls:
      - http://localhost:8080
    homePageRelativeUrl: /home # Normally used for informational purposes
    for other services to use it as a landing page
      statusPageRelativeUrl: /application/info # Appended to the
    instanceBaseUrl
      healthCheckRelativeUrl: /application/health # Appended to the
    instanceBaseUrl
    routes:
      - gatewayUrl: api/v2
        serviceRelativeUrl: /v2
    authentication:
      scheme: httpBasicPassTicket
      appId: ZOWEAPPL
    apiInfo:
      - apiId: io.swagger.petstore
        gatewayUrl: api/v2
        swaggerUrl: http://localhost:8080/v2/swagger.json
        documentationUrl: https://petstore.swagger.io/
        version: 2.0.0
        defaultApi: true
    customMetadata:
```

```

yourqualifier:
  key1: value1
  key2: value2

catalogUiTiles:
  static:
    title: Static API services
    description: Services which demonstrate how to make an API service
discoverable in the APIML ecosystem using YAML definitions

additionalServiceMetadata:
  - serviceId: petstore
    mode: UPDATE # How to update UPDATE=only missing, FORCE_UPDATE=update
all set values
  authentication:
    scheme: bypass

```

- **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 (i.e. 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 monitoringpr1, the API URL in the API Gateway appears as the following URL:

`https://gateway:port/api/v1/monitoringpr1/...`

- If customer system administrator sets the service ID to authenticationprod1, the API URL in the API Gateway appears as the following URL:

`http://gateway:port/api/v1/authenticationprod1/...`

- **title**

This parameter specifies the human readable name of the API service instance (for example, Monitoring Prod or systemInfo 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.

Examples:

- Monitoring Service - Production Instance
- System Info Service 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's REST resource. It will be the prefix for the following URLs:

- **homePageRelativeUrl**
- **statusPageRelativeUrl**
- **healthCheckRelativeUrl**

Examples:

- – `http://host:port/ftpservice` for an HTTP service
- – `https://host:port/source-code-mngmnt` 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.

Examples:

- – `https://host1:port1/source-code-mngmnt`
- – `https://host2:port2/source-code-mngmnt`

- **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. The path is relative to the instanceBaseUrls.

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 your service doesn't have a status page, omit this parameter. The path is relative to the instanceBaseUrls.

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. The path is relative to the instanceBaseUrls.

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. Both specify how the API endpoints are mapped to the API Gateway endpoints.

- **routes.gatewayUrl**

The *gatewayUrl* parameter sets the target endpoint on the Gateway. This is the portion of the final URL that is Gateway specific.

Example:

For the petstore example, the full Gateway URL would be:

```
https://gatewayUrl:1345/api/v2/petstore/pets/1
```

In this case, the URL that will be called on the service is:

```
http://localhost:8080/v2/pets/1
```

- **routes.serviceRelativeUrl**

The *serviceRelativeUrl* parameter points to the target endpoint on the service. This is the base path on the service called through the Gateway.

- **authentication**

Parameters under this grouping allow a service to accept the Zowe JWT token. The API Gateway translates the token to an authentication method supported by a service.

- **authentication.scheme**

This parameter specifies a service authentication scheme. The following schemes are supported by the API Gateway:

- **bypass**

This value specifies that the token is passed unchanged to the service. This is the default scheme when no authentication parameters are specified.

- **zoweJwt**

This value specifies that a service accepts the Zowe JWT token. No additional processing is done by the API Gateway.

- **httpBasicPassTicket**

This value specifies that a service accepts PassTickets in the Authorization header of the HTTP requests using the basic authentication scheme. It is necessary to provide a service APPLID in the apiml.authentication.applid parameter.

Tip: For more information, see [Enabling PassTicket creation for API Services that Accept PassTickets](#) on page 348.

- **zosmf**

This value specifies that a service accepts z/OSMF LTPA (Lightweight Third-Party Authentication). This scheme should only be used for a z/OSMF service used by the API Gateway Authentication Service, and other z/OSMF services that are using the same LTPA key.

Tip: For more information about z/OSMF Single Sign-on, see [Establishing a single sign-on environment](#).

- **authentication.applid**

This parameter specifies a service APPLID. This parameter is only valid for the `httpBasicPassTicket` authentication scheme.

- **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: ..

Tip: 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.

- **apiInfo.defaultApi**

(Optional) This parameter specifies that the API is the default one to show in the API Catalog. If this is not set to true for any API, or multiple APIs have it set to true, then the default API becomes the API with the highest major version as seen in *apiInfo.version*.

- **customMetadata**

Custom metadata are described [here](#).

- **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**

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.{tileId}.description**

This parameter 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.

- **additionalServiceMetadata**

This section contains a list of changes that allows adding or modifying metadata parameters for the corresponding service.

- **additionalServiceMetadata.serviceId**

This parameter specifies the service identifier for which metadata is updated.

- **additionalServiceMetadata.mode**

This parameter specifies how the metadata are updated. The following modes are available:

UPDATE

Only missing parameters are added. Already existing parameters are ignored.

FORCE_UPDATE

All changes are applied. Existing parameters are overwritten.

- **additionalServiceMetadata.{updatedParameter}**

This parameter specifies any metadata parameters that are updated.

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. [Validate successful onboarding](#)

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`

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 on z/OS.

Follow these steps:

1. Locate the Zowe instance directory. The Zowe instance directory is the directory from which Zowe was launched, or else was passed as an argument to the SDSF command used to start Zowe. If you are unsure which instance

directory a particular Zowe job is using, open the JESJCL spool file and navigate to the line that contains STARTING EXEC ZWESVSTC, INSTANCE=. This is the fully qualified path to the instance directory.

Tip: For more information, see [Extensions](#) on page 154.

Note: We use the \${zoweInstanceDir} symbol in following instructions.

2. Add the fully qualified zFS path of your YAML file to `instance.env`.

- To hold your YAML file outside of the instance directory, append the fully qualified zFS path of the YAML file to the `ZWEAD_EXTERNAL_STATIC_DEF_DIRECTORIES` variable in the `instance.env` file. This variable contains a semicolon separated list of static API extension YAML files.
- To place your YAML file within the instance directory, copy your YAML file to the `${zoweInstanceDir}/workspace/api-mediation/api-defs` directory.

Notes:

- The `${zoweInstanceDir}/workspace/api-mediation/api-defs` directory is created the first time that Zowe starts. If you have not yet started Zowe, this directory might be missing.
- The user ID `ZWESVUSR` that runs the Zowe started task must have permission to read the YAML file.

3. Ensure that your application that provides the endpoints described in the YAML file is running.

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](#) on page 327 which allows you to add your static API service to an already running Zowe.

5. [Validate successful onboarding](#)

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://1${zoweHostname}:${gatewayHttpsPort}/api/v2/petstore/pets/1
```

(Optional) Check the log of the API Mediation Layer

The API Mediation Layer log can contain messages based on the API ML configuration. The API ML 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
```

Note: If these messages are not displayed in the log, ensure that the [API ML debug mode](#) is active.

(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](#) and is run with Python 3 installed:

```
httpie --cert=keystore/localhost/localhost.pem --verify=keystore/local_ca/localca.cer -j POST https://localhost:10011/discovery/api/v1/staticApi
```

Alternatively, it is possible to use curl to issue the POST call if it is installed on your system:

```
curl -X POST --cert keystore/localhost/localhost.pem --cacert keystore/localhost/localhost.keystore.cer 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 328
- [Creating a message](#) on page 329
- [Mapping a message](#) on page 329
- [API ML Logger](#) on page 330

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 Message 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}
```

Zowe API Mediation Layer Security

- [Zowe API Mediation Layer Security](#) on page 331
 - [Zowe API Mediation Layer Single-Sign-On Overview](#)
 - [How API ML transport security works](#) on page 331
 - [Transport layer security](#) on page 332
 - [Authentication](#) on page 332
 - [Zowe API ML services](#) on page 332
 - [Zowe API ML TLS requirements](#) on page 332
 - [Authentication for API ML services](#) on page 333
 - [Authentication endpoints](#) on page 333
 - [Authentication providers](#) on page 336
 - [z/OSMF Authentication Provider](#)
 - [Dummy Authentication Provider](#) on page 336
 - [Authorization](#) on page 336
 - [JWT Token](#) on page 335
 - [z/OSMF JSON Web Tokens Support](#)
 - [API ML truststore and keystore](#) on page 337
 - [API ML SAF Keyring](#) on page 337
 - [Discovery Service authentication](#) on page 338
 - [Setting ciphers for API ML services](#) on page 338
 - [ZAAS Client](#) on page 339
 - [Pre-requisites](#) on page 339
 - [API Documentation](#) on page 339
 - [Obtain a JWT token \(`login`\)](#)
 - [Validate and get details from the token \(`query`\)](#)
 - [Invalidate a JWT token \(`logout`\)](#)
 - [Obtain a PassTicket \(`passTicket`\)](#)
 - [Getting Started \(Step by Step Instructions\)](#)
 - [Certificate management in Zowe API Mediation Layer](#) on page 342
 - [Running on localhost](#) on page 342
 - [How to start API ML on localhost with full HTTPS](#) on page 342
 - [Certificate management script](#) on page 342
 - [Generate certificates for localhost](#) on page 342
 - [Generate a certificate for a new service on localhost](#) on page 342
 - [Add a service with an existing certificate to API ML on localhost](#) on page 342
 - [Service registration to Discovery Service on localhost](#) on page 343
 - [Zowe runtime on z/OS](#)
 - [Import the local CA certificate to your browser](#) on page 343
 - [Generate a keystore and truststore for a new service on z/OS](#)
 - [Add a service with an existing certificate to API ML on z/OS](#)
 - [Procedure if the service is not trusted](#) on page 345

How API ML transport security works

Security within the API Mediation Layer (API ML) is performed on several levels. This article describes how API ML uses Transport Layer Security (TLS). As a system administrator or API developer, use this guide to familiarize yourself with the following security concepts:

Transport layer security

Secure data during data-transport by using the TLS protocol for all connections to API Mediation Layer services. While it is possible to disable the TLS protocol for debugging purposes or other use-cases, the enabled TLS protocol is the default mode.

Authentication

Authentication is the method of how an entity, whether it be a user (API Client), or an application (API Service), proves its true identity.

API ML uses the following authentication methods:

- **User ID and password**
 - The user ID and password are used to retrieve authentication tokens.
 - Requests originate from a user.
 - The user ID and password are validated by a z/OS security manager and a token is issued that is then used to access the API service.
- **TLS client certificates**
 - Certificates are for service-only requests.

Zowe API ML services

The following range of service types apply to the ZoweTM API ML:

- **Zowe API ML services**
 - **Gateway Service (GW)** The Gateway is the access point for API clients that require access to API services. API services can be accessed through the Gateway by API Clients. The Gateway receives information about an API Service from the Discovery Service.
 - **Discovery Service (DS)** The Discovery Service collects information about API services and provides this information to the Gateway and other services. API ML internal services also register to the Discovery Service.
 - **API Catalog (AC)** The Catalog displays information about API services through a web UI. The Catalog receives information about an API service from the Discovery Service.
- **Authentication and Authorization Service (AAS)**
AAS provides authentication and authorization functionality to check user access to resources on z/OS. The API ML uses z/OSMF API for authentication. For more information, see: [APIML wiki](#)
- **API Clients**
API Clients are external applications, users, or other API services that are accessing API services via the API Gateway
- **API Services**
API services are applications that are accessed through the API Gateway. API services register themselves to the Discovery Service and can access other services through the Gateway. If an API service is installed in such a way that direct access is possible, API services can access other services without the Gateway. When APIs access other services, they can also function as API clients.

Zowe API ML TLS requirements

The API ML TLS requires servers to provide HTTPS ports. Each API ML service has the following specific requirements:

- **API Client**
 - The API Client is not a server
 - Requires trust of the API Gateway
 - Has a truststore or SAF keyring that contains certificates required to trust the Gateway

- **Gateway Service**
 - Provides an HTTPS port
 - Has a keystore or SAF keyring with a server certificate
 - The certificate needs to be trusted by API Clients
 - This certificate should be trusted by web browsers because the API Gateway can be used to display web UIs
 - Has a truststore or SAF keyring that contains certificates needed to trust API Services
- **API Catalog**
 - Provides an HTTPS port
 - Has a keystore or SAF keyring with a server certificate
 - The certificate needs to be trusted by the API Gateway
 - This certificate does not need to be trusted by anyone else
- **Discovery Service**
 - Provides an HTTPS port
 - Has a keystore or SAF keyring with a server certificate
 - Has a truststore or SAF keyring that contains certificates needed to trust API services
- **API Service**
 - Provides an HTTPS port
 - Has a keystore or SAF keyring with a server and client certificate
 - The server certificate needs to be trusted by the Gateway
 - The client certificate needs to be trusted by the Discovery Service
 - The client and server certificates can be the same
 - These certificates do not need to be trusted by anyone else
 - Has a truststore or SAF keyring that contains one or more certificates that are required to trust the Gateway and Discovery Service

Authentication for API ML services

- **API Gateway**
 - The API Gateway handles authentication
 - There are two authentication endpoints that allow to authenticate the resource by providers
 - Diagnostic endpoints /application/** in API Gateway are protected by basic authentication or Zowe JWT token
- **API Catalog**
 - API Catalog is accessed by users and requires protection by a login
 - Protected access is performed by the Authentication and Authorization Service
- **Discovery Service**
 - Discovery Service is accessed by API Services
 - This access (reading information and registration) requires protection needs by a client certificate
 - (Optional) Access can be granted to users (administrators)
 - Diagnostic endpoints /application/** in Discovery Service are protected by basic authentication or Zowe JWT token
- **API Services**
 - Authentication is service-dependent
 - Recommended to use the Authentication and Authorization Service for authentication

Authentication endpoints

The API Gateway contains two REST API authentication endpoints: auth/login and auth/query.

The `/login` endpoint authenticates mainframe user credentials and returns an authentication token. The login request requires user credentials though one of the following methods:

- Basic access authentication
- JSON with user credentials
- Client certificate

When authentication is successful, the response to the request is an empty body and a token is contained in a secure `HttpOnly` cookie named `apimlAuthenticationToken`. When authentication fails, the user receives a 401 status code.

The `/query` endpoint validates the token and retrieves the information associated with the token. The query request requires the token through one of the following methods:

- A cookie named `apimlAuthenticationToken`
- Bearer authentication

When authentication is successful, the response to the request is a JSON object which contains information associated with the token. When authentication fails, the user receives a 401 status code.

The `/ticket` endpoint generates a PassTicket for the user associated with a token.

This endpoint is protected by a client certificate. The ticket request requires the token in one of the following formats:

- Cookie named `apimlAuthenticationToken`.
- Bearer authentication

The request takes the `applicationName` parameter, which is the name of the application for which the PassTicket should be generated. This parameter must be supplied.

The response is a JSON object, which contains information associated with the ticket.

For more details, see the OpenAPI documentation of the API Mediation Layer in the API Catalog.

Supported authentication methods

The API Mediation Layer provides multiple methods which clients can use to authenticate. When the API ML is run as part of Zowe, all of the following methods are enabled and supported. All methods are supported at least to some extent with each authentication provider.

Username/Password

The client can authenticate via Username and password. There are multiple methods which can be used to deliver credentials. For more details, see the ZAAS Client documentation.

Client certificate

Note:

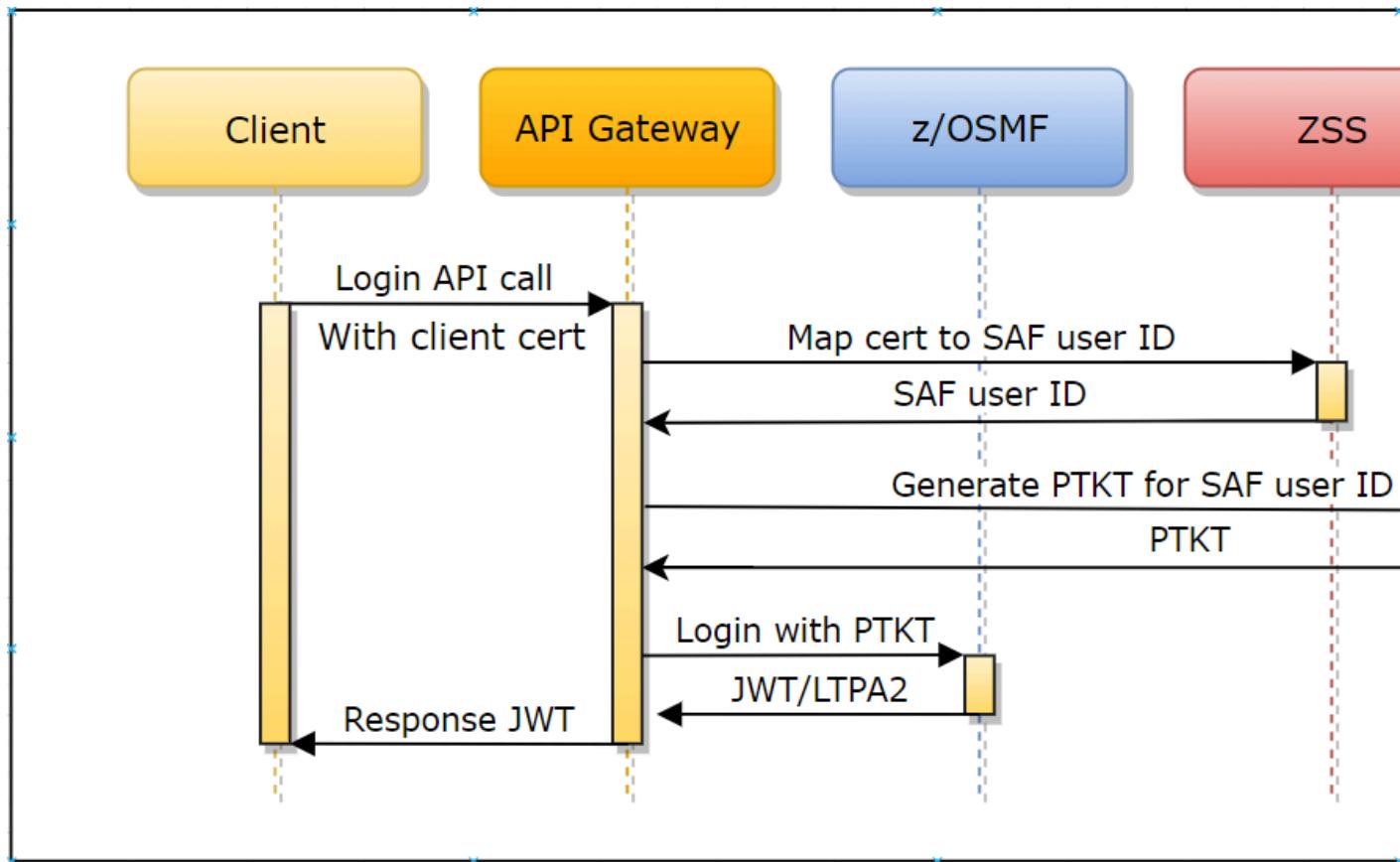
Beginning with release 1.19 LTS, it is possible to perform authentication with client certificates. This feature is functional and tested, but automated testing on various security systems is not yet complete. As such, the feature is provided as a beta release for early preview. If you would like to offer feedback using client certificate authentication, please create an issue against the api-layer repository. Client Certificate authentication will move out of Beta once test automation is fully implemented across different security systems.

If the keyring or a truststore contains at least one valid certificate authority (CA) other than the CA of the API ML, it is possible to use the client certificates issued by this CA to authenticate to the API ML. This feature is not enabled by default and needs to be configured.

Authentication is performed in the following ways:

- The client calls the API ML Gateway login endpoint with the client certificate.
- The client certificate and private key are checked as a valid TLS client certificate against the Gateway's trusted CAs.
- The public part of the provided client certificate is checked against SAF, and SAF subsequently returns a user ID that owns this certificate. ZSS provides this API for the Mediation Layer.

- The Gateway performs the login of the mapped user and returns a valid JWT token.



Prerequisites:

- Alter the Zowe runtime user and set protection by password. The user is created with the NOPASSWORD parameter by the Zowe installer. This password must be changed and a new password has to be set. For RACF, issue the following TSO command:

```
ALTUSER <ZOWE_RUNTIME_USER (ZWESVUSR by default)> PASSWORD(<NEWPASSWORD>)
```

For other security systems, please refer to the documentation for equivalent command.

- Ensure that the Zowe runtime user is allowed to log in to z/OSMF (For example user is member of the default IZUUSER group)
- Ensure that you have an external Certificate Authority and signed client certificates, or generate these certificates in SAF. The client certificate has to have correct Extended Key Usage metadata to allow being used for TLS client authentication. (OID: 1.3.6.1.5.5.7.3.2)
- Import the client certificates to SAF, or add them to a user profile. (Examples: RACDCERT ADD or RACDCERT GENCERT). For more information, see your security system documentation.
- Import the external CA to the truststore or keyring of the API Mediation Layer.
- [Gateway client certificate authentication](#) on page 197.
- To upgrade from Zowe 1.18 or lower, see the [Additional security rights that need to be granted](#).
- Passticket generation must be enabled for Zowe runtime user. The user has to be able to generate Passticket for itself and for ZOSMF's APPLID. [Enabling PassTicket creation for API Services that Accept PassTickets](#) on page 348
- Zowe runtime user has to be enabled to perform identity mapping in SAF. [Additional security rights that need to be granted](#)
- ZSS has to be configured to participate in Zowe SSO. [Using web tokens for SSO on ZLUX and ZSS](#) on page 142

When the client authenticates with the API ML, the client receives the JWT token in exchange. This token can be used for further authentication. If z/OSMF is configured as the authentication provider and the client already received a JWT token produced by z/OSMF, it is possible to reuse this token within the API ML for authentication.

Authentication providers

API ML contains the following providers to handle authentication for the API Gateway:

- z/OSMF Authentication Provider
- SAF Authentication Provider
- Dummy Authentication Provider

z/OSMF Authentication Provider

The z/OSMF Authentication Provider allows the API Gateway to authenticate with the z/OSMF service. The user needs z/OSMF access in order to authenticate.

Use the following properties of the API Gateway to enable the z/OSMF Authentication Provider:

```
apiml.security.auth.provider: zosmf
apiml.security.auth.zosmfServiceId: zosmf # Replace me with the correct z/
OSMF service id
```

SAF Authentication Provider

The SAF Authentication Provider allows the API Gateway to authenticate directly with the z/OS SAF provider that is installed on the system. The user needs a SAF account to authenticate.

Use the following property of the API Gateway to enable the SAF Authentication Provider:

```
apiml.security.auth.provider: saf
```

Dummy Authentication Provider

The Dummy Authentication Provider implements simple authentication for development purpose using dummy credentials (username: user, password user). The Dummy Authentication Provider makes it possible for the API Gateway to run without authenticating with the z/OSMF service.

Use the following property of API Gateway to enable the Dummy Authentication Provider:

```
apiml.security.auth.provider: dummy
```

Authorization

Authorization is a method used to determine access rights of an entity.

In the API ML, authorization is performed by the z/OS security manager ([CA ACF2](#), [IBM RACF](#), [CA Top Secret](#)). An authentication token is used as proof of valid authentication. The authorization checks, however, are always performed by the z/OS security manager.

JWT Token

The JWT secret that signs the JWT Token is an asymmetric private key that is generated during Zowe keystore configuration. The JWT token is signed with the RS256 signature algorithm.

You can find the JWT secret, alias jwtsecret, in the PKCS12 keystore that is stored in \${KEYSTORE_DIRECTORY}/localhost/localhost.keystore.p12. The public key necessary to validate the JWT signature is read from the keystore.

For easy access, you can find the public key in the \${KEYSTORE_DIRECTORY}/localhost/localhost.keystore.jwtsecret.pem file.

You can also use /api/v1/gateway/auth/keys/public/all endpoint to obtain all public keys that can be used to verify JWT tokens signature in a standard [JWK format](#).

z/OSMF JSON Web Tokens Support

Your z/OSMF instance can be enabled to support JWT tokens as described at [Enabling JSON Web Token support](#). In this case, the Zowe API ML uses this JWT token and does not generate its own Zowe JWT token. All authentication APIs, such as `/api/v1/gateway/login` and `/api/v1/gateway/check` function in the same way as without z/OSMF JWT. The `zowe-setup-certificates.sh` stores the z/OSMF JWT public key to the `localhost.keystore.jwtsecret.pem` that can be used for JWT signature validation.

API ML truststore and keystore

A *keystore* is a repository of security certificates consisting of either authorization certificates or public key certificates with corresponding private keys (PK), used in TLS encryption. A *keystore* can be stored in Java specific format (JKS) or use the standard format (PKCS12). The Zowe API ML uses PKCS12 to enable the keystores to be used by other technologies in Zowe (Node.js).

API ML SAF Keyring

As an alternative to using a keystore and truststore, API ML can read certificates from a SAF keyring. The user running the API ML must have rights to access the keyring. From the java perspective, the keyring behaves as the JCERACFKS keystore. The path to the keyring is specified as `safkeyring:///user_id/key_ring_id`. The content of SAF keyring is equivalent to the combined contents of the keystore and the truststore.

Note: When using JCEFACFKS as the keystore type, ensure that you define the class to handle the RACF keyring using the `-D` options to specify the `java.protocol.handler.pkgs` property:

```
-Djava.protocol.handler.pkgs=com.ibm.crypto.provider
```

The elements in the following list, which apply to the API ML SAF Keyring, have these corresponding characteristics:

The API ML local certificate authority (CA)

- The API ML local CA contains a local CA certificate and a private key that needs to be securely stored.
- The API ML local certificate authority is used to sign certificates of services.
- The API ML local CA certificate is trusted by API services and clients.

The API ML keystore or API ML SAF Keyring

- Server certificate of the Gateway (with PK). This can be signed by the local CA or an external CA.
- Server certificate of the Discovery Service (with PK). This can be signed by the local CA.
- Server certificate of the Catalog (with PK). This can be signed by the local CA.
- Private asymmetric key for the JWT token, alias `jwtsecret`. The public key is exported to the `localhost.keystore.jwtsecret.cer` directory.
- The API ML keystore is used by API ML services.

The API ML truststore or API ML SAF Keyring

- Local CA public certificate.
- External CA public certificate (optional).
- Can contain self-signed certificates of API Services that are not signed by the local or external CA.
- Used by API ML services.

Zowe core services

- Services can use the same keystore and truststore or the same keyring as APIML for simpler installation and management.
- When using a keystore and truststore, services have to have rights to access and read them on the filesystem.
- When using a keyring, the user of the service must have authorization to read the keyring from the security system.
- Alternatively, services can have individual stores for higher security.

API service keystore or SAF keyring (for each service)

- The API service keystore contains a server and client certificate signed by the local CA.

API service truststore or SAF keyring (for each service)

- (Optional) The API service truststore contains a local CA and external CA certificates.

Client certificates

- A client certificate is a certificate that is used for validation of the HTTPS client. The client certificate of a Discovery Service client can be the same certificate as the server certificate of the services which the Discovery Service client uses.

Discovery Service authentication

There are several authentication mechanisms, depending on the desired endpoint, as described by the following matrix:

| Endpoint | Authentication method | Note |
|--------------------------------------|------------------------------------|--|
| UI (eureka homepage) | basic auth(MF), token | see note about mainframe authentication |
| application/** | basic auth(MF), token | see note about mainframe authentication |
| application/health, application/info | none | |
| eureka/** | client certificate | Allows for the other services to register without mainframe credentials or token. API ML's certificate can be used. It is stored in the <code>keystore/localhost/localhost.keystore.p12</code> keystore or in the SAF keyring. It is exported to .pem format for convenience. Any other certificate which is valid and trusted by Discovery service can be used. |
| discovery/** | certificate, basic auth(MF), token | see note about mainframe authentication |

Note: Some endpoints are protected by mainframe authentication. The authentication function is provided by the API Gateway. This functionality is not available until the Gateway registers itself to the Discovery Service.

Since the Discovery Service uses HTTPS, your client also requires verification of the validity of its certificate. Verification is performed by validating the client certificate against certificates stored in the truststore or SAF keyring.

Some utilities including HTTPie require the certificate to be in PEM format. The exported certificate in .pem format is located here: `keystore/localhost/localhost.pem`.

The following example shows the HTTPie command to access the Discovery Service endpoint for listing registered services and provides the client certificate:

```
http --cert=keystore/localhost/localhost.pem --verify=false -j GET https://localhost:10011/eureka/apps/
```

Setting ciphers for API ML services

You can override ciphers that are used by the HTTPS servers in API ML services by configuring properties of the Gateway, Discovery Service, and API Catalog.

Note: You do not need to rebuild JAR files when you override the default values in shell scripts.

The *application.yml* file contains the default value for each service, and can be found [here](#). The default configuration is packed in .jar files. On z/OS, you can override the default configuration in <RUNTIME_DIR>/components/<APIML_COMPONENT>/bin/start.sh. Add the launch parameter of the shell script to set a cipher:

```
-Dapimpl.security.ciphers=<cipher-list>
```

On localhost, you can override the default configuration in [config/local/gateway-service.yml](#) (including other YAML files for development purposes).

The following list shows the default ciphers. API ML services use the following cipher order:

Note: Ensure that the version of Java you use is compatible with the default cipherset.

```
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,  
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,  
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256, TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,  
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
```

Only IANA ciphers names are supported. For more information, see [Cipher Suites](#) or [List of Ciphers](#).

ZAAS Client

The ZAAS client is a plain Java library that provides authentication through a simple unified interface without the need for detailed knowledge of the REST API calls presented in this section. The Client function has only a few dependencies including Apache HTTP Client, Lombok, and their associated dependencies. The client contains methods to perform the following actions:

- To obtain a JWT token
- To validate and get details from a JWT token
- To invalidate the JWT token
- To obtain a PassTicket

Pre-requisites

- Java SDK version 1.8.
- An active instance of the API ML Gateway Service.
- A property file which defines the keystore or truststore certificates.

API Documentation

The plain java library provides the ZaasClient interface with following public methods:

```
public interface ZaasClient {  
    String login(String userId, String password) throws ZaasClientException;  
    String login(String authorizationHeader) throws ZaasClientException;  
    ZaasToken query(String token) throws ZaasClientException;  
    ZaasToken query(HttpServletRequest request) throws ZaasClientException;  
    String passTicket(String jwtToken, String applicationId) throws  
        ZaasClientException, ZaasConfigurationException;  
    void logout(String token) throws ZaasClientException,  
        ZaasConfigurationException;  
}
```

This Java code enables your application to add the following functions:

- **Obtain a JWT token (`login`)**
- **Validate and get details from the token (`query`)**
- **Invalidate a JWT token (`logout`)**
- **Obtain a PassTicket (`passTicket`)**

Obtain a JWT token (login)

To integrate login, call one of the following methods for login in the ZaasClient interface:

- If the user provides credentials in the request body, call the following method from your API:

```
String login(String userId, String password) throws ZaasClientException;
```

- If the user provides credentials as Basic Auth, use the following method:

```
String login(String authorizationHeader) throws ZaasClientException;
```

These methods return the JWT token as a String. This token can then be used to authenticate the user in subsequent APIs.

Note: Both methods automatically use the truststore file to add a security layer, which requires configuration in the ConfigProperties class.

Validate and get details from the token (query)

Use the query method to get the details embedded in the token. These details include creation time of the token, expiration time of the token, and the user who the token is issued to.

Call the query method from your API in the following format:

```
ZaasToken query(String token) throws ZaasClientException;
```

In return, you receive the ZaasToken Object in JSON format.

This method automatically uses the truststore file to add a security layer, which you configured in the ConfigProperties class.

The query method is overloaded, so you can provide the HttpServletRequest object that contains the token in the apimlAuthenticationToken cookie or in an Authorization header. You then receive the ZaasToken Object in JSON format.

```
ZaasToken query(HttpServletRequest request) throws ZaasClientException;
```

Invalidate a JWT token (logout)

The logout method is used to invalidate the JWT token. The token must be provided in the Cookie header and must follow the format accepted by the API ML.

Call the logout method from your API in the following format:

```
void logout(String token) throws ZaasClientException,  
ZaasConfigurationException;
```

In return, you receive a 204 HTTP status code if the token was successfully invalidated.

Obtain a PassTicket (passTicket)

The passTicket method has an added layer of protection. To use this method, call the method of the interface and provide a valid APPLID of the application and JWT token as an input.

The APPLID is the name of the application (up to 8 characters) that is used by security products to differentiate certain security operations (like PassTickets) between applications.

This method has an added layer of security, whereby you do not have to provide an input to the method since you already initialized the ConfigProperties class. As such, this method automatically fetches the truststore and keystore files as an input.

In return, this method provides a valid pass ticket as a String to the authorized user.

Tip: For additional information about PassTickets in API ML see [Enabling PassTicket creation for API Services that Accept PassTickets](#).

Getting Started (Step by Step Instructions)

To use this library, use the procedure described in this section.

Follow these steps:

1. Add zaas-client as a dependency in your project.

Gradle:

```
dependencies {
    compile 'org.zowe.apiml.sdk:zaas-client:{version}'
}
```

Maven:

```
<dependency>
    <groupId>org.zowe.apiml.sdk</groupId>
    <artifactId>zaas-client</artifactId>
    <version>{version}</version>
</dependency>
```

2. In your application, create your java class which will be used to create an instance of ZaasClient, which enables you to use its method to login, query, and to issue passTicket.
3. To use zaas-client, provide a property file for configuration.

Tip: Check `org.zowe.apiml.zaasclient.config.ConfigProperties` to see which properties are required in the property file.

Configuration Properties:

```
public class ConfigProperties {
    private String apimlHost;
    private String apimlPort;
    private String apimlBaseUrl;
    private String keyStoreType;
    private String keyStorePath;
    private String keyStorePassword;
    private String trustStoreType;
    private String trustStorePath;
    private String trustStorePassword;
}
```

4. Create an instance of ZaasClient in your class and provide the configProperties object.

Example:

```
ZaasClient zaasClient = new ZaasClientHttps(getConfigProperties());
```

You can now use any method from ZaasClient in your class.

Example:

For login, use the following code snippet:

```
String zaasClientToken = zaasClient.login("user", "user");
```

The following codeblock is an example of a SampleZaasClientImplementation.

Example:

```
public class SampleZaasClientImplementation {
```

```

    /**
     * This method is used to fetch token from zaasClient
     * @param username
     * @param password
     * @return
     */
    public String login(String username, String password) {
        try {
            ZaasClient zaasClient = new
ZaasClientHttps(getConfigProperties());
            String zaasClientToken = zaasClient.login(username, password);
            //Use this token in subsequent calls
            return zaasClientToken;
        } catch (ZaasClientException | ZaasConfigurationException exception)
{
            exception.printStackTrace();
        }
    }

    private ConfigProperties getConfigProperties() {
        // Load the values for configuration properties
    }
}

```

Certificate management in Zowe API Mediation Layer

Running on localhost

How to start API ML on localhost with full HTTPS

The <https://github.com/zowe/api-layer> repository already contains pre-generated certificates that can be used to start API ML with HTTPS on your computer. The certificates are not trusted by your browser so you can either ignore the security warning or generate your own certificates and add them to the truststore of your browser or system.

The certificates are described in more detail in the [TLS Certificates for localhost](#).

Note: When running on localhost, only the combination of using a keystore and truststore is supported.

Certificate management script

Zowe API Mediation Layer provides a script that can be used on Windows, Mac, Linux, and z/OS to generate a certificate and keystore for the local CA, API Mediation Layer, and services.

This script is stored in `zowe/zowe-install-packaging` repository `bin/apiml_cm.sh`. It is a UNIX shell script that can be executed by Bash or z/OS Shell. For Windows, install Bash by going to the following link: [cmdr](#).

Generate certificates for localhost

Follow these steps:

1. Clone the `zowe-install-packaging` repository to your local machine.
2. Place the `bin/apiml_cm.sh` script into `scripts` directory in your API Mediation Layer repository folder
3. Use the following script in the root of the `api-layer` repository to generate certificates for localhost:

`scripts/apiml_cm.sh --action setup`

This script creates the certificates and keystore for the API Mediation Layer in your current workspace.

Generate a certificate for a new service on localhost

To generate a certificate for a new service on localhost, see [Generating certificate for a new service on localhost](#).

Add a service with an existing certificate to API ML on localhost

For more information about adding a service with an existing certificate to API ML on localhost, see [Trust certificates of other services](#).

Service registration to Discovery Service on localhost

To register a new service to the Discovery Service using HTTPS, provide a valid client certificate that is trusted by the Discovery Service.

Zowe runtime on z/OS

Certificates for the API ML local CA and API ML service are managed by installing the Zowe runtime on z/OS. Follow the instructions in [z/OS Installation Roadmap](#) on page 99.

There are two ways of setting up certificates on a z/OS machine.

- certificates in SAF keyring
- certificates in UNIX files (keystore and truststore)

The [Configuring Zowe certificates](#) on page 136 contains instructions about how to set up certificates during installation. Follow the related section below, according to your choice during installation.

Import the local CA certificate to your browser

Trust in the API ML server is a necessary precondition for secure communication between Browser or API Client application. Ensure this trust through the installation of a Certificate Authority (CA) public certificate. By default, API ML creates a local CA. Import the CA public certificate to the truststore for REST API clients and to your browser. You can also import the certificate to your root certificate store.

Notes:

- If a SAF keyring is being used and set up with ZWEKRING JCL, the procedure to obtain the certificate does not apply. It's recommended that you work with your security system administrator to obtain the certificate. Start the procedure at step 2.
- The public certificate in the [PEM format](#) is stored at <KEYSTORE_DIRECTORY>/local_ca/localca.cer where <KEYSTORE_DIRECTORY> is defined in a customized <RUNTIME_DIR>/bin/zowe-setup-certificates.env file during the installation step that generates Zowe certificates. The certificate is stored in UTF-8 encoding so you need to transfer it as a binary file. Since this is the certificate to be trusted by your browser, it is recommended to use a secure connection for transfer.

Follow these steps:

1. Download the local CA certificate to your computer. Use one of the following methods to download the local CA certificate to your computer:
 - **Use Zowe CLI (Recommended)** Issue the following command:

```
zowe zos-files download uss-file --binary $KEYSTORE_DIRECTORY/local_ca/
localca.cer
```
 - **Use sftp** Issue the following command:

```
sftp <system>
get $KEYSTORE_DIRECTORY/local_ca/localca.cer
```

To verify that the file has been transferred correctly, open the file. The following heading and closing should appear:

```
-----BEGIN CERTIFICATE-----
...
-----END CERTIFICATE-----
```

2. Import the certificate to your root certificate store and trust it.

- **For Windows**, run the following command:

```
certutil -enterprise -f -v -AddStore "Root" localca.cer
```

Note: Ensure that you open the terminal as **administrator**. This will install the certificate to the Trusted Root Certification Authorities.

- **For macOS**, run the following command:

```
$ sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.keychain localca.cer
```

- **For Firefox**, manually import your root certificate via the Firefox settings, or force Firefox to use the Windows truststore.

Note: Firefox uses its own certificate truststore.

Create a new Javascript file `firefox-windows-truststore.js` at `C:\Program Files (x86)\Mozilla Firefox\defaults\pref` with the following content:

```
/* Enable experimental Windows truststore support */
pref("security.enterprise_roots.enabled", true);
```

Generate a keystore and truststore for a new service on z/OS

Note: This procedure applies to UNIX file keystore and truststore only. For the SAF keyring option, it's recommended that you perform the actions manually using your security system commands.

You can generate a keystore and truststore for a new service by calling the `apiml_cm.sh` script in the directory with API Mediation Layer:

Call the `apiml_cm.sh` script in the directory with the API Mediation Layer as in the following example.

Example:

```
cd $RUNTIME_DIR
bin/apiml_cm.sh --action new-service --service-alias <alias> --service-ext
<ext> \
--service-keystore <keystore_path> --service-truststore <truststore_path> \
--service-dname <dname> --service-password <password> --service-validity
<days> \
--local-ca-filename $KEYSTORE_DIRECTORY/local_ca/localca
```

where:

- **service-alias** is a unique string to identify the key entry. All keystore entries (key and trusted certificate entries) are accessed via unique aliases. Since the keystore has only one certificate, you can omit this parameter and use the default value `localhost`.
- **service-keystore** is a repository of security certificates plus corresponding private keys. The `<keystore_path>` is the path excluding the extension to the keystore that is generated. It can be an absolute path or a path relative to the current working directory. The key store is generated in PKCS12 format with the `.p12` extension. Ensure that the path is in an existing directory where your service expects the keystore.

Example: `/opt/myservice/keystore/service.keystore`.

- **service-truststore** contains certificates from other parties that you expect to communicate with, or from Certificate Authorities that you trust to identify other parties. The `<truststore_path>` is the path excluding the extension to the trust store that is generated. It can be an absolute path or a path relative to the current working directory. The truststore is generated in PKCS12 format.

- **service-ext** specifies the X.509 extension that should be the Subject Alternate Name (SAN). The SAN contains host names that are used to access the service. You need to specify the same hostname that is used by the service during API Mediation Layer registration.

Example: "SAN=dns:localhost.localdomain,dns:localhost,ip:127.0.0.1"

Note: For more information about SAN, see *SAN or SubjectAlternativeName* at [Java Keytool - Common Options](#).

- **service-dname** is the X.509 Distinguished Name and is used to identify entities, such as those which are named by the subject and issuer (signer) fields of X.509 certificates.

Example: "CN=Zowe Service, OU=API Mediation Layer, O=Zowe Sample, L=Prague, S=Prague, C=CZ"

- **service-validity** is the number of days until the certificate expires.
- **service-password** is the keystore password. The purpose of the password is the integrity check. The access protection for the keystore and keystore need to be achieved by making them accessible only by the ZOVESVR user ID and the system administrator.

The local-ca-filename is the path to the keystore that is used to sign your new certificate with the local CA private key. It should point to the \$KEYSTORE_DIRECTORY/local_ca/localca where \$KEYSTORE_DIRECTORY is defined in a customized \$ZOWE_ROOT_DIR/bin/zowe-setup-certificates.env file during the installation step that generates Zowe certificates.

Add a service with an existing certificate to API ML on z/OS

Note: This procedure applies only to UNIX file keystore/truststore. For the SAF keyring option, we recommend to perform the actions manually using your security system commands.

The API Mediation Layer requires validation of the certificate of each service that it accessed by the API Mediation Layer. The API Mediation Layer requires validation of the full certificate chain. Use one of the following methods:

- Import the public certificate of the root CA that has signed the certificate of the service to the APIML truststore.
- Ensure that your service has its own certificate. If it was signed by intermediate CA, ensure that all intermediate CA certificates are contained in the service's keystore.

Note: If the service does not provide an intermediate CA certificates to the API ML, then validation fails. This can be circumvented by importing the intermediate CA certificates to the API ML truststore.

The following path is an example of importing a public certificate to the API ML truststore by calling in the directory with API Mediation Layer.

Example:

```
cd <RUNTIME_DIR>
bin/apiml_cm.sh --action trust --certificate <path-to-certificate-in-PEM-format> --alias <alias>
```

Procedure if the service is not trusted

If your service is not trusted, you may receive a response with the HTTP status code [502 Bad Gateway](#) and a JSON response in the standardized format for error messages. The following request is an example of when this error response may occur.

Example:

```
http --verify=$KEYSTORE_DIRECTORY/local_ca/localca.cer GET https://<gatewayHost>:<port></port>/api/v1/<untrustedService>/greeting
```

In this example, you receive a similar response:

```
HTTP/1.1 502
Content-Type: application/json; charset=UTF-8
{
    "messages": [
```

```

        {
            "messageContent": "The certificate
            of the service accessed by HTTPS using URI '/api/v1/
            <untrustedService>/greeting' is not trusted by the API Gateway:
            sun.security.validator.ValidatorException: PKIX path building failed:
            sun.security.provider.certpath.SunCertPathBuilderException: unable to find
            valid certification path to requested target",
            "messageKey": "apiml.common.tlsError",
            "messageNumber": "AML0105",
            "messageType": "ERROR"
        }
    ]
}

```

The message has the key `apiml.common.tlsError`, and the message number `AML0105`, and content that explains details about the message.

If you receive this message, import the certificate of your service or the CA that signed it to the truststore of the API Mediation Layer as described above.

API Mediation Layer routing

As an application developer, you can route your service through the Gateway using the API Mediation Layer to consume a specific resource.

There are two ways to route your service to the API Mediation Layer:

- Basic Routing (using Service ID and version)
- Basic Routing (using only the service ID)

Terminology

- **Service**

A service provides one or more APIs, and is identified by a service ID. Note that sometimes the term "service name" is used to mean service ID.

The default service ID is provided by the service developer in the service configuration file.

A system administrator can replace the service ID with a deployment environment specific name using additional configuration that is external to the service deployment unit. Most often, this is configured in a JAR or WAR file.

Services are deployed using one or more service instances, which share the same service ID and implementation.

- **URI (Uniform Resource Identifier)**

A string of characters used to identify a resource. Each URI must point to a single corresponding resource that does not require any additional information, such as HTTP headers.

APIML Basic Routing (using Service ID and version)

This method of basic routing is based on the service ID that identifies the service. The specific instance is selected by the API Gateway. All instances require an identical response. Eureka and Zuul expect this type of routing.

The URI identifies the resource, but does not identify the instance of the service as unique when multiple instances of the same service are provided. For example, when a service is running in high-availability (HA) mode.

Services of the same product that provide different resources, such as CA SYSVIEW on one system and CA SYSVIEW in a different sysplex, cannot have the same service ID (the same URI cannot have two different meanings).

In addition to the basic Zuul routing, the Zowe API Gateway supports versioning in which you can specify a major version. The Gateway routes a request only to an instance that provides the specified major version of the API.

The `/api/` prefix is used for REST APIs. The prefix `/ui/` applies to web UIs and the prefix `/ws/` applies to WebSockets.

You can implement additional routing using a Zuul pre-filter. For more information about how to implement a Zuul filter, see [Router and Filter: Zuul](#)

The URL format expected by the API Gateway is:

```
https://{{gatewayHost}}:{{port}}/api/v{{majorVersion}}/{{serviceId}}/{{resource}}
```

Example:

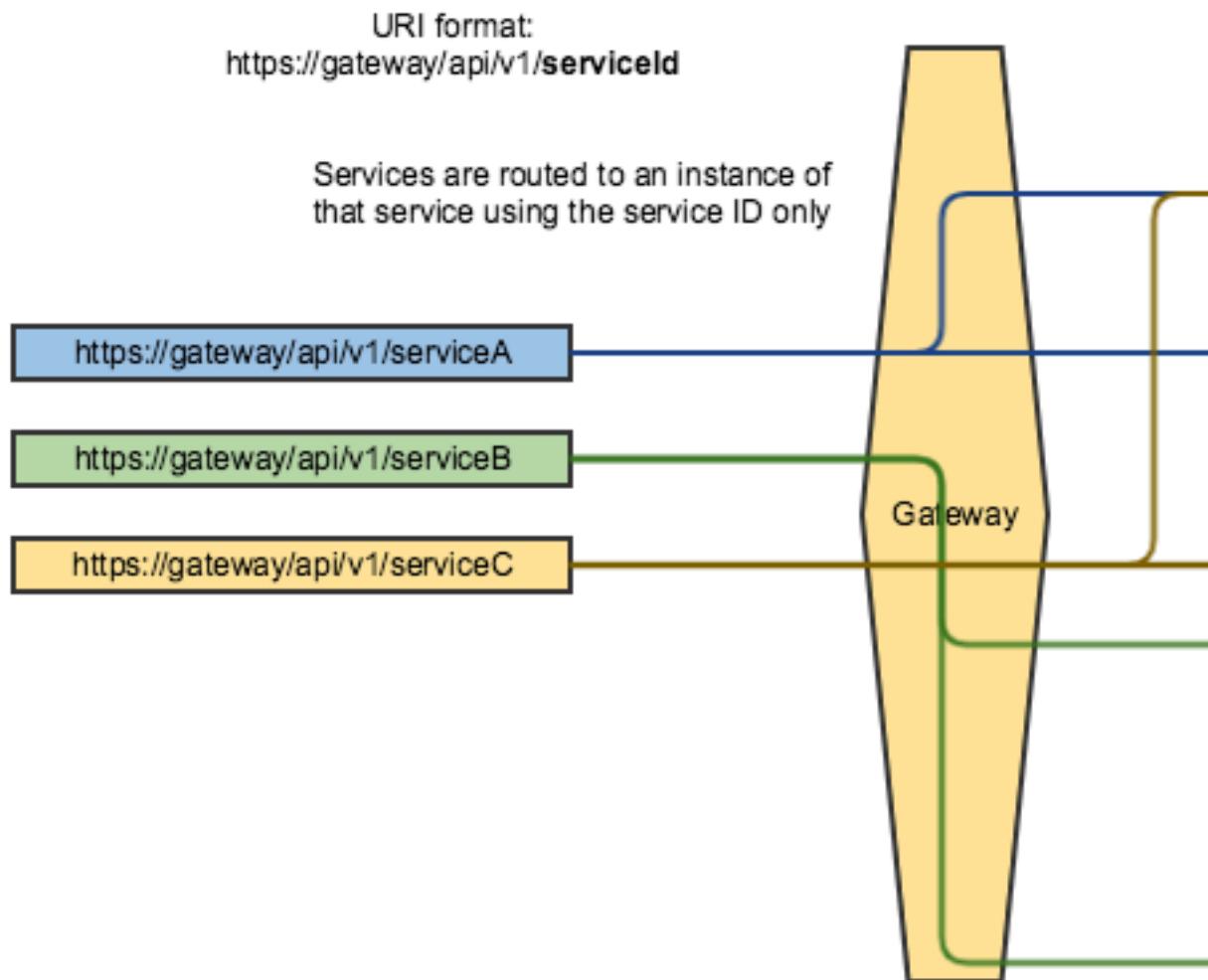
The following address shows the original URL of a resource exposed by a service:

```
http://service:10015/enablerv1sampleapp/api/v1/samples
```

The following address shows the API Gateway URL of the resource:

```
https://gateway:10010/api/v1/enablerv1sampleapp/samples
```

The following diagram illustrates how basic routing works:



Implementation Details

Service instances provide information about routing to the API Gateway via Eureka metadata.

Example:

```
metadata-map:
  apiml:
    routes:
      ui_v1:
        gatewayUrl: "ui/v1"
        serviceUrl: "/helloworld"
      api_v1:
        gatewayUrl: "api/v1"
        serviceUrl: "/helloworld/v1"
      api_v2:
        gatewayUrl: "api/v2"
        serviceUrl: "/helloworld/v2"
```

In this example, the service has a service ID of helloworldservice that exposes the following endpoints:

- **UI - https://gateway/ui/v1/helloworldservice** routed to **https://hwServiceHost:port/helloworld/**
- **API major version 1 - https://gateway/api/v1/helloworldservice** routed to **https://hwServiceHost:port/helloworld/v1**
- **API major version 2 - https://gateway/api/v2/helloworldservice** routed to **https://hwServiceHost:port/helloworld/v2**

where:

- The gatewayUrl is matched against the prefix of the URL path used at the Gateway `https://gateway/urlPath`, where `urlPath` is `prefix/serviceId/resourcePath`.
- The service ID is used to find the service host and port.
- The serviceUrl is used to prefix the resourcePath at the service host.

Note: The service ID is not included in the routing metadata, but the service ID is in the basic Eureka metadata.

Basic Routing (using only the service ID)

This method of routing is similar to the previous method, but does not use the version part of the URL. This approach is useful for services that handle versioning themselves with different granularity.

One example that only uses a service ID is z/OSMF.

Example:

z/OSMF URL through the Gateway: `https://gateway:10010/api/zosmf/restjobs/jobs/...`

where:

- `zosmf` is the service ID.
- `/restjobs/1.0/...` is the rest of the endpoint segment.

Note that no version is specified in this URL.

Enabling PassTicket creation for API Services that Accept PassTickets

As system programmer, you can configure Zowe to use PassTickets for API services that are compatible to accept them to authenticate your service with the API Mediation Layer.

Overview

API clients can use a Zowe JWT token to access an API service even if the API service itself does not support the JWT token. The Zowe JWT token is available through the API Gateway [authentication endpoint](#).

When an API client provides a valid Zowe JWT token to the API ML, the API Gateway then generates a valid PassTicket for any API service that supports PassTickets. The API Gateway then uses the PassTicket to access that API service. The API Gateway provides the user ID and password in the Authorization header of the HTTP requests using the [Basic authentication scheme](#).

- [Outline for enabling PassTicket support](#) on page 349
- [Security configuration that allows the Zowe API Gateway to generate PassTickets for an API service](#) on page 349
 - [CA ACF2](#) on page 349
 - [CA Top Secret](#) on page 349
 - [RACF](#) on page 350
- [API services that support PassTickets](#) on page 350
 - [API Services that register dynamically with API ML that provide authentication information](#) on page 350
 - [API Services that register dynamically with API ML but do not provide metadata](#) on page 350
 - [API services that are defined using a static YAML definition](#) on page 350
- [Adding YAML configuration to API services that register dynamically with API ML](#) on page 351

Outline for enabling PassTicket support

The following steps outline the procedure for enabling PassTicket Support:

1. Follow the API service documentation that explains how to activate support for PassTickets.
 - The PassTickets for the API service must have the replay protection switched off. The PassTickets are exchanged between Zowe API Gateway and the API Service in a secure mainframe environment.
2. Record the value of the APPLID of the API service.
3. Enable the Zowe started task user ID to generate PassTickets for the API service.
4. Enable PassTicket support in the API Gateway for your API service.

Note: PassTickets must be enabled for every user who requires access to the API service.

Security configuration that allows the Zowe API Gateway to generate PassTickets for an API service

Consult with your security administrator to issue security commands to allow the Zowe started task user ID to generate PassTickets for the API service.

Use the following variables to generate PassTickets for the API service to enable the Zowe started task user ID:

- <applid> is the APPLID value used by the API service for PassTicket support (e.g. OMVSAPPL)
- <zowesrv> is Zowe started task user ID used during the Zowe installation

Replace the variables in the following examples with actual values.

CA ACF2

Grant the Zowe started task user ID permission to generate PassTickets for users of that API service. The following code is an example of security commands that need to be issued.

Example:

```
ACF
SET RESOURCE(PTK)
RECKEY IRRPTAUTH ADD(<applid>.- UID(<zowesrv>) SERVICE(UPDATE,READ) ALLOW)
F ACF2,REBUILD(PTK),CLASS(P)
END
```

CA Top Secret

Grant the Zowe started task user ID permission to generate PassTickets for users of that API service.

Example:

```
TSS PERMIT(<zowesrv>) PTKTDATA(IRRPTAUTH.<applid>.) ACCESS(READ,UPDATE)
TSS REFRESH
```

RACF

To enable PassTicket creation for API service users, define the profile IRRPTAUTH.<applid>.* in the PTKTDATA class and set the universal access authority to NONE.

Grant the Zowe started task user ID permission to generate PassTickets for users of that API service.

Example:

```
RDEFINE PTKTDATA IRRPTAUTH.<applid>.* UACC(NONE)
PERMIT IRRPTAUTH.<applid>.* CL(PTKTDATA) ID(<zowesrv>) ACCESS(UPDATE)
SETROPTS RACLIST(PTKTDATA) REFRESH
```

API services that support PassTickets

The following types of API services support PassTickets:

- [API Services that register dynamically with API ML that provide authentication information](#) on page 350
- [API Services that register dynamically with API ML but do not provide metadata](#) on page 350
- [API services that are defined using a static YAML definition](#) on page 350

API Services that register dynamically with API ML that provide authentication information

API services that support Zowe API Mediation Layer and use dynamic registration to the Discovery Service already provide metadata that enables PassTicket support.

As a system programmer, you are not required to do anything in this case. All required information is provided by the API service automatically.

API Services that register dynamically with API ML but do not provide metadata

Some services can use PassTickets but the API ML does not recognize that the service can accept PassTickets. For such services, you can provide additional service metadata externally in the same file that contains the static YAML definiton. The static YAML definitions are described in [Onboard a REST API without code changes required](#) on page 317.

Add the following section to the YAML file with a static definition:

```
additionalServiceMetadata:
  - serviceId: <serviceId>
    mode: UPDATE
    authentication:
      scheme: httpBasicPassTicket
      applid: <applid>
```

where:

- <serviceId>
is the service ID of the service to which you want to add metadata.

API services that are defined using a static YAML definition

Add the following metadata to the same level as the serviceId:

Example:

```
- serviceId: ...
  authentication:
    scheme: httpBasicPassTicket
```

```
applid: TSTAPPL
```

Note: The fields in this example are explained later in this article.

Adding YAML configuration to API services that register dynamically with API ML

As a developer of an API service that registers dynamically with the API ML, you need to provide additional metadata to tell the API Gateway to use PassTickets. Additional metadata tells the API Gateway how to generate them. The following code shows an example of the YAML configuration that contains this metadata.

Example:

```
authentication:
  scheme: httpBasicPassTicket
  applid: <applid>
```

where:

- `httpBasicPassTicket`
is the value that indicates that the HTTP Basic authentication scheme is used with PassTickets.
- `<applid>`
is the APPLID value that is used by the API service for PassTicket support (e.g. OMVSAPPL).

Developing for Zowe Application Framework

Overview

You can create application plug-ins to extend the capabilities of the ZoweTM 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 356
- [Plug-ins definition and structure](#) on page 352
- [Dataservices](#) on page 360
- [Zowe Desktop and window management](#) on page 372
- [Configuration Dataservice](#) on page 375
- [URI Broker](#) on page 381
- [Application-to-application communication](#) on page 382
- [Error reporting UI](#) on page 388
- [Logging utility](#) on page 390

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 extensibility 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 353 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, include 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.

Tagging plugin files on z/OS

When Zowe App Framework is installed on z/OS developrs should tag their plugin files according to the file content. Tagging files helps programs on z/OS understand how to interpret those files, most importantly to know whether a file is encoded using EBCDIC (Extended Binary Coded Decimal Interchange Code). If you are unsure if a plugin you are using is tagged, it can be checked and set using the [chtag command](#). If you want to set the tags, it can be done in bulk with the help of these programs:

- Autotag: This free, open-source application is not part of Zowe. You can download the binary from here for example <https://anaconda.org/izoda/autotag>. Source: <https://github.com/RocketSoftware/autotag>
- The Zowe tagging script: This script tags by file extension. It might not work for all cases, but can be altered to suit your needs. Source: <https://github.com/zowe/zowe-install-packaging/blob/master/scripts/tag-files.sh>

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 357 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 352.

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/default/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 | <code>zlux-app-server/default/plugins</code> | <code>zlux-app-server/plugins</code> | |
| Instance plugins | <code><INSTANCE_DIR>/workspace/app-server/plugins</code> | <code>zlux-app-server/instance/ZLUX/plugins</code> | INSTANCE_DIR is <code>~/.zowe</code> 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;
```

```
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 ApplicationManager 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 ApplicationManager 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. Plugins 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.zosssystem.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 223.

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-i10n 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-i10n
```

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 223 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-i10n` objects within an Angular component, for example:

```
import { LocaleService, TranslationService, Language } from 'angular-i10n';
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;

```

- 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")
);

```

- 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:

- For each language, create a pluginDefinition.i18n.<lang_code>.json file. For example, for German create a pluginDefinition.i18n.de.json file.
- Place the .json files in the web/assets/i18n directory.
- 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",
}

```

- 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"
}

```

- 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 |
|--|--|
| <code>spawnApplication(plugin: DesktopPluginDefinition, launchMetadata: any): Promise<MVDHosting.InstanceId>;</code> | Opens an application instance into the Window Manager, with or without context on what actions it should perform after creation. |
| <code>killApplication(plugin: ZLUX.Plugin, appId:MVDHosting.InstanceId): void;</code> | Removes an application instance from the Window Manager. |
| <code>showApplicationWindow(plugin: DesktopPluginDefinitionImpl): void;</code> | Makes an open application instance visible within the Window Manager. |
| <code>isApplicationRunning(plugin: DesktopPluginDefinitionImpl): boolean;</code> | 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 |
|--|---|
| <code>createViewport(providers:
ResolvedReflectiveProvider[]):
MVDHosting.ViewportId;</code> | Creates a viewport into which an application plug-in's webcontent can be embedded. |
| <code>registerViewport(viewportId:
MVDHosting.ViewportId, instanceId:
MVDHosting.InstanceId): void;</code> | Registers a previously created viewport to an application plug-in instance. |
| <code>destroyViewport(viewportId:
MVDHosting.ViewportId): void;</code> | Removes a viewport from the Window Manager. |
| <code>getApplicationInstanceId(viewportId:
MVDHosting.ViewportId):
MVDHosting.InstanceId null;</code> | 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 372. 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 360.

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 353.

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 382
2. [Actions](#) on page 382
3. [Recognizers](#) on page 385
4. [Dispatcher](#) on page 386

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.

Cross-launch via URL

Another way the Zowe Application Framework invokes Actions is via URL Query Parameters, with parameters formatted in JSON. This feature enables users to bookmark a set of application-to-application communication actions (in the form of a URL) that will be executed when opening the webpage. Developers creating separate web apps can build a link that will open the Zowe Desktop and do specific actions in Apps, for example, opening a file in the Editor.

The Cross-launch feature allows you to:

1. Specify one or more actions that will be executed upon login, allowing you to bookmark a series of actions that you can share with someone else.
2. Specify actions that are declared by plugins (when formatter is equal to a known action ID) or actions that you have custom-made (when formatter = 'data').
3. Customize the action type, mode, and target plugin (when the formatter is equal to an existing action ID).

Sample URL

```

https://localhost:8544/ZLUX/plugins/org.zowe.zlux.bootstrap/web/?app2app=org.zowe.zlux.ng2desktop.webbrowser:launch:create:data:{"url":"https://github.com/zowe/zlux-app-manager/pull/234","enableProxy":true}&app2app=org.zowe.zlux.ng2desktop.webbrowser:message:create {"url":"https://github.com/zowe/zlux-app-manager/pull/234","enableProxy":true}&app2app=org.zowe.zlux.ng2desktop.webbrowser:message:create {"data": {"url": "https://github.com/zowe/zlux-app-manager/pull/234", "enableProxy": true}}

```

Query parameter format:

```
?app2app={pluginId}:{actionType}:{actionMode}:{formatter}:
{contextData}&app2app={pluginId}:{actionType}:{actionMode}:{formatter}:
{contextData}
```

- **pluginId** - application identifier, e.g. 'org.zowe.zlux.ng2desktop.webbrowser'
- **actionType** - 'launch' | 'message'
- **actionMode** - 'create' | 'system'
- **formatter** - 'data' | actionId
- **contextData** - context data in form of JSON

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

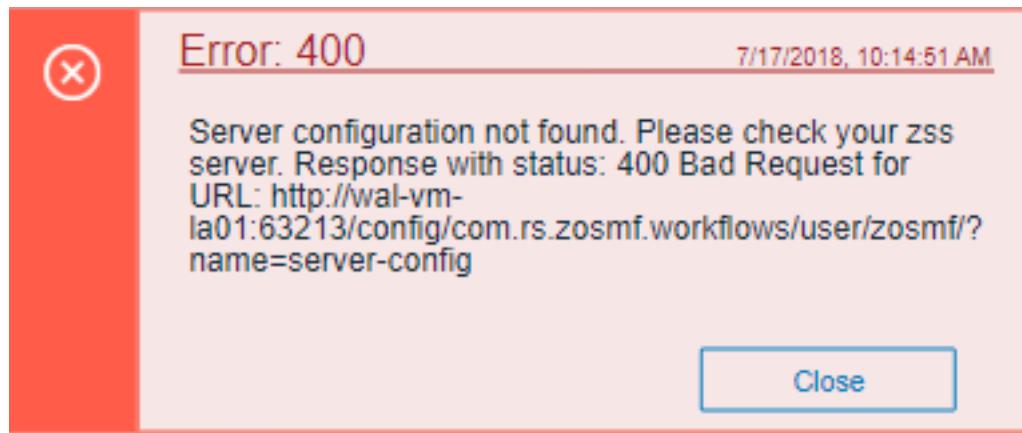
(often classified by a code, for example: 404). The optional parameters in options specify that this error will block the user from interacting with the application plug-in until the error is closed or it until goes away on its own. globalVeilService is optional and is used to create a gray veil on the 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 360.

Web

(Angular App Instance Injectable). See **Logger** in [Zowe Desktop and window management](#) on page 372.

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,:)
- Application created.
```

Or in another supported language, such as Russian:

```
DATE TIME:TIME:TIME.TIME <ZWED:> username INFO (org.zowe.app.name,:)
- ##### ####.
```

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 |
| 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 |

| Level | Number |
|--------|--------|
| 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.setLevelForComponentName('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 \(zLUX\) configuration](#).

Using Conda to make and manage packages of Application Framework Plugins

As Zowe is composed of components which can be extended by Plugins, a standardized and simple way to find, install, upgrade, and list Plugins in your Zowe environment is important to make it easy to get the most out of Zowe.

Package management as a concept generally provides a way to find packages such as plugins, check and possible co-install dependencies the package has, and ultimately install the desired package. Post-install, management tasks such as upgrading and uninstalling are common.

Conda is one such package manager, and if you are familiar with apt, yum, or npm, you will find that using Conda is very similar. But, there are some important abilities that make Conda stand out:

- Very cross platform: Conda is available, and acts very similar on z/OS, Windows, Linux, macOS, and various Unix. Packages can state which platforms they support, so it easy to know what packages you can install.
- Tagging: On z/OS, Conda packages can contain tagging information, to avoid issues around the difference between EBCDIC & ASCII.
- Software neutrality: Language-specific package managers are becoming popular, but Conda does not assume the purpose of the package, so you can install almost anything.

- Environments: If desired, every user can have a different set of packages, because Conda can install & manage packages in personal folders instead of system ones. A user can even have multiple such environments, and switch between them rapidly to work with different sets of related software without conflict.

Initial Conda setup

If you have not installed Conda yet, it can be downloaded as an all-in-one package that has no extra dependencies, known as "miniconda". For Linux, Unix, macOS, and Windows, this can be downloaded at <https://docs.conda.io/en/latest/miniconda.html> For z/OS, Conda can be downloaded from Rocket Software at <https://www.rocketsoftware.com/zos-open-source>

Conda will prompt during the install for certain setup options, and ultimately you'll want to put some Conda initialization content into your startup script so that whenever you open your terminal, Conda will be ready for your use.

Once you have Conda downloaded and installed, you'll want to create your first Conda "environment" this can be done by providing a path or a nickname

```
conda create --prefix PATH conda create --name ENVIRONMENT
```

Either will work, but path helps you better separate your content from content others use by placing it in a folder that you can have stricter permissions on.

If you need to know more about certain commands, you can use the help command for any.

```
conda create --help
```

Or, check the official documentation: <https://docs.conda.io/en/latest/index.html>

Once you have an environment, you should activate it so that the actions you do are on that environment, as opposed to the base one.

```
conda activate PATH_OR_NAME
```

Conda will detect whether the parameter is a path or a nickname, so this command works for both.

Finally, you can view the Conda environment and other information by checking "info"

```
conda info
```

Managing Conda channels

When downloading a package, such as a Zowe Plugin, the place that you download from is configurable. These are called "Channels", but are very similar to "Repositories" seen in other package managers. With Conda, you can install from:

- A network channel (Internet or company internal)
- A local channel (Collection of plugins on your computer)
- Just an individual package, without a channel

You can have multiple of each, and if a package is present in more than one location, you can specify which one to use.

Searching for packages

Conda has a search utility that searches for all Channels,

```
conda search anything_you_want
```

but it's important to note that because any type of software can be installed through Conda, you probably want to search through a detailed view to help identify which ones are meant for Zowe, or use Channels that are distinctly for Zowe so that you can get packages that are strictly for Zowe.

```
conda search --info anything_you_want
```

Using Conda with Zowe

Zowe is not yet available in the form of Conda packages yet, so it must be installed separately. If you have Zowe installed on the same system as Conda, some Zowe Plugins installed through Conda will automatically register into Zowe. In order to do this, the Plugins must be able to find Zowe. You should set environment variables before trying to install the Plugins:

Setting environment variables temporarily:

z/OS, Linux, Unix:

```
export ZOWE_INSTANCE_DIR=/path/to/zowe/instance
export ZOWE_ROOT_DIR=/path/to/zowe/installation
```

Windows cmd.exe:

```
set ZOWE_INSTANCE_DIR=\path\to\zowe\instance
set ZOWE_ROOT_DIR=\path\to\zowe\installation
```

INSTANCE_DIR and ROOT_DIR are also supported, but the ZOWE_ prefix helps distinguish its purpose.

Setting environment variables persistently

z/OS, Linux, Unix: You can put the `export` statements into the `.profile` file in your home directory to have them apply on login.

Windows: There is a UI to set variables, but it varies depending on Windows version. Try typing 'environment variable' into the Windows search bar to get to the relevant menu.

Installing a Zowe plugin

A Conda package could contain one or more Zowe Plugins, and a Conda package could contain non-Zowe code alongside Zowe Plugins. This is left up to the program vendor and regardless the install process is the same:

```
conda install package_name
```

If the Zowe environment variables are set, such a package may automatically register Plugins into the Zowe instance of your choice.

Zowe plugin configuration

Aside from possible automation during install and uninstall, Conda does not manage Zowe, its configuration, or configuration of the Plugins. However, Conda does manage the package files, and therefore you can do additional Zowe tasks on the Plugins by going into the Conda environment. Zowe Plugins are intended to be found in a standardized location in the Conda environment,

```
/opt/zowe/plugins
```

This folder contains Plugins, which in turn contain sub-folders that are the Zowe components that they utilize. If a plugin uses multiple Zowe components, its contents could be found within multiple component folders.

```
/opt/zowe/plugins/my_plugin/app-server /opt/zowe/plugins/my_plugin/cli
```

Zowe package structure

Zowe Plugins packaged into Conda follow the structure outlined here: <https://github.com/zowe/zowe-install-packaging/issues/1569> This structure allows for plugin to have content meant for one or more Zowe components. The Conda packages extend this by allowing for more than one Plugin, or a mix of Zowe Plugins and other software to be within a single package.

Building Conda packages for Zowe

This document is intended to be provided with example scripts by the Zowe community, which shows you how you can build a simple Zowe plugin into a Conda package. You can find the example scripts on the [Zowe zlux-build github repository](#). This is not intended to be a one-size-fits-all set of scripts. If you have more advanced needs, you can use these scripts as a basis for writing your own scripts.

To make a Conda package, you need conda-build, which you can install into a Conda environment:

```
conda install conda-build
```

Once you have it, you can build a package via

```
conda build path/to/build/scripts
```

However, first you must set up the build information.

Defining package properties

Conda needs a metadata file, `meta.yaml` to state information about the package, such as dependencies, what OS it supports, its name and version. This information can be programmatically found, and Zowe provides examples of how to do this by reading Zowe's own metadata files into this one.

Creating build step

It's recommended not to build your code from scratch to put into Conda. Rather, build your code however you want, and then just copy the contents into a Conda package. This keeps the Conda scripting small and simple.

In the same folder as `meta.yaml`, Conda requires `build.sh` for building on Unix, Linux, or z/OS and `build.bat` for Windows. Except for z/OS, this script does not determine where your package can be used, it's just about where you are building it. z/OS is the exception because when you build on z/OS, unix file tagging information is preserved. So, it's highly recommended that you tag your files so that users do not have to deal with encoding issues. For code that works equally well on all platforms, a simple way to build for all is:

1. Build your code on Linux
2. Transfer the output to z/OS
3. Run a Conda build on the output on Linux
4. Run a Conda build on the output on z/OS
5. Deliver the Linux package as 'noarch' content, and the z/OS package as 'zos-z' content.

Lifecycle scripts

When a Conda package is installed or uninstalled, a script from the package can be run. For Zowe, the scripts `post-link.sh` and `pre-unlink.sh` can be important, and you must put them into the same folder as `meta.yaml` for building.

Install automation

`post-link.sh` runs at install, after Conda has put the package content onto the system. At this time, registration into Zowe is recommended if the Plugin does not require any information from the user for configuration. If the Plugin is okay to be automatically installed, we recommend putting a script into the package folder named `autoinstall.sh`. Zowe's provided Conda examples will utilize `autoinstall.sh` to do any install steps your package needs, and provides Zowe information to make install simple. However, it's possible to do what you want in your own `post-link.sh` script instead.

Uninstall automation

`pre-unlink.sh` is the opposite of `post-link.sh`. It allows you to do anything you need to before the package is removed from the system. This is a good time to remove any package information from Zowe, but you should be careful because users may uninstall and later re-install, so you should not remove configuration information without consent.

Adding configuration to Conda packages

As a package manager, Conda is not responsible for configuration. Your packages can include defaults to utilize, but if configuration is needed you should alert the user to perform a post-install task. `post-link.sh` could be used to print such an alert.

Developing for Zowe SDKs

The Zowe SDKs are open source. You can contribute to add features, enhancements, and bug fixes to the source code.

The functionality is currently limited to the interfaces provided by IBM z/OSMF. As a plug-in developer, you can enhance the SDK by creating packages that exposes programmatic APIs for your service.

For detailed contribution guidelines, see the following documents:

- [Node.js SDK guidelines](#)
- [Coming soon! Python SDK guidelines](#)

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.

To learn the criteria of achieving Zowe conformance for an offering, see [Zowe Conformance Criteria](#).

How to suggest updates to the Zowe conformance program

The Zowe conformance criteria is available as a table in a [Markdown file](#) in the Open Mainframe Project's GitHub repo. If you find a mistake with the Zowe conformance documents, or you are a Zowe squad lead and want to make an amendment to the criteria, you can update that Markdown file. The same information is also held in another document [Zowe Conformance Test Evaluation Guide](#) that has history going back to Zowe 2019 conformance and allows easy change history comparison.

To submit a proposal to update the conformance criteria, fork the OMP's `foundation` repository at <https://github.com/openmainframeproject/foundation> and make a pull request. Flag the Pull Request to the attention of GitHub user ID `@mertic`, and also reach out to the Zowe onboarding squad in the `#zowe-onboarding` Slack channel. If you are not already signed up to Zowe Slack community, you can sign up at <https://slack.openmainframeproject.org> first.

Chapter

4

Troubleshooting

Topics:

- Overview
- Troubleshooting installation and startup of Zowe z/OS components
- Zowe API Mediation Layer
- Zowe Application Framework
- Troubleshooting z/OS Services
- Zowe CLI
- Zowe Explorer

Overview

Troubleshooting

To isolate and resolve Zowe™ problems, you can use the troubleshooting and support information.

Known problems and solutions

Some common problems with Zowe are documented, along with their solutions or workarounds. If you have a problem with Zowe installation and components, review the problem-solution topics to determine whether a solution is available to the problem that you are experiencing.

You can also find error messages and codes, must-gathers, and information about how to get community support in these topics.

- [Troubleshooting installation and startup of Zowe z/OS components](#) on page 407
- [Troubleshooting API ML](#) on page 413
- [Troubleshooting Zowe Application Framework](#) on page 439
- [Troubleshooting z/OS Services](#) on page 448
- [Troubleshooting Zowe CLI](#) on page 451

Collecting data for Zowe problems

Sometimes you cannot solve a problem by troubleshooting the symptoms. In such cases, you must collect diagnostic data. To collect diagnostic data about Zowe, see [Capturing diagnostics to assist problem determination](#) on page 401.

Verifying a Zowe release's integrity

Following a successful install of a Zowe release, the Zowe runtime directory should contain the code needed to launch and run Zowe. If the contents of the Zowe runtime directory have been modified then this may result in unpredictable behavior. To assist with this Zowe provides the ability to validate the integrity of a Zowe runtime directory, see [Verify Zowe runtime directory](#) on page 403

Understanding the Zowe release

Knowing which version of Zowe you are running might help you isolate the problem. Also, the Zowe community who helps you will need to know this information. For more information, see [Understanding the Zowe release](#) on page 400.

Understanding the Zowe release

Zowe releases

Zowe uses semantic versioning for its releases, also known as SemVer. Each release has a unique ID made up of three numbers that are separated by periods.

```
<Major Version>.<Minor Version>.<Patch Version>
```

Each time a new release is created, the release ID is incremented. Each number represents the content change since the previous release. For example,

- 1.5.0 represents the fifth minor release since the first major release.
- 1.5.1 represents the first patch to the 1.5.0 release.
- 1.6.0 is the first minor release to be created after 1.5.1.

Patch

A patch is usually reserved for a bug fix to a minor release.

Minor release

A minor release indicates that new functionality is added but the code is compatible with an earlier version. The Zowe community works on two-week sprints and creates a minor release at the end of these, typically once per month although the frequency might vary.

Major release

A major release is required if changes are made to the public API and the code is no longer compatible with an earlier version.

When Zowe is version one, it is associated with the Zowe v1 [Zowe Conformance Program](#) on page 397. Offerings that extend Zowe and achieve the Zowe v1 conformance badge will remain compatible with Zowe throughout its version 1 lifetime. A major release increment because of incompatibility is sometimes referred to as a "breaking" change.

The first SMP/E build for Zowe v1 has a Functional Module ID (FMID) of AZWE001, which was created with content from the 1.9.0 release. Each major release will be its own SMP/E FMID where the last digit is updated, for example AZWE00V where V represents the major version.

Subsequent minor and patch releases to V1 are delivered as SMP/E PTF SYSMODs. Because of the size of the content, two co-requisite PTFs are created for each Zowe release.

While Major releases are required for a "breaking" change, they also can be used to indicate to the community a significant content update over and above what would be included in a minor release.

Check the Zowe release number

To see the release number of Zowe, look at the `manifest.json` file. This is included in the top-level [Installing Zowe runtime from a convenience build](#) on page 102, the top-level directory of a Zowe runtime `<RUNTIME_DIR>`, and the [Creating and configuring the Zowe instance directory](#) on page 150 `<INSTANCE_DIR>/workspace`.

To see the version of a Zowe release, use the Unix `grep` command in a directory that contains a `manifest.json` file.

```
>cat manifest.json | grep version | head -1
```

will return a single line with the Zowe release number. For example,

```
"version": "1.10.0",
```

Capturing diagnostics to assist problem determination

To help Zowe™ Open Community effectively troubleshoot Zowe, a shell script `zowe-support.sh` captures diagnostics data that is required for successful problem determination. 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 `<INSTANCE_DIRECTORY>/bin` directory. To determine the `<INSTANCE_DIRECTORY>` for a Zowe started task, open the `JESJCL` step in the `ZWESVSTC` task and navigate to the line including `//STARTING EXEC ZWESVSTC, INSTANCE=.`. The `<INSTANCE_DIRECTORY>/bin/zowe-support.sh` 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)

Note: You will need to install the TSO exit IKJEFF53 to permit the TSO OUTPUT command to collect the Zowe started task output. If this exit is not enabled, you will see an error message when you run `zowe-support.sh`:

```
IKJ56328I JOB jobname REJECTED - JOBNAME MUST BE YOUR USERID OR MUST START WITH YOUR USERID
```

For how to correct this error, see the [TSO/E installation exit IKJEFF53](#) topic in IBM Knowledge Center. The above is the authoritative description, and will be the first to reflect changes. To assist you, a summary of the situation and actions you could take to allow TSO OUTPUT to work in your installation are provided in [IKJ56328I JOB name REJECTED](#) on page 402.

- 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)

Running the diagnostic support script

To run the `zowe-support.sh` script, issue the following commands:

```
<INSTANCE_DIRECTORY>/bin/zowe-support.sh [-l <install_logs_directory>]
```

where the `-l` optional parameter points to the custom directory supplied during the installation and setup scripts (`zowe-install.sh`, `zowe-setup-certificates.sh`, `zowe-install-xmem.sh`, `zowe-install-proc.sh`) if applicable.

Problems that may occur running the diagnostic script

IKJ56328I JOB job name REJECTED

Audience: Zowe users or the personnel who collects support logs. These individuals should also inform their z/OS system programmer.

The `zowe-support.sh` script collects logs that your support team needs to assist you with problem determination. One of the logs it collects is the JES job log for Zowe tasks. The `zowe-support.sh` script uses the TSO OUTPUT command to collect these logs. On an unmodified z/OS system, the TSO OUTPUT command is restricted to jobs starting with your user ID, and the Zowe tasks will typically have a different job name. You will know that the TSO

OUTPUT command is restricted if you see the following message when you issue the TSO OUTPUT command for a job whose job name does not start with your user ID.

```
IKJ56328I JOB job name REJECTED - JOB NAME MUST BE YOUR USERID OR MUST START
WITH YOUR USERID
```

Job name filtering is controlled by an exit that is called by the TSO OUTPUT command. The exit is named IKJEFF53. IBM provides the source code for a replacement exit that can remove this restriction in SYS1.SAMPLIB(IKJEFF53) which you can tailor and use instead. Review this exit and if it meets your needs, assemble it and replace it in your LINKLIB concatenation.

Warning: You are strongly advised to take great care before attempting to modify LINKLIB directly. You could easily corrupt your entire z/OS system and require an IPL or reinstallation of z/OS. Consult your system programmer before you continue. You should also read the TSO/E installation exit IKJEFF53 topic in the IBM Knowledge Center.

The original exit load module has the following attributes:

| DSLIST | | SYS1.LINKLIB | | | | | | Scroll ==> | |
|-------------|--|--------------|--------|----------|----------|--------|----|------------|--|
| Command ==> | | | | | | | | | |
| CSR | | Name | Prompt | Alias-of | Size | TTR | AC | AM | |
| RM | | IKJEFF53 | | | 000002A0 | 03A91D | 00 | 24 | |
| 24 | | | | | | | | | |

The replacement exit load module has the following attributes when assembled and linked:

| DSLIST | | SYS1.LINKLIB | | | | | | Scroll ==> | |
|-------------|--|--------------|--------|----------|----------|--------|----|------------|--|
| Command ==> | | | | | | | | | |
| CSR | | Name | Prompt | Alias-of | Size | TTR | AC | AM | |
| RM | | IKJEFF53 | | | 00000380 | 0CF015 | 00 | 24 | |
| 24 | | | | | | | | | |

It is safer to add a module to a private dataset further down the LINKLIST concatenation than to modify SYS1.LINKLIB directly. You can display the LINKLIST with this command at the operator console

```
D PROG,LNKLST
```

The private dataset or the exit itself does not have to be APF-authorized.

Whichever library you choose, rename IKJEFF53 in SYS1.LINKLIB (or its first occurrence in the LINKLIST concatenation) and add the assembled load module IKJEFF53 to your chosen library.

To activate your changes, refresh the link-list lookaside list with this command

```
F LLA,REFRESH
```

Now your TSO OUTPUT command will work as described in SYS1.SAMPLIB(IKJEFF53).

Note that this change will affect all users of the TSO OUTPUT command on LPARS sharing the SYS1.LINKLIB dataset. It is not limited to Zowe users. Consult your system programmer to ensure that this change does not impact your site rules about the OUTPUT command, because the specified jobs will be PURGED from the JES output queue if this exit is implemented as described above.

Verify Zowe runtime directory

The Zowe runtime directory RUNTIME_DIR contains the code modules that make up Zowe. If these code modules are altered in any way, the behavior of Zowe is unpredictable. To check if the RUNTIME_DIR has been altered,

Zowe provides a verify tool that comprises a script file `zowe-verify-authenticity.sh` and the files it needs to check the release contents.

You can use this verify tool on Zowe version 1.9 and later.

- If you use Zowe version 1.14 or later, the verify tool is delivered with Zowe, so you can skip [Step 1](#) below, but you can still download the verify tool if required.
- If you use Zowe version 1.9, 1.10, 1.11, 1.12, and 1.13, you must obtain the verify tool separately and use it to verify the `RUNTIME_DIR`.

Contents in this topic

- [Step 1: Obtain the verify tool \(Required for versions before v1.14\)](#) on page 404
- [Step 2: Verify your runtime directory](#) on page 404
- [Step 3: Review results](#) on page 405
 - [Mismatch](#) on page 405
 - [Match](#) on page 406
- [zowe-verify-authenticity.sh parameters](#) on page 406
- [Use of zowe-verify-authenticity.sh by zowe-support.sh](#) on page 407

Step 1: Obtain the verify tool (Required for versions before v1.14)

1. Start a USS terminal session with the z/OS system where Zowe is installed.
2. Create a new, writable, local directory, for example, `/u/username/hash`.
3. Go to the [download link](#) to download the `fingerprint.pax` PAX file.
4. Upload the downloaded file to a temporary directory such as `/tmp` on your z/OS USS file system by using SFTP or a similar file transfer utility. When you transfer the PAX file between systems, you must use binary transfer mode.
5. Extract the PAX file from inside the local directory you created (in this example, it is `/u/username/hash`) using commands like the following one:

```
cd /u/username/hash
pax -ppx -rf /tmp/fingerprint.pax
```

6. When the PAX file is extracted, you will see the following files in your `/u/username/hash` directory:
 - `HashFiles.class` (binary)
 - `RefRuntimeHash-1.9.0.txt` (text)
 - `RefRuntimeHash-1.10.0.txt` (text)
 - `RefRuntimeHash-1.11.0.txt` (text)
 - `RefRuntimeHash-1.12.0.txt` (text)
 - `RefRuntimeHash-1.13.0.txt` (text)
 - `zowe-verify-authenticity.sh` (text)

Each `RefRuntimeHash-V.v.p.txt` file is specific to a Zowe release, where `V.v.p` is your Zowe release number, for example, `1.9.0`. This list of files is updated to include new Zowe releases as they become available. For example, if you use Zowe version 1.14, you will see `RefRuntimeHash-1.14.0.txt` in the list.

Step 2: Verify your runtime directory

Now you are ready to verify your runtime directory `RUNTIME_DIR`, for example, `/usr/lpp/zowe/v1.14`, which contains the following files. You can show these files by using the `ls` command.

```
/u/username/hash:>ls /usr/lpp/zowe/v1.14
bin  components  fingerprint  install_log  manifest.json  scripts
```

Note that you will not have a `fingerprint` directory in releases prior to v1.14.0.

1. Change to the runtime directory.

```
cd /usr/lpp/zowe/v1.14
```

2. Run the zowe-verify-authenticity.sh script.

The script will automatically choose the correct RefRuntimeHash-V.v.p.txt file that matches the release found in your runtime directory.

For Zowe v1.14 and later

Issue the following command. You do not need to specify any parameters to this script.

```
bin/zowe-verify-authenticity.sh
```

If you suspect that the versions of the files in RUNTIME_DIR have been altered since this version of Zowe was installed, you might want to use the verify tool's script or files which you downloaded instead of the ones in your runtime directory. In this case, you can call the downloaded script and specify the options -f and -h in the following way:

```
/u/username/hash/zowe-verify-authenticity.sh -f /u/username/hash -h /u/username/hash
```

To display a list of parameters, enter this command

```
bin/zowe-verify-authenticity.sh -?
```

For Zowe releases prior to v1.14

Issue commands similar to the following. In this example, you use Zowe v1.9.

```
/u/username/hash/zowe-verify-authenticity.sh -r /usr/lpp/zowe/v1.9 -f /u/username/hash -h /u/username/hash
```

The zowe-verify-authenticity.sh script creates a CustRuntimeHash.txt file, which it compares with the RefRuntimeHash-V.v.p.txt file.

Step 3: Review results

You will get one of the following results.

- [Mismatch](#) on page 405
- [Match](#) on page 406

Mismatch

When files don't match, you see output similar to the following.

```
USERNAME:/u/username/hash: >zowe-verify-authenticity.sh -l ~/hash-v1.12.0
zowe-verify-authenticity.sh started
Info: Logging to directory /u/username/hash-v1.12.0
Info: zoweVersion = 1.12.0
Info: Gathering files ...
Info: Checking java ...
Info: Calculating hashes ...
Info: Comparing results ...
Info: Number of files different = 14749
Info: Number of files extra      = 171
Info: Number of files missing   = 22
Error: Verification FAILED
Info: Result files and script log are in directory /u/username/hash-v1.12.0
zowe-verify-authenticity.sh ended
USERNAME:/u/username/hash: >
```

Troubleshooting and hints

This is a worst-case scenario of a bad mismatch. To find out what the problem is, you could, for example, start by referring to the [Check the Zowe release number](#) on page 401 to see whether one of the components is from the wrong release.

If you have many files different but none missing or extra, you might have a file tagging or code-page problem. Check that the environment variables are set as required according to `zowe-set-env.sh`.

The hash files mentioned above are left in the `/u/username/hash` directory in case you want to use a GUI tool to perform a better file comparison.

Match

When files match, you see output similar to the following.

```
zowe-verify-authenticity.sh started
Info: Logging to directory /u/username/hash-v1.12.0
Info: zoweVersion = 1.12.0
Info: Gathering files ...
Info: Checking java ...
Info: Calculating hashes ...
Info: Comparing results ...
Info: Number of files different = 0
Info: Number of files extra      = 0
Info: Number of files missing    = 0
Info: Verification PASSED
Info: Result files and script log are in directory /u/username/hash-v1.12.0
zowe-verify-authenticity.sh ended
```

`zowe-verify-authenticity.sh` parameters

Usage:

```
zowe-verify-authenticity.sh [-r <runtime-dir>] [-h <HashPgm-dir>] [-f
<HashRef-dir>] [-l <output-dir>]
```

- All parameters are optional
- You can use dot (.) and tilde (~) in the parameters

Description of parameters:

- `-r <runtime-dir>`

Root directory of the executables used by Zowe at run time. The typical value is `/usr/lpp/zowe`. The default value is the parent directory of the `bin` folder where this script is located.

- `-h <HashPgm-dir>`

Directory of the hash key program. The typical value is `/usr/lpp/zowe/fingerprint`. The default value is the `fingerprint` directory of the parent folder where this script is located.

- `-f <HashRef-dir>`

Directory of the reference hash key file `RefRuntimeHash-v.v.p.txt`. The typical value and default value are the same as that of the `-h` parameter. The values specified for `-h` and `-f` can be the same or different.

- `-l <output-dir>`

Output directory where the following log and output files will be written.

```
zowe-verify-authenticity.log
CustRuntimeHash.sort
CustRuntimeHash.txt
```

`RefRuntimeHash.sort`

The typical value is `~/zowe/fingerprint`. The directory will be created if you specify it but it does not exist.

The following defaults will be tried in the listed order:

`/global/zowe/log`

`~/zowe`

`$TMPDIR`

`/tmp`

Use of `zowe-verify-authenticity.sh` by `zowe-support.sh`

Starting in Zowe v1.14, the `zowe-verify-authenticity.sh` script is automatically called, with no parameters, by `zowe-support.sh`.

Troubleshooting installation and startup of Zowe z/OS components

The following topics contain information that can help you troubleshoot problems when you encounter unexpected behavior installing Zowe z/OS components or starting Zowe's `ZWESVSTC` started task.

How to check if `ZWESVSTC` startup is successful

The `ZWESVSTC` started task on z/OS brings up a number of address spaces. There is no single **Zowe has launched and is ready to run** message as the sequence of address spaces initialization is environment-dependent, although the message ID `ZWED0021I` is typically the last one that is logged. More details on each subsystem and their startup messages are described in the following sections.

- [Check the startup of API Mediation Layer](#) on page 407
- [Check the startup of Zowe Desktop](#) on page 408
- [Check the startup of Zowe File and Jobs API servers](#) on page 409
- [Check the startup of Zowe Secure Services](#) on page 409

To check that Zowe has started successfully, the most complete way is to check that each component successfully completed its initialization. Each component writes messages to the JES `STDOUT` and writes severe errors to the `STDERR` job spool file.

To learn more about the Zowe components and their role, see [Zowe architecture](#) on page 13. It is possible to configure Zowe to bring up only a subset of its components by using the `LAUNCH_COMPONENT_GROUPS` variable in the `instance.env` file. See [Component groups](#) on page 151 for more information.

To monitor `ZWESVSTC` to check whether each component has launched successfully, you can use one of the following ways:

- A good approach is to look at the active address spaces by using a command such as `DA` in SDSF. Each address space is named to identify its component, see [Address space names](#) on page 152.
- You can also look for particular messages in the `STDOUT` Job spool file.

Check the startup of API Mediation Layer

The API Mediation Layer has three address spaces: API Catalog `ZWE1AC`, API Gateway `ZWE1AG`, and API Discovery `ZWE1AD`. These might have been changed from their defaults. For more information, see [Address space names](#) on page 152.

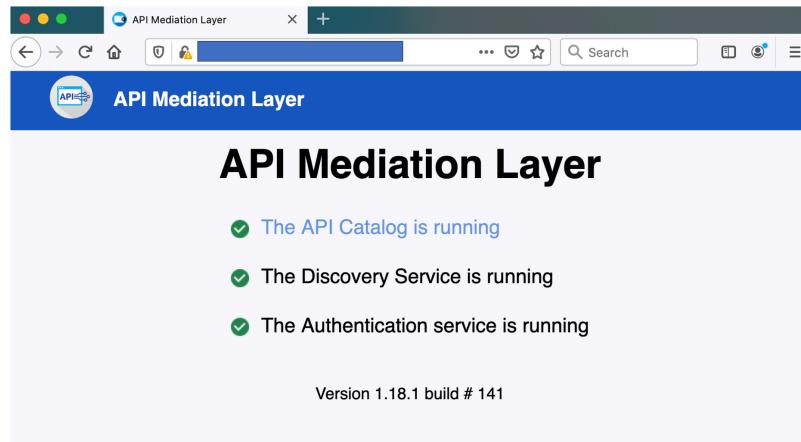
To check whether the API mediation layer is fully initialized, you can look for the ZWEAM000I message. Each component writes a successful startup message ZWEAM000I to the JES as shown below. The message also indicates the CPU of seconds spent. Check that each address space has written this message.

```
021-01-12 17:48:23.738 <ZWEADS1:main:33557015> ZWESVUSR INFO
(o.z.a.p.s.ServiceStartupEventHandler) `ZWEAM000I` Discovery Service
started in 97.725 seconds
...
2021-01-12 17:48:30.145 <ZWEAGW1:main:50334212> ZWESVUSR INFO
(o.z.a.p.s.ServiceStartupEventHandler) `ZWEAM000I` Gateway Service started
in 104.248 seconds
...
2021-01-12 17:48:31.036 <ZWEAAC1:main:33557009> ZWESVUSR INFO
(o.z.a.p.s.ServiceStartupEventHandler) `ZWEAM000I` API Catalog Service
started in 105.127 seconds
```

As well as looking for ZWEAM000I in the JES log, you can also log in to the gateway homepage and check the service status indicator. If there is a red or yellow tick beside one of its three services, the components are still starting.

The image contains two side-by-side screenshots of a web browser window titled "API Mediation Layer". Both screenshots show the same basic layout with a blue header bar containing the title and a white content area. In the left screenshot, there are three red warning icons followed by text: "The API Catalog is not running", "The Discovery Service is not running", and "The Authentication service is not running". In the right screenshot, the first icon is green (checkmark) and the other two are red. At the bottom of both screenshots, the text "Version 1.18.1 build # 141" is visible.

When all services are fully initialized, there will be three green ticks.



Check the startup of Zowe Desktop

The Zowe Desktop address space is named ZWE1DS1. During its initialization process, the desktop loads its plug-ins and writes a message ZWED0031I when it is completed.

```
2021-01-22 14:35:00.300 <ZWED:16842882> ZWESVUSR INFO
(_zsf.install,index.js:340) ZWED0031I - Server is ready at
https://0.0.0.0:8554, Plugins successfully loaded: 100% (21/21)
```

The ZWED0031I message includes a count of the number of loaded plug-ins as well as the total number of plug-ins, for example Plugins successfully loaded: 100% (21/21). A failed plug-in load will not abort the launch of the desktop.

In the preceding message example, it indicates that 21 plug-ins have loaded successfully. Each of these plug-ins writes its individual message ZWED0290I, so ZWED0031I indicates 21 plug-ins have loaded and there will be 21 instances of ZWED0290I, for example:

```
2021-01-22 14:34:58.762 <ZWED:16842882> ZWESVUSR INFO
(_zsf.install,index.js:263) ZWED0290I - Plugin (org.zowe.zosmf.workflows)
loaded. Successful: 5% (1/21) Attempted: 5% (1/21)
...
2021-01-22 14:35:00.114 <ZWED:16842882> ZWESVUSR
INFO (_zsf.install,index.js:263) ZWED0290I - Plugin
(org.zowe.zlux.appmanager.app.propview) loaded. Successful: 52% (11/21)
Attempted: 52% (11/21)
...
2021-01-22 14:35:00.278 <ZWED:16842882> ZWESVUSR INFO
(_zsf.install,index.js:263) ZWED0290I - Plugin (org.zowe.api.catalog)
loaded. Successful: 95% (20/21) Attempted: 95% (20/21)
```

Messages for ZWED0290I will be written when the JES Explorer org.zowe.explorer-jes, the MVS Explorer org.zowe.explorer-mvs, and the USS Explorer org.zowe.explorer-uss are loaded.

When the Zowe desktop and the API Gateway are both started in a launch configuration, it will register itself with the API Gateway after the Zowe desktop has started. This step must be completed before a user is able to successfully log in, as the API Mediation layer is used as the backing authentication service. The message that is written to indicate that the registration has been successful is ZWED0021I, for example

If you try to log into the Zowe desktop too early before the Eureka client registration has occurred you may get an **Authentication failed** message on the login page because the APIML handshake is incomplete. If this occurs wait for the registration to be complete as indicated by the ZWED0021I message.

Check the startup of Zowe File and Jobs API servers

Zowe has two servers that are used to provide API services for jobs and files. The Jobs API server address space is named ZWE1EF and the Files API server address space is named ZWE1EJ. When these have successfully started, a message Started <name> in <nn> seconds (JVM running for <nn>) is written to the log, for example:

```
2021-01-22 14:35:17.869 <ZWEAAJ1:main:50397279> ZWESVUSR INFO
(o.z.j.JesJobsApplication,StartupInfoLogger.java:59) Started
JesJobsApplication in 29.867 seconds (JVM running for 37.296)
...
2021-01-22 14:35:21.567 <ZWEAAD1:main:67174496> ZWESVUSR INFO
(o.z.DataSetsAndUnixFilesApplication,StartupInfoLogger.java:59) Started
DataSetsAndUnixFilesApplication in 33.597 seconds (JVM running for 41.002)
```

Check the startup of Zowe Secure Services

The zssServer is used for secure services for the Zowe desktop.

```
ZWES1013I ZSS Server has started. Version '1.18.0+20201214'
```

The zssServer will register itself with the cross memory server running under the address space ZWESISTC. You can use the attach message ID ZWES1014I to check that this has occurred successfully. If this message contains a

nonzero return code in the cmsRC= value, then a failure occurred. For more information on how to diagnose these, see [ZSS server unable to communicate with X-MEM](#) on page 440.

```
ZWES1014I ZIS status - 'Ok' (name='ZWESIS_STD', cmsRC='0',
description='Ok', clientVersion='2')
```

Unable to launch Zowe with

When you run `zowe-start.sh` from a unix shell path `<zowe-instance-directory>/bin`, you encounter the following error:

```
<RUNTIME_DIRECTORY>/scripts/internal/opercmd: ./zowe-start.sh 6: FSUM7351
not found
```

This can be because the value of `ROOT_DIR` in the `<zowe-instance-directory>/instance.env` file is pointing to an invalid Zowe runtime. This can occur in scenarios where the Zowe runtime directory was removed during an upgrade of a convenience build, and the `instance.env` file's `ROOT_DIR` value was not updated to point to the new fully qualified path for the new Zowe runtime.

This errors can also occur if the user ID running `zowe-start.sh` does not have read and traverse access to the directory tree ancestors of the `ROOT_DIR` itself. For example, if `ROOT_DIR` is set to `/usr/lpp/zowe` then the TSO user executing `zowe-start.sh` must have `rx` access to the directories `usr/lpp/zowe`, `usr/lpp` and `usr`. To see the access for a directory issue the unix command `ls -alT`. If you do not wish to open up `rx` access to the directory tree ancestors of the `ROOT_DIR` then Zowe can still be launched using a TSO command, see [Option 2: Starting Zowe with a /S TSO command](#) on page 157.

Unable to create BPXAS instances

Symptom:

When you start `ZWESVSTC` started task, either by running the `zowe-start.sh` script or by launching the started task directly, you encounter the following error in the log:

```
<ROOT_DIR>/bin/internal/run-zowe.sh 1: FSUM7726 cannot fork: reason code =
094500f7: EDC5112I Resource temporarily unavailable.
```

You will also encounter the following messages in the SYSLOG:

```
0290  S ZWESVSTC
0281  $HASP100 ZWESVSTC ON STCINRDR
0290  IEF695I START ZWESVSTC WITH JOBNAM ZWESVSTC IS ASSIGNED TO USER
      ZWESVUSR, GROUP ZWEADMIN
0281  $HASP373 ZWESVSTC STARTED
0090  IEA602I ADDRESS SPACE CREATE FAILED. MAXUSERS WOULD HAVE BEEN
      EXCEEDED
0290  BPXP005I A FORK OR SPAWN ERROR WAS ENCOUNTERED. RETURN CODE 00000070
      REASON CODE 094500F7
0090  IEA602I ADDRESS SPACE CREATE FAILED. MAXUSERS WOULD HAVE BEEN
      EXCEEDED
0090  IEA602I ADDRESS SPACE CREATE FAILED. MAXUSERS WOULD HAVE BEEN
      EXCEEDED
0090  IEA602I ADDRESS SPACE CREATE FAILED. MAXUSERS WOULD HAVE BEEN
      EXCEEDED
```

Solution:

This problem occurs when the maximum number of BPXAS instances have been reached.

This may be because when the Zowe instance directory was created, it was generated in the same location as the Zowe root directory. The Zowe instance directory is created by using the script <RUNTIME_DIR>/bin/zowe-configure-instance.sh -c <PATH_TO_INSTANCE_DIR>. See [Creating an instance directory](#) on page 150. The Zowe runtime directory is replaced when new PTFs are applied and should be considered as a read-only set of files. Zowe instance directories are designed to live outside the directory structure and are used to start a Zowe runtime.

This problem will only occur with Zowe drivers prior to v1.10 and has been resolved in v1.10 where the zowe-configure-instance.sh script will report error if it detects the -c argument because the installation directory location is an existing Zowe runtime directory.

Errors caused when running the Zowe desktop with node 8.16.1

Symptom:

When you start the ZWESVSTC started task, you encounter the following error messages:

```
/usr/lpp/zowe/components/app-server/share/zlux-app-server/lib/
initInstance.js:1
(function (exports, require, module, __filename, __dirname) {
SyntaxError: Invalid or unexpected token
    at createScript (vm.js:80:10)
    at Object.runInThisContext (vm.js:139:10)
    at Module._compile (module.js:617:28)
    at Object.Module._extensions..js (module.js:664:10)
    at Module.load (module.js:566:32)
    at tryModuleLoad (module.js:506:12)
    at Function.Module._load (module.js:498:3)
    at Function.Module.runMain (module.js:694:10)
    at startup (bootstrap_node.js:204:16)
    at bootstrap_node.js:625:3

/global/zowe/instances/prod/bin/internal/run-zowe.sh 3: FSUM7332 syntax
error: got ), expecting Newline
```

Solution:

This problem occurs when you use Node.js v8.16.1 which is not supported on Zowe. There is a known issue with node.js v8.16.1 and Zowe desktop encoding. Use a supported version of Node.js instead. For more information, see [Supported Node.js versions](#) on page 76.

Cannot start Zowe and UNIX commands not found with FSUM7351

Symptom:

When you start the ZWESVSTC started task, you might encounter the following error message:

```
dirname: <instance-dir>/bin/internal/run-zowe.sh 2: FSUM7351 not found
pwd: <instance-dir>/bin/internal/run-zowe.sh 2: FSUM7351 not found
.: <instance-dir>/bin/internal/run-zowe.sh 3: /bin/internal/read-
instance.sh: not found
```

Solution:

Check that /bin is part on your PATH. Do echo \$PATH to check. If it is missing, make sure that it is appended to PATH in your profile, for example, in /etc/profile/.

Various warnings show when connecting Zowe with another domain

Symptoms:

When you configure the Zowe environment variable `ZOWE_EXPLORER_HOST` in `instance.env` with a domain (for example, `domain-a.com`), and access Zowe with another domain (for example, `domain-b.com`), you may see the following errors:

- Certificate warnings similar to the following one:

```
domain-b.com:8544 uses an invalid security certificate.  
The certificate is only valid for the following names: domain-a.com, <ip-of-domain-a>, localhost.localdomain, localhost, 127.0.0.1
```

- No pinned applications show in Zowe Desktop.
- JES Explorer, MVS Explorer, USS Explorer may show errors similar to the following one if you ignore the certificate error.

```
Blocked by Content Security Policy  
An error occurred during a connection to domain-a.com:7554.  
Firefox prevented this page from loading in this way because the page has a content security policy that disallows it.
```

The above warnings and errors will also show when you plan to use Zowe with multiple domain names.

Solutions:

You can take the following steps:

- When you prepare the `bin/zowe-setup-certificates.env` file, specify the `HOSTNAME=` and `IPADDRESS=` parameters to accept multiple domains separated by comma (from Zowe v1.14.0). The following configuration is an example:

```
HOSTNAME=domain-a.com, domain-b.com  
IPADDRESS=<ip-of-domain-a>, <ip-of-domain-b>
```

Then you can proceed to run the `bin/zowe-setup-certificates.sh` script.

- After you run the `bin/zowe-configure-instance.sh` script, modify the `instance.env` file located in the `instance` directory in the following ways to reflect the multiple domains you plan to use.
 - Add a line of `ZWE_EXTERNAL_HOSTS`. For example, `ZWE_EXTERNAL_HOSTS=domain-a.com, domain-b.com`.
 - Add a line of `ZWE_REFERRER_HOSTS`. For example, `ZWE_REFERRER_HOSTS=domain-a.com, domain-b.com`.
 - Find the line that starts with `ZOWE_EXPLORER_FRAME_ANCESTORS` and modify its values to `ZOWE_EXPLORER_FRAME_ANCESTORS=" ${ZOWE_EXPLORER_HOST} : * , domain-a.com: * , domain-b.com: * , ${ZOWE_IP_ADDRESS} : * "`.

Drawback:

With this change, you must use the API Mediation Layer Gateway port (default is 7554) to access Zowe Desktop, for example, `https://domain-a.com:7554/ui/v1/zlux` or `https://domain-b.com:7554/ui/v1/zlux`. Using Desktop port (default is 8544) like `https://domain-b.com:8544/` is not supported.

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 `instance.env`.
2. Find the line that contains the `APIML_DEBUG_MODE_ENABLED`= parameter and set the value to `true`:

```
APIML_DEBUG_MODE_ENABLED=true
```

By default debug mode is disabled, so the `APIML_DEBUG_MODE_ENABLED` is set to `false`.

3. Restart Zowe™.

You enabled debug mode for the API ML core services (API Catalog, API Gateway and Discovery Service).

4. (Optional) Reproduce a bug that causes issues and review debug messages. If you are unable to resolve the issue, create an issue [here](#).

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 HTTPPie tool: <https://httpie.org/>.

Example:

```
HTTPPie 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).
```

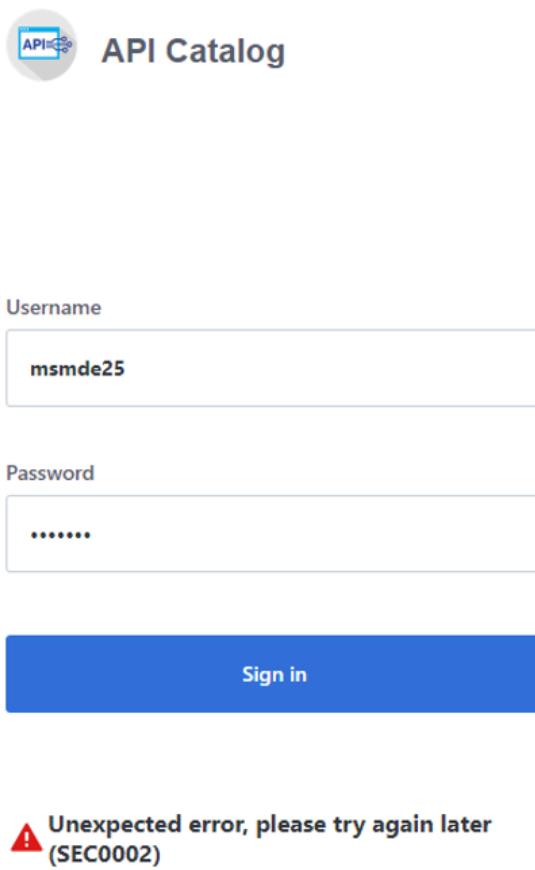
Solution:

Restart API Mediation Layer.

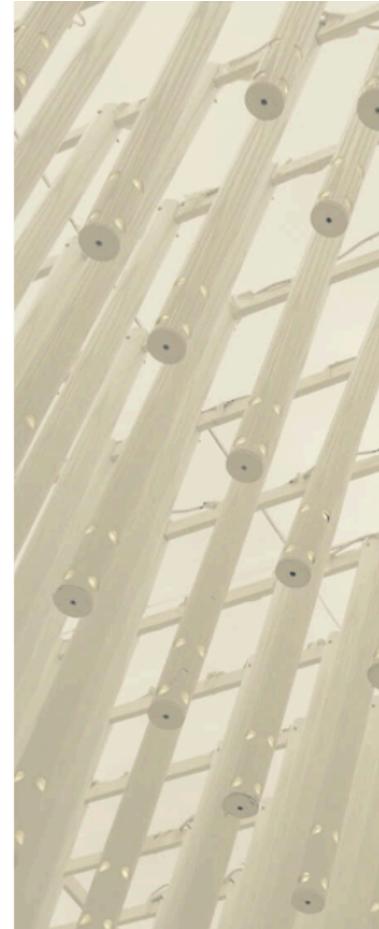
Tip: To prevent this issue from occurring, it is strongly recommended not to restart the TCP/IP stack while API ML is running.

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 screenshot shows the API Catalog login interface. At the top left is the API Catalog logo. Below it is a form with two input fields: 'Username' containing 'msmde25' and 'Password' containing several dots. A large blue 'Sign in' button is centered below the fields. At the bottom of the form, there is an error message: **⚠ Unexpected error, please try again later (SEC0002)**.



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 416
- [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 417

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. Re-create the Zowe keystore by deleting it and re-creating it. For more information, see [Configuring Zowe certificates](#) on page 136. The Zowe keystore directory is the value of the `KEYSTORE_DIRECTORY` variable in the `instance.env` file in the instance directory that is used to launch Zowe. See [Keystore configuration](#) on page 152 for more information.

Insecure fix

Follow these steps:

1. Re-create the Zowe keystore by deleting it and re-creating it. For more information, see [Configuring Zowe certificates](#) on page 136. In the `zowe-setup-certificates.env` file that is used to generate the keystore, ensure that the property `VERIFY_CERTIFICATES` is set to FALSE.

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 418
- [Re-create the Zowe keystore](#) on page 418

Request a new certificate

Request a new certificate that contains a valid z/OSMF host name in the subject alternative names.

Re-create the Zowe keystore

Re-create the Zowe keystore by deleting it and re-creating it. For more information, see [Configuring Zowe certificates](#) on page 136. The Zowe keystore directory is the value of the KEYSTORE_DIRECTORY variable in the instance.env file in the instance directory that is used to launch Zowe. See [Keystore configuration](#) on page 152.

API ML throws I/O error on GET request and cannot connect to other services

Symptom:

The API ML services are running but they are in DOWN state and not working properly. The following exceptions can be found in the log: java.net.UnknownHostException and java.net.NoRouteToHostException.

Sample message:

See the following message for full exceptions.

```
org.springframework.web.client.ResourceAccessException: I/O error
on GET request for "https://USILCA32.lvn.broadcom.net:7553/eureka/
apps/apicatalog": USILCA32.lvn.broadcom.net; nested exception is
java.net.UnknownHostException: USILCA32.lvn.broadcom.net

.at
org.springframework.web.client.RestTemplate.doExecute(RestTemplate.java:732)
~\Spring-web-5.0.8.RELEASE.jar!/:5.0.8.RELEASE"

.at
org.springframework.web.client.RestTemplate.execute(RestTemplate.java:680)
~\Spring-web-5.0.8.RELEASE.jar!/:5.0.8.RELEASE"

.at
org.springframework.web.client.RestTemplate.exchange(RestTemplate.java:600)
~\Spring-web-5.0.8.RELEASE.jar!/:5.0.8.RELEASE"

.at
com.ca.mfaas.apicatalog.services.initialisation.InstanceRetrievalService.queryDiscovery
Yclasses!/:na"
```

```

.at
com.ca.mfaas.apicatalog.services.initialisation.InstanceRetrievalService.getInstanceStateIn
Ýclasses!/:na"

.at
com.ca.mfaas.apicatalog.services.initialisation.InstanceRetrievalService.retrieveAndRe
Ýclas

...
main" o.a.http.impl.client.DefaultHttpClient : I/O exception
  (java.net.NoRouteToHostException) caught when connecting to {s}->https://
localhost:7553: EDC8130I Host cannot be reached. (Host unreachable)

main" o.a.http.impl.client.DefaultHttpClient : Retrying connect to {s}-
>https://localhost:7553

```

Solution:

The Zowe started task needs to run under the same user ID as z/OSMF (typically IZUSVR). This is stated in the [installation documentation](#).

The hostname that is displayed in the details of the exception is a valid hostname. You can validate that the hostname is valid by using ping command on the same mainframe system. For example, ping USILCA32.lvn.broadcom.net. If it is valid, then the problem can be caused by insufficient privileges of your started task that is not allowed to do network access.

You can fix it by setting up the security environment as described in the [Configure security environment switching](#) on page 129.

Certificate error when using both an external certificate and Single Sign-On to deploy Zowe

Symptom:

You used an external certificate and Single Sign-On to deploy Zowe. When you log in to the Zowe Desktop, you encounter an error similar to the following:

```

2020-07-28 02:13:43.203 <ZWED:262486> IZUSVR WARN
  (org.zowe.zlux.auth.safssso,apimlHandler.js:263) APIML query error: self
  signed certificate in certificate chain
2020-07-28 02:13:43.288 <ZWED:262486> IZUSVR WARN
  (org.zowe.zlux.auth.safssso,apimlHandler.js:337) APIML login has failed:
2020-07-28 02:13:43.288 <ZWED:262486> IZUSVR WARN
  (org.zowe.zlux.auth.safssso,apimlHandler.js:338) Error: self signed
  certificate in certificate chain
    at TLSSocket.onConnectSecure (_tls_wrap.js:1321:34)
    at TLSSocket.emit (events.js:210:5)
    at TLSSocket._finishInit (_tls_wrap.js:794:8)
    at TLSWrap.ssl.onhandshakedone (_tls_wrap.js:608:12) {
  code: 'SELF_SIGNED_CERT_IN_CHAIN'
}

```

Solution:

This issue might occur when you use a Zowe version of 1.12.0 or later. To resolve the issue, you can download your external root certificate and intermediate certificates in PEM format. Then, add the following parameter in the Zowe instance.env file.

```
ZWED_node_https_certificateAuthorities="/path/to/zowe/keystore/
local_ca/localca.cer-ebcdic","/path/to/carootcert.pem","/path/to/
caintermediatecert.pem"
```

Recycle your Zowe server. You should be able to log in to the Zowe Desktop successfully now.

Browser unable to connect due to a CIPHER error**Symptom:**

When connecting to the API Mediation Layer, the web browser throws an error saying that the site is unable to provide a secure connection because of an error with ciphers.

The error shown varies depending on the browser. For example,

- For Google Chrome:

This site can't provide a se

9.20.205.190 uses an unsupported prot

ERR_SSL_VERSION_OR_CIPHER_MISMATCH

- For Mozilla Firefox:



An error occurred during a connection to 9.20.205.110:7554. Common encryption algorithm(s).

Error code: SSL_ERROR_NO_CYPHER_OVERLAP

- The page you are trying to view cannot be shown because verified.
- Please contact the website owners to inform them of this p

Solution:

Remove GCM as a disabled TLS algorithm from the Java runtime being used by Zowe.

To do this, first locate the \$JAVA_HOME/lib/security/java.security file. You can find the value of \$JAVA_HOME in one of the following ways.

- Method 1: By looking at the JAVA_HOME= value in the instance.env file used to start Zowe.

For example, if the instance.env file contains the following line,

```
JAVA_HOME=/usr/lpp/java/J8.0_64/
```

then, the \$JAVA_HOME/lib/security/java.security file will be /usr/lpp/java/J8.0_64/lib/security/java.security.

- Method 2: By inspecting the STDOUT JES spool file for the ZWESVSTC started task that launches the API Mediation Layer.

In the java.security file, there is a parameter value for jdk.tls.disabledAlgorithms, for example,

```
jdk.tls.disabledAlgorithms=SSLv3, RC4, MD5withRSA, DH keySize < 1024,  
3DES_EDE_CBC, DESede, EC keySize < 224, GCM
```

Note: This line may have a continuation character \ and be split across two lines due to its length.

Edit the parameter value for jdk.tls.disabledAlgorithms to remove GCM. If as shown above the line ends <224, GCM, remove the preceding comma so the values remain a well-formed list of comma-separated algorithms:

```
jdk.tls.disabledAlgorithms=SSLv3, RC4, MD5withRSA, DH keySize < 1024,  
3DES_EDE_CBC, DESede, EC keySize < 224
```

Note: The file permissions of java.security might be restricted for privileged users at most z/OS sites.

After you remove GCM, restart the ZWESVSTC started task for the change to take effect.

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 utility messages

ZWEAM000I

%s started in %s seconds

Reason:

The service started.

Action:

No action required.

API mediation common messages

ZWEAO102E

Gateway not yet discovered. The Transform service cannot perform the request

Reason:

The Transform service was requested to transform a url, but the Gateway instance was not 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.

ZWEAO105W

Gateway HTTP Client per-route connection limit of %s has been reached for the '%s' route.

Reason:

Too many concurrent connection requests were made to the same route.

Action:

Further connections will be queued until there is room in the connection pool. You may also increase the per-route connection limit via the gateway start-up script by setting the Gateway configuration for maxConnectionsPerRoute.

ZWEAO106W

Gateway Http Client total connection limit of %s has been reached.

Reason:

Too many concurrent connection requests were made.

Action:

Further connections will be queued until there is room in the connection pool. You may also increase the total connection limit via the gateway start-up script by setting the Gateway configuration for maxTotalConnections.

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

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:

The Message service is requested to create a 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:

The 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:

- An 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`, and then restart the service. Do not use this option in a production environment.

ZWEAM501W

Service is connecting to Discovery service using the non-secure 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:

The Gateway does not trust the requested service and refuses to communicate with it.

Action:

Possible actions regarding to message content:

- Message: Certificate does not match any of the subject alternative names. Action: Verify that the hostname which the certificate is issued for matches the hostname of the service.
- Message: Unable to find the valid certification path to the requested target. Action: Import the root CA that issued services's certificate to API Gateway truststore

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:

The request failed because of an internal error.

Action:

Refer to specific exception details for troubleshooting. Create an issue with this message.

Security common messages**ZWEAT100E**

Token is expired for URL '%s'

Reason:

The validity of the token is expired.

Action:

Obtain a new token by performing an authentication request.

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 z/OSMF 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.

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:

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.

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 is invalid.

Action:

Provide a valid username and password.

ZWEAS121E

Authorization header is missing, or the 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.

ZAAS client messages

ZWEAS100E

Token is expired for URL

Reason:

The application using the token kept it for longer than the expiration time

Action:

When this error occurs it is necessary to get a new JWT token.

ZWEAS120E

Invalid username or password

Reason:

Provided credentials weren't recognized

Action:

Try with different credentials

ZWEAS121E

Empty or null username or password values provided

Reason:

One of the credentials was null or empty

Action:

Try with full set of credentials

ZWEAS122E

Empty or null authorization header provided

Reason:

The authorization header was empty or null.

Action:

Try again with a valid authorization header.

ZWEAS170E

An exception occurred while trying to get the token

Reason:

General exception. There is more information in the message.

Action:

Log the message from the exception and then handle the exception based on the information provided there.

ZWEAS400E

Unable to generate PassTicket. Verify that the secured signon (PassTicket) function and application ID is configured properly by referring to Using PassTickets in the guide for your security provider

Reason:

Unable to generate a PassTicket.

Action:

Verify that the secured signon (PassTicket) function and application ID is configured properly by referring to Using PassTickets in the guide for your security provider.

ZWEAS401E

Token is not provided

Reason:

There was no JWT token provided for the generation of the PassTicket.

Action:

Ensure that you are passing a JWT token for PassTicket generation.

ZWEAS404E

Gateway service is unavailable

Reason:

The Gateway Service does not respond.

Action:

Ensure that the Gateway Service is up and that the path to the gateway service is properly set.

ZWEAS417E

The application name wasn't found

Reason:

The application id provided for the generation of the PassTicket was not recognized by the security provider.

Action:

Ensure that the security provider recognized the application id.

ZWEAS130E

Invalid token provided

Reason:

The JWT token is not valid.

Action:

Provide a valid token.

ZWEAS500E

There was no path to the trust store.

Reason:

The Zaas Client configuration does not contain the path to the trust store.

Action:

Ensure that the configuration contains the trustStorePath and that it points to valid trust store.

ZWEAS501E

There was no path to the key store.

Reason:

The Zaas Client configuration does not contain the path to the key store.

Action:

Ensure that the configuration contains the keyStorePath and that it points to valid key store.

ZWEAS502E

The configuration provided for SSL is invalid.

Reason:

The type of the keystore, truststore or the included keys/certs aren't considered valid

Action:

Ensure that the combination of the configuration is cryptographically valid.

ZWEAS503E

The SSL configuration contained invalid path.

Reason:

There was an invalid path to either trust store or keystore

Action:

Ensure that both provided paths are resolved to valid trust store and valid key store.

Discovery service messages**ZWEAD400E**

Cannot notify Gateway on '%s' about new instance '%s'

Reason:

The Discovery Service tried to notify the Gateway about an instance update, but the REST call failed. The purpose of this call is to update the Gateway caches. The Gateway might be down or a network problem occurred.

Action:

Ensure that there are no network issues and that the Gateway was not restarted. If the problem reoccurs, contact Broadcom support.

ZWEAD401E

Cannot notify Gateway on '%s' about cancelled registration

Reason:

The Discovery Service tried to notify the Gateway about service un-registration, but the REST call failed. The purpose of this call is to update the Gateway caches. The Gateway might be down or a network problem occurred.

Action:

Ensure that there are no network issues and that the Gateway was not restarted. If the problem reoccurs, contact Broadcom support.

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' - '%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 determine 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 instanceBaseUrl 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 a valid hostname.
- Invalid protocol. The specified protocol of the URL is invalid. Make sure to specify valid protocol.
- Additional service metadata of %s in processing file %s could not be created: %s

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`

ZWEAD704E

Gateway Service is not available so it cannot be notified about changes in Discovery Service

Reason:

Gateway Service is probably mis-configured or failed to start from another reason.

Action:

Review the log of Gateway Service and its configuration.

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.

ZWEAG701E

Service '%s' does not allow encoded characters in the 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 request enablement of encoded characters in the service.

ZWEAG702E

Gateway does not allow encoded slashes in request: '%s'.

Reason:

The request that was issued to the Gateway contains an encoded slash in the URL path. Gateway configuration does not allow this encoding in the URL.

Action:

Contact the system administrator and request enablement of encoded slashes in the Gateway.

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.

ZWEAG706E

RequestContext is not prepared for load balancing.

Reason:

Custom Ribbon load balancing is not in place before calling Ribbon.

Action:

Contact Broadcom support.

ZWEAG707E

The request to the URL '%s' aborted without retrying on another instance. Caused by: %s

Reason:

Request to the server instance failed and will not be retried on another instance.

Action:

Refer to Caused by details for troubleshooting.

ZWEAG708E

The request to the URL '%s' failed after retrying on all known service instances. Caused by: %s

Reason:

Request to the server instance could not be executed on any known service instance.

Action:

Verify the status of the requested instance.

ZWEAG709E

Service is not available at URL '%s'. Error returned: '%s'

Reason:

The service is not available.

Action:

Make sure that the service is running and is accessible by the URL provided in the message.

ZWEAG100E

Authentication exception: '%s' for URL '%s'

Reason:

A generic failure occurred during authentication.

Action:

Refer to the 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

The token has 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 dummy mode. Use credentials '%s'/'%s' to log in. Do not use this option in the production environment.

Reason:

The authentication is running in dummy mode.

Action:

Ensure that this option is not being used in a production environment.

ZWEAG107W

Incorrect value: apiml.security.auth.provider = '%s'. The authentication provider is not set correctly. The default 'zosmf' authentication provider is being 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 and/or password are invalid.

Action:

Provide a valid username and password.

ZWEAG121E

Authorization header is missing, or the 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.

ZWEAG140E

The 'applicationName' parameter name is missing.

Reason:

The application name is not provided.

Action:

Provide the 'applicationName' parameter.

ZWEAG141E

The generation of the PassTicket failed. Reason: %s

Reason:

An error occurred in the SAF Auth Service. Review the reason in the error message.

Action:

Supply a valid user and application name, and check that corresponding permissions have been set up.

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:

The status of 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.

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 '%s' service. %s

Reason:

ApiDoc for service could not be retrieved.

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. The 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 ... The 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.

ZWEAC707E

Static API refresh failed, caused by exception: %s

Reason:

The Static API refresh could not be performed because of exception.

Action:

Check the specific exception for troubleshooting.

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 which includes the Zowe Desktop.

Most of the solutions below identify issues by referring to the [Gathering information to troubleshoot Zowe Application Framework](#) on page 445. To identify and resolve issues, you should be familiar with their names and locations.

The Zowe Application Framework manages issues in GitHub. When you troubleshoot a problem, you can check whether a GitHub issue (open or closed) that covers the problem already exists. For a list of issues, see the [zlux repo](#).

Desktop apps fail to load

Symptom:

When you open apps in the Zowe desktop, a page is displayed with the message "The plugin failed to load."

Solution:

This problem might occur when you use Node.js v8.16.1, which performs auto-encoding in a way that breaks Zowe apps. See <https://github.com/ibmruntimes/node/issues/142> for details.

To solve the problem, use a different version of Node.js v8, such as v8.17.0, or use Node.js v12. You can obtain them from the [Node.js marketplace](#). Download the `ibm-trial-node-v8.17.0-os390-s390x.pax.Z` file.

NODEJSAPP disables immediately

Symptom:

You receive the message CEE5207E The signal SIGABRT was received in stderr.

Solution:

You might have reached the limit for shared message queues on your LPAR. When Node.js applications are terminated by a SIGKILL signal, shared message queues might not be deallocated. For more information, see the **If the NODEJSAPP disables immediately** section in the [Troubleshooting Node.js applications](#) topic on IBM Knowledge Center.

Cannot log in to the Zowe Desktop

Symptom:

When you attempt to log in to the Zowe Desktop, you receive the following error message that is displayed beneath the **Username** and **Password** fields.

```
Authentication failed for 3 types:  Types: [ "saf", "apiml", "zss" ]
```

The Zowe desktop attempts to authenticate the credentials using the types that have been configured, by default the three above of ["saf", "apiml", "zss"]. If Zowe has been configured with the

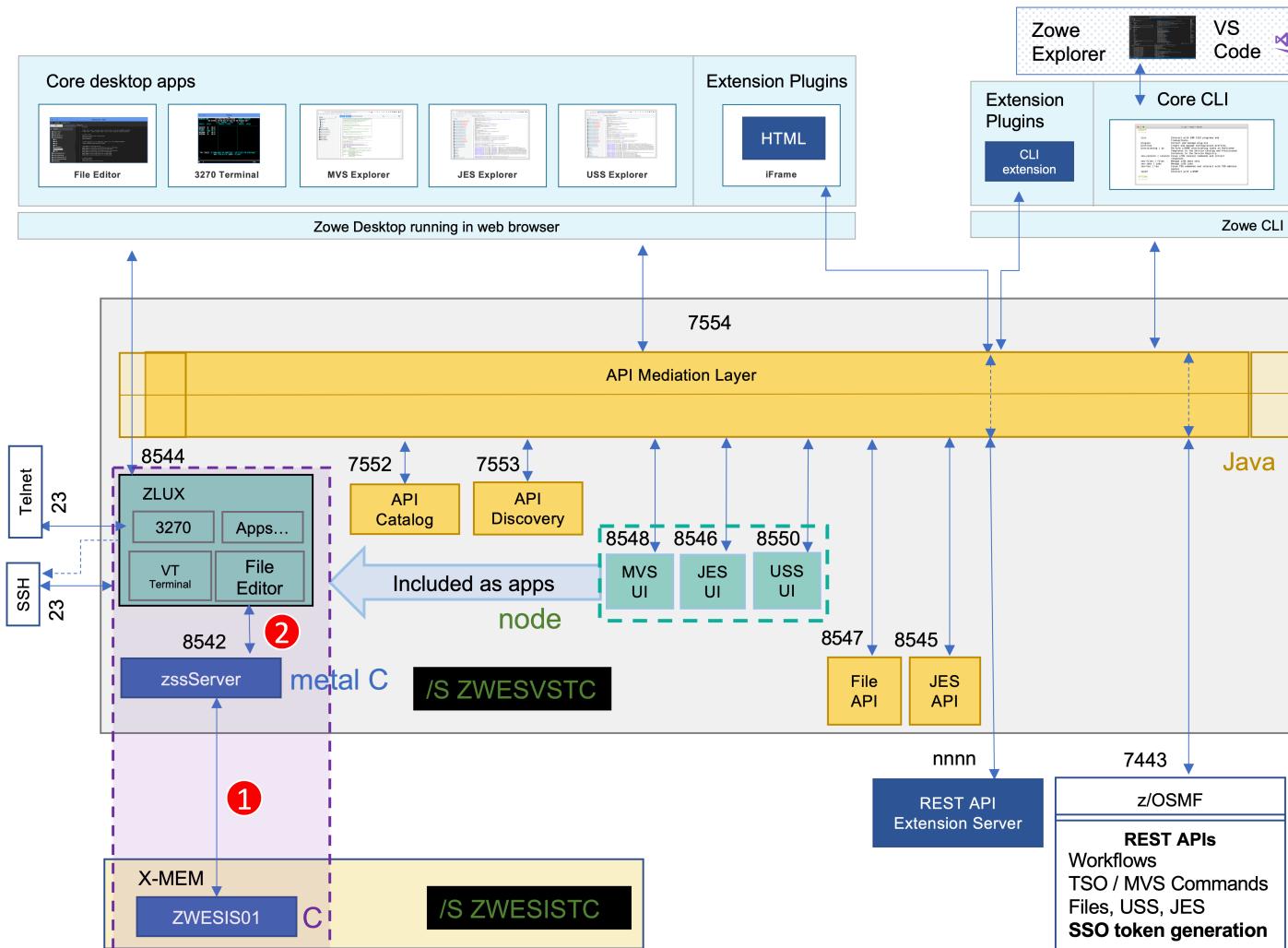
LAUNCH_COMPONENT_GROUPS=DESKTOP where GATEWAY is not a launch group, then the message will just include the types ["saf" , "zss"].

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.

There are two known problems that might cause this error. The [Zowe architecture](#) on page 13 shows the following connections. One of these two connections likely failed.

1. The zssServer connection to the ZWESISTC started task using cross memory communication. If this fails, see [ZSS server unable to communicate with X-MEM](#) on page 440. The architecture diagram below has been annotated with a (1) to show this connection.
2. The Zowe Desktop ZLUX server connection to the zssServer across the default port 8542. If this fails, see [ZLUX unable to communicate with zssServer](#) on page 442. The architecture diagram below has been annotated with a (2) to show this connection.



ZSS server unable to communicate with X-MEM

- Open the log file `$INSTANCE_DIR/logs/zssServer-yyyy-mm-dd-hh-ss.log`. This file is created each time ZWESVSTC is started and only the last five files are kept.

- Look for the message that starts with ZIS status.
 - If the communication works, the message includes Ok. For example:

```
ZIS status - Ok (name='ZWESIS_STD' , cmsRC=0, description='Ok')
```

If the communication works, the problem is likely that the ZLUX server is unable to communicate to the zssServer. For more information, see [ZLUX unable to communicate with zssServer](#) on page 442.

- If the communication is not working, the message includes Failure. For example:

```
ZIS status - Failure (name='ZWESIS_STD' , cmsRC=39, description='Cross-memory call ABENDED')
```

or

```
ZIS status - Failure (name='ZWESIS_STD' , cmsRC=64, description='N/A', clientVersion='2')
```

or

```
ZIS status - 'Failure' (name='ZWESI_STD' , cmsRC='12', description='N/A', clientVersion='2')
```

In this case, check that the ZWESISTC started task is running. If not, start it with the TSO command /S ZWESISTC

- If the problem cannot be easily fixed (such as the ZWESISTC task not running), then it is likely that the cross memory server is not running. To check whether the cross memory is running, check the started task ZWESISTC log for any errors.
- If the cross memory server ZWESISTC started task is running, check that the program name of the cross memory procedure matches between the ZWESISTC PROBLIB member and the instance.env file used to launch Zowe.

By default the proc value is ZWESIS_STD, and if a new name is chosen then both files need to be updated for the handshake to be successful.

The line in the ZWESISTC proplib that defines the procedure name that cross memory will use is

```
//ZWESISTC PROC NAME='ZWESIS_STD' ,MEM=00 ,RGN=0M
```

The line in the instance.env that specifies the cross memory procedure that the zssServer will try to attach to is

```
ZOWE_ZSS_XMEM_SERVER_NAME=ZWESIS_STD
```

- If this is the first time you set up Zowe, it is possible that the cross memory server configuration did not complete successfully. To set up and configure the cross memory server, follow steps as described in the topic [Installing and configuring the Zowe cross memory server \(ZWESISTC\)](#) on page 146. Once ZWESISTC

is started, if problems persist, check its log to ensure it has been able to correctly locate its load module ZWESIS01 as well as the parmlib ZWESIP00.

- 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 APP-authorized load module.

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 ZWESVUSR) 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 183.

ZLUX unable to communicate with zssServer

Follow these steps:

- Open the log file `$INSTANCE_DIR/logs/appServer-yyyy-mm-dd-hh-ss.log`. This file is created each time ZWESVSTC is started and only the last five files are kept.
- Look for the message that starts with `GetAddrInfoReqWrap.onlookup` and the log messages below.

```
yyyy-mm-dd hh:mm:ss.ms <ZUED:16842977> ZWESVUSR INFO (_zsf.apiml,apiml.
yyyy-mm-dd hh:mm:ss.ms <ZUED:16842977> ZWESVUSR INFO (_zsf.auth,webauth
yyyy-mm-dd hh:mm:ss.ms <ZUED:16842977> ZWESVUSR WARN (_zsf.proxy,proxy.
    at GetAddrInfoReqWrap.onlookup 'as oncomplete' (dns.js:64:26) {
        errno: 'ENOTFOUND',
        code: 'ENOTFOUND',
        syscall: 'getaddrinfo',
        hostname: 'localhost'
```

These messages show that the host name `localhost` cannot be reached between the Zowe desktop server and the zssServer because `localhost` has not been mapped to an IP address.

- Map `localhost` to port 127.0.0.1.

Create an entry in the file `/etc/hosts` that contains the line

```
127.0.0.1      localhost
```

- Restart the ZWESVSTC address space.

Server startup problem ret=1115

Symptom: When ZWESVSTC is restarted, the following message is returned in the output of the ZSS server log file, `$INSTANCE_DIR/logs/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.

Solution: To check whether 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 445 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 that is 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 that is built 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
```

Error: Zowe Desktop address space fails to start

After launching the started task ZWESVSTC there are no Zowe desktop ZWE1DS address space(s).

Symptom: Check the log for the message

```
ZWED0115E - Unable to retrieve storage object from cluster. This is probably due to a timeout.  
You may change the default of '5000' ms by setting 'node.cluster.storageTimeout' within the config. Timeout call null/clusterManager/getStorageAll
```

The timeout value was increased to be 30000 in 1.11.0 release. To check which release of Zowe you are running, see [Determining the Zowe release number](#). To further increase this, or update the value on a previous release you can add an entry to your \$INSTANCE_DIR/instance.env.

```
ZWED_node_cluster_storageTimeout=30000
```

where the timeout value is in milliseconds.

Warning: Problem making eureka request

Symptom: The Zowe started task ZWESVSTC log contains error messages reporting problems connecting

```
Problem making eureka request { Error: connect ECONNREFUSED 10.1.1.2:7553
at TCPConnectWrap.afterConnect [as oncomplete] (net.js:1195:14)
errno: 'ECONNREFUSED',
code: 'ECONNREFUSED',
syscall: 'connect',
address: '10.1.1.2',
port: 7553 }
```

Solution: You can ignore these messages. These messages are timing-related where different Eureka servers come up, try to connect to each other, and warn that the endpoint they are trying to perform a handshake with is not available. When all of the Eureka services have started, these errors will stop being logged.

Warning: ZWED0159W - Plugin (org.zowe.zlux.proxy.zosmf) loading failed.

Symptom: The Zowe started task ZWESVSTC log contains messages

```
ZWED0159W - Plugin (org.zowe.zlux.proxy.zosmf) loading failed.
Message: "ZWED0047E - Proxy (org.zowe.zlux.proxy.zosmf:data) setup failed.
Host & Port for proxy destination are required but were missing.
```

Solution: You can ignore these messages which should not occur in 1.11 or later releases. To check which release of Zowe you are running, see [Determining the Zowe release number](#).

Warning: ZWED0050W - Could not read swagger doc folder (..)

Symptom: The Zowe started task ZWESVSTC log contains messages ending

```
ZWED0050W - Could not read swagger doc folder <RUNTIME_DIR>/components/app-
server/share/zlux-workflow/doc/swagger
ZWED0050W - Could not read swagger doc folder <RUNTIME_DIR>/components/app-
server/share/zlux-app-manager/virtual-desktop/doc/swagger
ZWED0050W - Could not read swagger doc folder <RUNTIME_DIR>/components/app-
server/share/zlux-app-manager/bootstrap/doc/swagger
ZWED0050W - Could not read swagger doc folder <RUNTIME_DIR>/components/app-
server/share/zlux-server-framework/plugins/terminal-proxy/doc/swagger
ZWED0050W - Could not read swagger doc folder <RUNTIME_DIR>/components/app-
server/share/tn3270-ng2/doc/swagger
```

Solution: You can ignore these messages.

Warning: ZWED0047W - Swagger file for server (...) not found

Symptom:

The Zowe started task ZWESVSTC log contains messages ending

```
ZWED0047W - Swagger file for service (org.zowe.zosmf.workflows:zosmf) not
found
ZWED0047W - Swagger file for service (org.zowe.zlux.ng2desktop:browser-
preferences) not found
```

```
ZWED0047W - Swagger file for service
  (org.zowe.zlux.bootstrap:adminnotificationdata) not found
ZWED0047W - Swagger file for service (org.zowe.terminal.proxy:tn3270data)
  not found
ZWED0047W - Swagger file for service
  (org.zowe.terminal.tn3270:statediscovery) not found
```

Solution: You can ignore these messages.

Unable to log in to the explorers when using Zowe V1.13 or V1.14

Symptom:

You installed Zowe V1.13 or V1.14. When you start the Zowe server, you see the following error message in the appServer log:

```
failed to process config
TypeError: config.csp.frame-ancestors.split is not a function
```

When you log in to the Zowe Desktop, you cannot open the JES, MVS, or USS Explorers. You receive the following error message:

```
{ "messages" :
[ { "messageType" : "ERROR" , "messageNumber" : "ZWEAG708E" , "messageContent" : "The
request to the URL '/ui/v1/explorer-uss/' has failed after retrying
on all known service instances. Caused by: java.net.ConnectException:
EDC8128I Connection refused. (errno2=0x74940000) (Connection
refused)" , "messageKey" : "org.zowe.apiml.gateway.connectionRefused" } ] }
```

Solution:

A new property ZOWE_EXPLORER_FRAME_ANCESTORS was introduced in V1.12. This property is required to be present in the instance.env file with some valid value. When undefined, it is treated as Boolean, which breaks the string split function. To resolve the issue, define the value for this property in the instance.env file.

Gathering information to troubleshoot Zowe Application Framework

Gather the following information to troubleshoot ZoweTM Application Framework issues:

- [z/OS release level](#)
- [Zowe version and release level](#) on page 446
- [Zowe application configuration](#) on page 446
- [Zowe Application Server ports](#) on page 446
- [Log output from the Zowe Application Server](#) on page 447
- [Error message codes](#) on page 447
- [Javascript console output](#) on page 447
- [Screen captures](#) on page 447
- [Other relevant information](#) on page 447

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 [Configuring Zowe Application Framework](#) on page 179 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 193.

```
# 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
```

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 439 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/OSMF JVM cache corruption

Symptom:

When you work with Zowe, there are situations when z/OSMF abends.

The following is a snippet from the Java core dump.

```
CEE3DMP V2 R4.0: Condition processing resulted in the unhandled condition.

...
Condition Information for Active Routines
Condition Information for (DSA address 0000005F026FDE40)
CIB Address: 0000005F026FA1E8
Current Condition:
    CEE0198S The termination of a thread was signaled due to an
unhandled condition.
Original Condition:
    CEE3250C The system or user abend SDC2  R=4A001620 was issued.
Location:
    Program Unit: Entry: ntv_createJoinWorkUnit
    Statement: Offset: +000ABD14
Machine State:
    ILC..... 0002    Interruption Code..... 000D
    PSW..... 0785240180000000 000000003825D954
```

Solution:

The error occurs when the Java runtime being used by the z/OSMF Liberty server and the Java runtimes being used by Zowe share a user ID of `IZUSVR1`, which results in a collision. To resolve this issue, review the following steps.

1. [Isolate the started task user IDs](#) on page 448
2. [Update z/OSMF to not use JVM class caching](#) on page 449

Isolate the started task user IDs

The z/OSMF started task `IZUSVR1` runs under the user ID of `IZUUSER`. Before version 1.9 of Zowe, its started task `ZWESVSTC` also ran under the same user ID. With Zowe 1.9, the default configuration changed to use a new user ID of `ZWESVUSR` and group of `ZWEADMIN`.

If your started task `ZWESVSTC` is configured to run under the user ID `IZUUSER`, change it to run under user ID `ZWESVUSR`. For more information, see [User IDs and groups for the Zowe started tasks](#) on page 133.

Update z/OSMF to not use JVM class caching

If you need to run ZWESVSTC under the same user ID as z/OSMF for your environment, you can update the z/OSMF configuration to switch off shared class caching which stops the crash from occurring. Disabling shared class caching reduces the performance of z/OSMF so the preferred fix is to change the user ID of ZWESVSTC away from IZUUSER to ZWESVUSR as described above.

Navigate to the file `/var/zosmf/configuration/local_override.cfg`. This contains the startup arguments for the Java runtime used by z/OSMF. Add the following line:

```
JVM_OPTIONS=-Xshareclasses:none
```

You will need to recycle the z/OSMF server running, which by default will be running under the started task IZUSVR1.

For more information on the effect that disabling a shared class cache has on a Java runtime, see [Class data sharing](#) in the IBM Knowledge Center.

Unable to generate unique CeaTso APPTAG

Symptom:

When you request a Zowe data set or z/OS Files API, you receive a response code 500 - 'Internal Server Error', with a message "Unable to generate unique CeaTso APPTAG".

Solution:

Check z/OSMF settings of REST API of file. You must define RESTAPI_FILE in IZUPRMxx by the following statement:

```
RESTAPI_FILE ACCT(IZUACCT) REGION(32768) PROC(IZUFPROC)
```

The default IZUFPROC can be found in SYS1.PPROCLIB. And the proper authorization is needed to get IZUFPROC work successfully.

z/OS Services are unavailable

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
//ISPSSLIB 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 454 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 451.

The following information is also useful to collect:

- If you are experiencing HTTP errors, see [z/OSMF troubleshooting](#) on page 454 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
```

The exact Zowe CLI version may vary depending upon if the @latest or @zowe-v1-lts, or @lts-incremental version is installed.

For the @zowe-v1-lts and the @latest (forward-development) version tags:

```
npm list -g @zowe/cli
```

For the @lts-incremental version tag:

```
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.

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 **echoed**.

Home directory structure

| Name | Date Modified | Size |
|-----------------|----------------------|------|
| imperative | 18 Jun 2019 at 14:40 | |
| logs | Today at 14:38 | |
| imperative.log | Today at 13:46 | 9 |
| plugins | Today at 14:38 | |
| installed | Today at 14:38 | |
| lib | 3 May 2019 at 08:50 | |
| node_modules | 19 Jun 2019 at 13:32 | |
| plugins.json | Yesterday at 16:40 | 398 |
| profiles | 14 Jun 2019 at 07:10 | |
| settings | 3 Jun 2019 at 08:40 | |
| imperative.json | 10 Jun 2019 at 11:50 | 56 |
| zowe | 18 Jun 2019 at 14:02 | |
| logs | 11 Oct 2018 at 16:24 | |
| zowe.log | 24 Jun 2019 at 17:36 | 3 |

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 connection 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 is compatible with the currently supported Node.js LTS versions. For an up-to-date list of supported LTS versions, see [Node.js.org](#).

To gather the Node.js and npm versions installed on your computer, issue the following commands:

```
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 command 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.

EACCESS error when issing `npm install` command

Valid on Windows, Mac, or Linux

Symptom:

An EACCESS error is returned when you issue the `npm install -g` command to install a package from Zowe.org or npm.

Solution:

To resolve the issue, follow the steps described in [Resolving EACCESS permissions errors when installing packages globally](#) in the npm documentation.

***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 72

`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 173.
- 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 451 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.

Zowe Explorer**Troubleshooting Zowe Explorer**

As a Zowe Explorer user, you may encounter problems with how the VS Code extension functions. This article presents known Zowe Explorer issues and their solutions.

Before reaching out for support

1. Is there already a GitHub issue (open or closed) that covers the problem? Check [Zowe Explorer Issues](#).

2. Review the current list of [Known Zowe Explorer issues](#) on page 457 in documentation. Also, try searching using the Zowe Docs search bar.
3. Collect the following information to help diagnose the issue:
 - Zowe Explorer and VS Code version installed.
 - Node.js and NPM versions installed.
 - Whether you have Zowe CLI and the Secure Credential Store Zowe CLI plug-in installed.
 - Your operating system.
 - Zowe Logs.

Usually, can be found in C:\Users\userID\.zowe\zowe\logs.

Use [the Slack channel](#) to reach the Zowe Explorer community for assistance.

Known Zowe Explorer issues

The following topics contain information that can help you troubleshoot problems when you encounter unexpected behavior, using Zowe Explorer.

Data Set Creation Error

Symptom:

Data set creation fails.

Sample message:

Error running command zowe.createDataset: z/OSMF REST API Error: http(s) request error event called Error: self signed certificate in certificate chain. This is likely caused by the extension that contributes zowe.createDataset.

Solution:

Set the value of the Reject-Unauthorized parameter to `false`. Use the profile edit function to change profile's parameters.

Opening Binary Files Error

Symptom:

When opening a binary file, an error message pops up.

Sample message:

Cannot open file:///Users/userID/.vscode/extensions/zowe-vs.code-extension-for-zowe-1.8.0/resources/temp/binaryfilename. Detail: File seems to be binary and cannot be opened as text

Error running command zowe.editMember: cannot open file:///Users/userID/.vscode/extensions/zowe-vs.code-extension-for-zowe-1.8.0/resources/temp/binaryfilename. Detail: File seems to be binary and cannot be opened as text. This is likely caused by the extension that contributes zowe.editMember.

Solution:

Raising a Zowe Explorer issue on GitHub

You can raise GitHub issues against [the Zowe Explorer repository](#). It is suggested that you use either the bug or feature request.

Raising a bug report

Please provide as much of the information listed on [Troubleshooting Zowe Explorer](#) on page 456 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.

Submitting a feature request

Feature requests are just as important to the Zowe project as bug reports. Feature requests should contain clearly formulated ideas that can improve user experience.