MICRO FOCUS | SERENA

ChangeMan® ZMF

REST Services Getting Started Guide

Product version: 8.2 Patch 5

Publication date: March 2021

Table of Contents

Welcome to ChangeMan® ZMF

ChangeMan® ZMF is a comprehensive and fully integrated solution for Software Change Management systems in z/OS environments. It provides reliable and streamlined implementation of software changes from development into production. ChangeMan ZMF manages and automates the application life cycle, protects the integrity of the code migration process, and results in higher quality delivered code to any test environment and to the production environment.

Before You Begin

See the Readme for the latest updates and corrections for this manual.

Objective

ChangeMan ZMF REST Services Getting Started Guide reviews the basics of REST Services.

Audience

Use this document if you are responsible for any of these tasks:

- Managing software at your enterprise
- Managing test environments
- Developing and changing software components managed by ChangeMan ZMF

This guide assumes that you are familiar with ChangeMan ZMF and your security system.

Navigating this book

This manual is organized as follows:

Chapter 1 - Getting Started with REST Services

Change Bars

Change bars in the left margin are used when changes are introduced since the previous version of this publication.

Guide to ChangeMan ZMF Documentation

The following sections provide basic information about ChangeMan ZMF documentation.

ChangeMan ZMF Documentation Suite

The ChangeMan ZMF documentation set includes the following manuals in PDF format.

Manual	Description
Administrator's Guide	Describes ChangeMan ZMF features and functions with instructions for choosing options and configuring global and application administration parameters.
ChangeMan ZMF Quick Reference	Provides a summary of the commands you use to perform the major functions in the ChangeMan ZMF package life cycle.
REST Services Getting Started Guide	Getting Started Guide for ZMF REST Services (this manual).

Manual	Description
Customization Guide	Provides information about ChangeMan ZMF skeletons, exits, and utility programs that will help you to customize the base product to fit your needs.
Db2 Option Getting Started Guide	Describes how to install and use the Db2 Option of ChangeMan ZMF to manage changes to Db2 components.
ERO Concepts	Discusses the concepts of the ERO Option of ChangeMan ZMF for managing releases containing change packages.
ERO Getting Started Guide	Explains how to install and use the ERO Option of ChangeMan ZMF to manage releases containing change packages.
IMS Option Getting Started Guide	Provides instructions for implementing and using the IMS Option of ChangeMan ZMF to manage changes to IMS components.
INFO Option Getting Started Guide	Describes two methods by which ChangeMan ZMF can communicate with other applications: ■ Through a VSAM interface file. ■ Through the Tivoli Information Management for z/OS product from IBM.
Installation Guide	Provides step-by-step instructions for initial installation of ChangeMan ZMF. Assumes that no prior version is installed or that the installation will overlay the existing version.
Java / zFS Getting Started Guide	Provides information about using ZMF to manage application components stored in USS file systems, especially Java application components.
Load Balancing Option Getting Started Guide	Explains how to install and use the Load Balancing Option of ChangeMan ZMF to connect to a ZMF instance from another CPU or MVS image.
M+R Getting Started Guide	Explains how to install and use the M+R Option of ChangeMan ZMF to consolidate multiple versions of source code and other text components.
M+R Quick Reference	Provides a summary of M+R Option commands in a handy pamphlet format.
Messages	Explains messages issued by ChangeMan ZMF, SERNET, and System Software Manager (SSM) used for the Staging Versions feature of ZMF.
Migration Guide	Provides guidance for upgrading ChangeMan ZMF
OFM Getting Started Guide	Explains how to install and use the Online Forms Manager (OFM) option of ChangeMan ZMF.
SER10TY User's Guide	Gives instructions for applying licenses to enable ChangeMan ZMF and its selectable options.
User's Guide	Describes how to use ChangeMan ZMF features and functions to manage changes to application components.

Manual	Description
XML Services User's Guide	Documents the most commonly used features of the XML Services application programming interface to ChangeMan ZMF.
ZMF Web Services User's Guide	Documents the Web Services application programming interface to ChangeMan ZMF.

Using the Manuals

Use Adobe® Reader® to view ChangeMan ZMF PDF files. Download the Reader for free at get.adobe.com/reader/.

This section highlights some of the main Reader features. For more detailed information, see the Adobe Reader online help system.

The PDF manuals include the following features:

- **Bookmarks.** All of the manuals contain predefined bookmarks that make it easy for you to quickly jump to a specific topic. By default, the bookmarks appear to the left of each online manual.
- **Links.** Cross-reference links within a manual enable you to jump to other sections within the manual with a single mouse click. These links appear in blue.
- **Comments.** All PDF documentation files that Serena delivers with ChangeMan ZMF have enabled commenting with Adobe Reader. Adobe Reader version 7 and higher has commenting features that enable you to post comments to and modify the contents of PDF documents. You access these features through the Comments item on the menu bar of the Adobe Reader.
- **Printing.** While viewing a manual, you can print the current page, a range of pages, or the entire manual.
- **Advanced search.** Starting with version 6, Adobe Reader includes an advanced search feature that enables you to search across multiple PDF files in a specified directory.

Searching the ChangeMan ZMF Documentation Suite

There is no cross-book index for the ChangeMan ZMF documentation suite. You can use the Advanced Search facility in Adobe Acrobat Reader to search the entire ZMF book set for information that you want. The following steps require Adobe Reader 6 or higher.

1. Download the ZMF All Documents Bundle ZIP file and the *ZMF Readme* to your workstation from the My Downloads tab on the Serena Support website.
2. Unzip the PDF files in the ZMF All Documents Bundle into an empty folder. Add the *ZMF Readme* to the folder.
3. In Adobe Reader, select **Edit** | **Advanced Search** (or press **Shift+Ctrl+F**).
4. Select the **All PDF Documents in** option and use **Browse for Location** in the drop down menu to select the folder containing the ZMF documentation suite.
5. In the text box, enter the word or phrase that you want to find.

6 Optionally, select one or more of the additional search options, such as **Whole words only** and **Case-Sensitive**.

7 Click **Search**.

8 In the **Results**, expand a listed document to see all occurrences of the search argument in that PDF.

9 Click on any listed occurrence to open the PDF document to the found word or phrase.

Typographical Conventions

The following typographical conventions are used in the online manuals and online help. These typographical conventions are used to assist you when using the documentation; they are not meant to contradict or change any standard use of typographical conventions in the various product components or the host operating system.

Convention	Explanation
italics	Introduces new terms that you may not be familiar with and occasionally indicates emphasis.
bold	Emphasizes important information and field names.
UPPERCASE	Indicates keys or key combinations that you can use. For example, press the ENTER key.
`monospace`	Indicates syntax examples, values that you specify, or results that you receive.
`monospaced italics`	Indicates names that are placeholders for values you specify; for example, *`filename`*.
vertical rule |	Separates menus and their associated commands. For example, select File | Copy means to select Copy from the File menu. Also, indicates mutually exclusive choices in a command syntax line.

Chapter 1
Installation and Configuration

This chapter presents an overview of steps to install and configure.

Pre-requisites

ChangeMan ZMF 8.2 Patch 4 is a requirement.

Sufficient memory - the Tomcat JVM startup can fail due to a restriction on memory imposed by local exits (e.g. IEFUSI or IEALIMIT etc.). The actual amount of storage needed during initialization varies but is in the region of between 550 and 700 Mb depending on several factors including which version of z/OS you are running. If you do not allow the address space to get the storage it needs then it will fail during initialization, possibly with symptoms including rc=100 and java.lang.OutOfMemoryError: Failed to create a thread.

Note also the external requirements listed at "External 3rd Party Dependencies" on page 49"

Instructional Videos

You can get further information from the following instructional videos:

- **Introduction to ZMF REST Services**

  YouTube Direct Link: https://youtu.be/4q1b5Ya1Mzs

  YouTube Embed Link: <iframe width="961" height="541" src="https://www.youtube.com/embed/4q1b5Ya1Mzs" frameborder="0" allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>

- **How to Subscribe to a webhook using ZMF REST Services**

  YouTube Direct Link: https://youtu.be/dU9v7EEKvWQ

  YouTube Embed Link: <iframe width="961" height="541" src="https://www.youtube.com/embed/dU9v7EEKvWQ" frameborder="0" allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>

- **An Overview of a Sample Jenkins Process**

  YouTube Direct Link: https://youtu.be/4lEKJimFk9Y

  YouTube Embed Link: <iframe width="961" height="541" src="https://www.youtube.com/embed/4lEKJimFk9Y" frameborder="0" allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>

- **ZMF REST Services Overview**

  YouTube Direct Link: https://youtu.be/gyGk3PCJd-A

  YouTube Embed Link: <iframe width="961" height="541" src="https://www.youtube.com/embed/gyGk3PCJd-A" frameborder="0" allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>

- **How to Store ZMF Credentials for Jenkins**

  YouTube Direct Link: https://youtu.be/ZA5CtfR_IfY

  YouTube Embed Link: <iframe width="961" height="541" src="https://www.youtube.com/embed/ZA5CtfR_IfY" frameborder="0" allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>

- **How to Install ZMF REST Server Support**

  YouTube Direct Link: https://youtu.be/mFSU0DYkAdE

  YouTube Embed Link: <iframe width="961" height="541" src="https://www.youtube.com/embed/mFSU0DYkAdE" frameborder="0" allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>

- **How to Activate and Use ZMF REST Server**

  YouTube Direct Link: https://youtu.be/XjNh1SojFcg

  YouTube Embed Link: <iframe width="961" height="541" src="https://www.youtube.com/embed/XjNh1SojFcg" frameborder="0" allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>

Security set up

You will be creating a Tomcat started task as part of this installation. This started task needs to be assigned a userid with a valid OMVS segment. To allow the file permissions on the Tomcat install directories/files to work best you should ensure that the started task userid is connected to a group that has an OMVS gid associated with it. The userid submitting the install job must also be connected to this group.

Note that neither the started task userid nor the group to which it is connected needs any kind of ZMF authority. The stc userid/group need only have full access to the directories and libraries which are part of the Tomcat installation process.

Run INSTALL job

Change the job card in the sample JCL INSTALL member to suit your site. Make sure you specify the GROUP parameter to be the common group to which the started task userid and the installer userid are connected.

Set the JCL symbolic parameters to reflect your choice of location for the Tomcat install, e.g.

```
// SET INSTJCL=<dsnHLQ>.ZMF822TC.CNTL
// SET TCHOME='/usr/tomcat'
// SET SUBDAT='/usr/data'
// SET TDIR='/tmp'
```

&INSTJCL is the sample install JCL library. This library also contains the ZMFPARMS member which will be referenced by the Tomcat started task.
&TCHOME is where the Tomcat executables, and subsequently deployed application archives, will be stored.
The &SUBDAT directory should be specified to be outside of the Tomcat install directory structure (&TCHOME). It will eventually contain your event subscribers data file.

Then run the Install job which should complete with all step return codes=0.

The /usr/tomcat directory should then be populated.

Locate JZOS Batch Loader (JVMLDM86)

Locating JVMLDM86 - JVMLDM86 is an executable module is supplied by IBM as part of their Java implementation and may well have already been copied to a PDSE by your site. If it hasn't then you can do this with the following USS process:

```
cd /usr/lpp/java/J8.0_64/mvstools
cp -X JVMLDM86 "//'CMNTP.JZOS.LOADLIB(JVMLDM86)'"
```

Create the Started task proc

Update the TCPROC member to set the CNFGLIB and JZOSLIB variables to your libraries. The JZOSLIB is where you have the JVMLDM86 load module, e.g. CMNTP.JZOS.LINKLIB, and then place this member in a system proclib to be run as a started task.

```
//TCPROC PROC CNFGLIB=CMNTP.TOMCAT.ZMF822TC.C7.CNTL,  config XML & env script
//   TCENV=TCENV,                        < Member of CNFLIB with STDENV script
//   JZOSLIB=CMNTP.JZOS.LOADLIB,         < JZOS launcher PDSE LIB
//   VERSION='86',                       < JZOSVM version: 70,76,80,86
```

Update Environment settings

Then update the TCENV member with appropriate values, for example:

```
export JAVA_HOME=/usr/lpp/java/J8.0_64
CATALINA_HOME=/usr/tomcat
CATALINA_BASE=/usr/tomcat
IJO="-Xms64m -Xmx128m"  # min and max Java heap sizes
```

Choose the ports on which you want tomcat to listen, and update the SERVVARS member, with those values, for example:

```
<!ENTITY httpPort "8085">              <!-- the Tomcat HTTP port         -->
<!ENTITY httpPorts "9992">             <!-- the Tomcat HTTP SSL port    -->
```

Update the ZMFPARMS member - parameters are described within the member including the default values if any.

The Subscribers.dat file will be created by the zmfrest application if it is not there (i.e. on first start up). This is where the event subscriber information is held and there should be a unique location for each REST server application that is deployed (see later for information on deploying multiple applications).

Start Tomcat

When this has been done you can start the TomCat started task to verify success. Look out for security errors which will occur if the permissions are not right:

```
ICH408I USER(SERT    ) GROUP(CMNTP   ) NAME(CHANGEMAN TECH PUBS ) 457
  /u/sert/Q001/TomCat/logs/localhost_access_log.2019-09-17.txt
  CL(DIRACC  ) FID(01E2D9C8C6E2F5000F04000688610000)
  INSUFFICIENT AUTHORITY TO OPEN
  ACCESS INTENT(-W-)  ACCESS ALLOWED(OTHER      R-X)
  EFFECTIVE UID(0000000586)  EFFECTIVE GID(0000000024)
```

Issue the STOP command to shutdown Tomcat

Deploy .war files

Edit the DEPLOY member for the zmfrest war file which copies the .war file to the webapps folder.

As soon as TomCat detects the presence of the .war files it starts activating the relevant servlets.

Enable in ZMF Admin

Once TomCat is running and the servlets are ready, Then the next step is to enable the interface in the ZMF Global Administration Options facility (=A.G) panel CMNGAMN1.

```
CMNGAMN1             Update Global Administration Options
Option ===> ____________________________________________________

1  Parms          Global parameters
2  Library        Library types
3  Language       Language names
4  Procedures     Compiling procedures
5  Reason Codes   Reason codes for unplanned packages
6  Sites          Site information
7  Lock           Application parameter locks
8  HLL Exits      High level language exits
9  Field Names    User field name substitution
C  Component      Component information
D  Dates          Installation calendar
E  REST           REST api server
H  Housekeeping   Housekeeping tasks
I  Impact         Impact Analysis
N  Notify         Global notification file
O  Options        Selectable options
R  Reports        ChangeMan ZMF batch reports
S  Skeletons      Skeleton procedures
```

Select option E - REST api server and then you will see panel CMNGRS01:

```
CMNGRS01                 REST api server
Command ===> ___________________________________________

     Server procedure . . SERDTCI
              address . . d001.microfocus.com ____________________________
                 port . . 08085
              context . . zmfrest/list ___________________________________

   http send time-out . . 0000002
   http recv time-out . . 0000002

 Enter / to select option
   _  Issue start command for procedure?
   _  Server active?
   /  Poll for server using http?

   /  Apply saved admin settings
```

This allows you to maintain the values used by the ZMF REST api servers.

To save all changes and leave this panel use PF3/end. To discard all changes and leave use the CANCEL primary command. ENTER redisplays the panel with the same values, nothing is saved.

Server procedure - The name of the cataloged procedure which is used by ZMF to start the Tomcat started task which will host the REST api server.

address - The DNS name or IP address of the server.

port - The port number on which the server is listening.

context - The context used by the REST api servlet running in the server address space. The default is zmfrest.
http send time-out - The send time out value (in seconds) applied to connections from internal ZMF functions to the REST api server. The range is 1 - 2678400 and the default is 2 seconds.
http recv time-out - The receive time out value (in seconds) applied to connections from internal ZMF functions to the REST api server.
The range is 1 - 2678400 and the default is 2 seconds.
Issue start command for procedure? - Use '/' to have ZMF issue the start command for the Tomcat procedure. If the REST api server is required to be active then ZMF will issue the start command during initialization if it cannot detect the presence of the relevant servlet. The same is true should these admin definitions be applied 'in-flight'. Only select this option if you wish to run your Tomcat started task on the same LPAR as this ZMF instance.
Server active? - Use '/' to activate REST server support within this ZMF instance. Turning this on will prompt ZMF to fill in relevant ISPF skeleton and HLLX REXX variables supporting the event emission processes. It will also cause ZMF to actively look for the presence of the relevant Tomcat hosted REST servlet.
Poll for server using http? - Use '/' to require ZMF to issue http requests to detect the presence of the REST server. The default detection process is via a sysplex wide enqueue mechanism which is more efficient than using http. However, if your Tomcat started task is not running on the local sysplex then the http mechanism must be used.
Apply saved admin settings - Once any updates have been made the dialog will use the REFRESH service to take the actions required to apply the settings, e.g. start the server, as required.

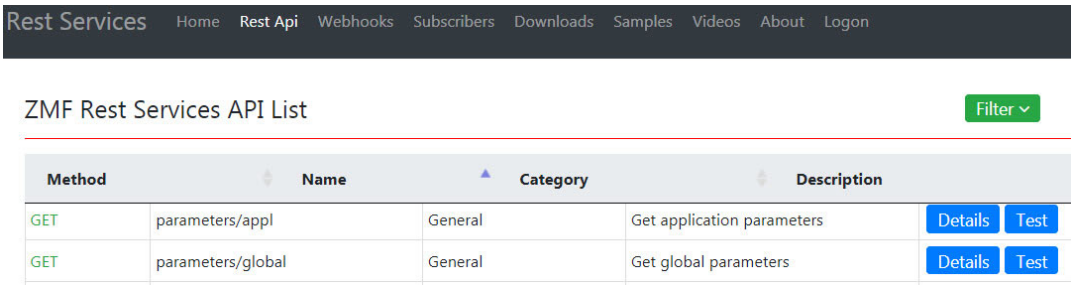
This completes the TomCat install and initial configuration.

Simple Installation Verification Procedure:

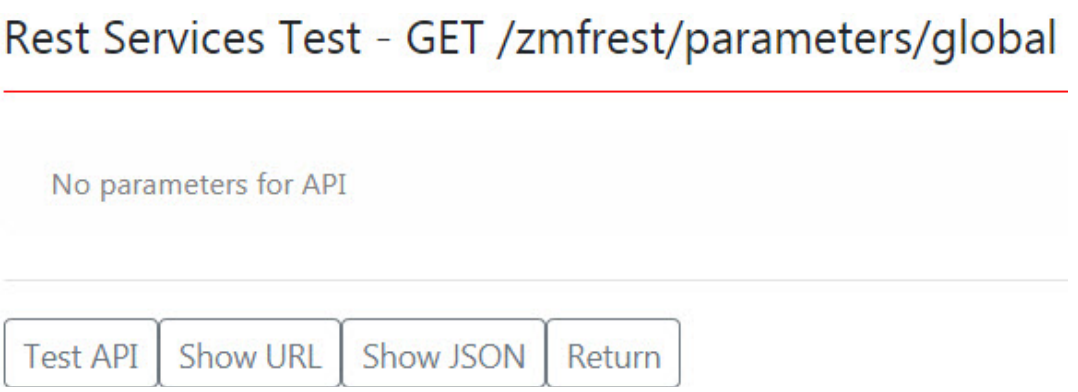
With your ZMF instance up and running and the tomcat web apps configured correctly in ZMF Global Admin, you should be able to contact your target ZMF with a REST call. Use the /zmfrest/list url to get a list of ZMF REST api's up, an example:
http://d001.microfocus.com:8085/zmfrest/list

You can logon to your target ZMF using your TSO userid and password and then try driving one of the apis via the 'prototyping' facility, for example:

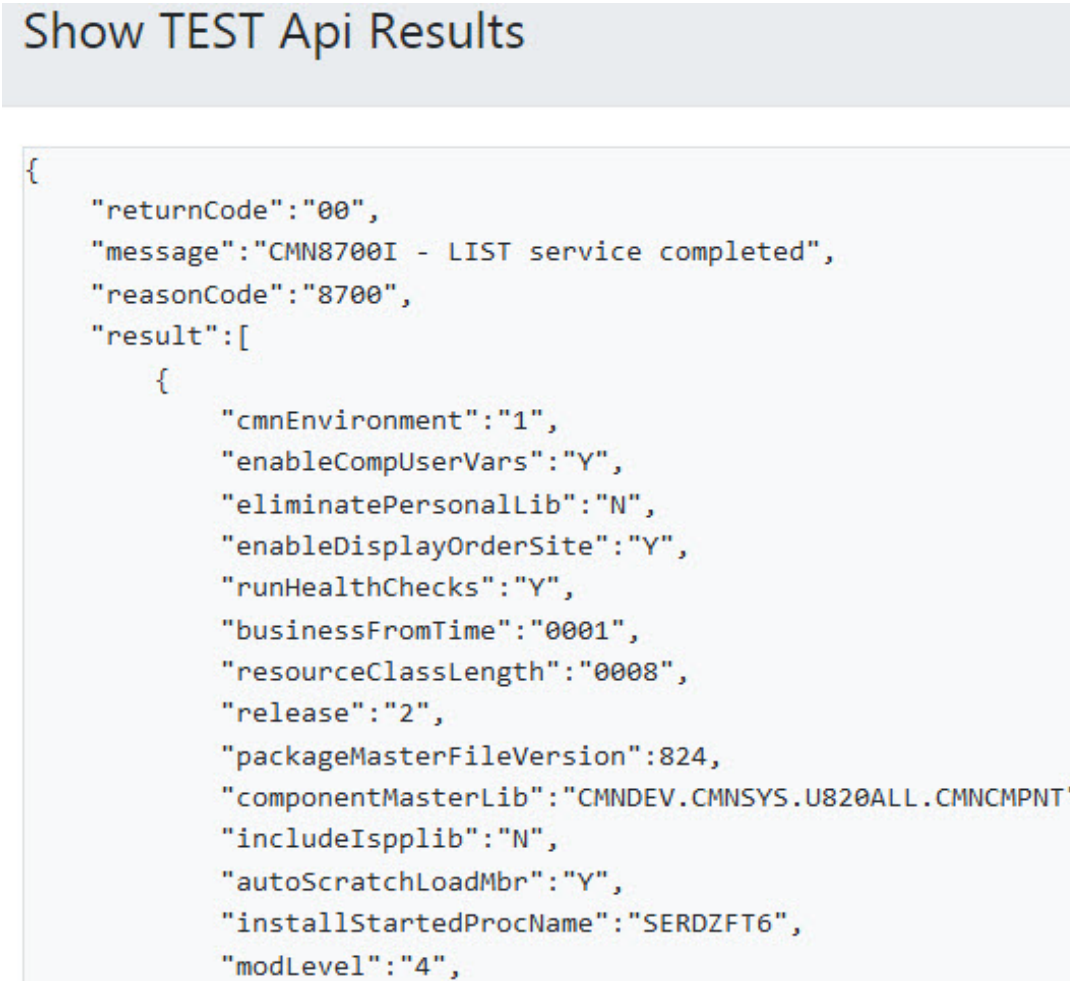
Scroll down to find the 'Parms' category, then open up the two apis by clicking the row.



Get the global parameters for your target ZMF subsystem. This API requires no parameters (there is only one set of global parms) so clicking the 'Test' button on the right brings up the next panel where you would normally specify parameters (there are none for this call), then click on the ’Test API’ button on that panel.



You should receive a list of global parameters for your target ZMF subsystem in JSON format looking similar to this:



Running multiple instances of ZMF

The Tomcat started task will support multiple instances of ZMF simultaneously. You need a different 'context' for each ZMF instance you wish to support. In order to do this you must copy the zmfrest.war file to different names within the Tomcat webapps directory, e.g. zmfrestj.war was created to support another instance (subsys=J) separately from an existing

instance. Then you add context specific qualifiers to the parameters in ZMFPARMS - an example

```
#
# General Parameters
#
ZMFHOST=D001.MICROFOCUS.COM
ZMFRESTHOST=D001.MICROFOCUS.COM
#
# U820ALL Parameters
#
ZMFREST.ZMFSUBSYS=I
ZMFREST.ZMFNAME=U820ALL
ZMFREST.ZMFPORT=6611
ZMFREST.ZMFEVENTFILE=/u/cmndev/tomcati/Subscribers.dat
#
# U820DP Parameters
#
ZMFRESTJ.ZMFSUBSYS=J
ZMFRESTJ.ZMFNAME=U820DP
ZMFRESTJ.ZMFPORT=6621
ZMFRESTJ.ZMFEVENTFILE=/u/cmndev/tomcatj/Subscribers.dat
```

You will need to add the relevant 'log' DD name to the tomcat procedure, e.g.

//ZMFREST DD SYSOUT=*

This needs to be added for the context zmfrest, via the A.G.E panel accordingly.

Chapter 2

Using ZMF REST Services

This chapter presents an overview of REST Services concepts.

Implementing http event notifications and REST api's into ZMF

Overview

The purpose of this product enhancement is essentially two fold:

1. To expose ChangeMan ZMF events to external authorized subscribers in the form of webhooks, and to act upon responses returned from those subscribers.

- There may be from zero to any number of webhook subscribers of a given ZMF event notification.
- The webhook subscription on which support is focused in this first release is serviced by Jenkins.
- 'Endpoint' support such as SonarQube, Jira, Octane etc is provided via existing Jenkins Plugins etc.
- While Jenkins is the focus, the support is generic and other products/processes may subscribe to these webhooks. The webhook is driven using standard http messages and it is for the webhook target to handle the message.

2. To take incoming 'unsolicited' requests from authorized users to perform a function within ZMF. These will be in the form of an incoming REST api call.

Exposing ChangeMan ZMF function through REST Services

Many existing ZMF users are exploring Agile development and automated tool chains as part of a continual improvement process for accelerating Mainframe Application development. The role of ZMF in these processes is changing with the requirement being that ZMF now participate rather than drive the development process as sites move to continuous integration and delivery pipelines.

A number of large ZMF sites are building workflows with a variety of proprietary and open source technologies and want to be able to integrate their ZMF implementation into these new pipelines.

The requirement is support the user needs to be able to integrate ZMF as part of an automated workflow by exposing ZMF functions as REST apis which will enable these functions to be initiated from a participating subscriber in an automated workflow.

As a generic requirement it is key to make sure the api meets the standards and requirements that are expected in this space. This means ensuring the following:

1 Supports a REST-based interface

2 Supports a HTTP callback (WebHook) mechanism for event notification. This will provide the push mechanism from ZMF to whatever distributed orchestration started the workflow, such as Jenkins. This is a specific implementation of the bidirectional support.

3 Provides a secure interface. Credentials are not visible in scripts and are transmitted securely. Others use a Jenkins plug-in to generate a pass token that is passed in the REST api.

Development Workflow

1 A code change has been identified as a task in an agile planning tool. This drives the creation of a simple package in ZMF

2 Source members can then be checked out from the base line into a specific package. This could be driven from an IDE task initiated by a developer or as part of an automated process.

3 For any given package there is a need to

- **a** Get a member list of the package with filters on source type as an option.
- **b** Get the actual source member or members from the package
- **c** Get build meta-data for a particular member or members

4 When any components from a package are committed back to ZMF (Check in) an event is triggered which could be used by any tool which is part of a distributed orchestration

5 For coding standards or security rules analysis when a component has been checked into ZMF then a workflow requests a component and all dependencies from a package. Source code would be delivered as part of the request and this could then be passed by an orchestration engine to a separate process for analysis.

6 As part of a workflow or process initiated by a developer a component build can be requested. This will build the component on the mainframe based on the build meta-data for the component. When the build has completed this triggers an event

- **a** The build status from a component build needs to available so that any errors in the build step can be returned to an IDE or to a distributed orchestration tool.
- **b** This build process could be iterative based on the number of components in a package

7 On successful build the contents of a package can then be promoted which could then trigger automated testing

8 When a code change as part of a package has been tested then this can be associated to an existing ERO release

Design Overview

The ZMF/Jenkins integration implementation consists of a number of loosely coupled services. This section gives a brief overview of the main components:

Events
Describe something that has occurred in ZMF. For example, Package Create.

Event Variables
Standardized variable names used throughout the various services.

Event Source
The source of a ZMF Event.

Event Clients
Program to forward a specific event from an Event Source to REST Services

Event Subscribers
An entity represented by a URI that subscribes to one or more ZMF Events.

REST Services
REST services provides a REST API for Change Man ZMF Services. This is a true REST API where each transaction connects to ZMF, processes the request, disconnects from ZMF and returns the result to the client. Also processes events sent from the event clients. Handles sending events to multiple subscribers and custom response handling.

ZMF REST Services Extensions
A client side wrapper for ZMF REST API's. This implements support for bulk client transactions and helper methods that demystify ZMF data structures.

Jenkins SCM Plugin interface
A Jenkins Plugin to provide ZMF SCM functions in pipeline scripts.

Security Considerations
Security Considerations for ZMF / Jenkins Environment.

Events
The ZMF Package/Component lifecycle may be described by a series of Events. This section identifies the published events as well as the event source.

The following sources generate events depending on the type of processing:

SKEL
ISPF Skeletons customize batch processing in ZMF. Batch processes generate Events by imbedding the appropriate Event Skeleton (An Event Client).

HLLX
Exit routines in ZMF that provide consistent processing for multiple client types. These routines generate Events by calling the Event Client.

LOG
ZMF writes Log records at many points during processing. A post-processing routine sends the records to REST Services.

The table below lists the currently defined ZMF Events (Taken from the existing ZMF Event Log) A review of ZMF Services is required to determine additional event points.

Events with an Asterisk are not processed. The SKEL/HLLX/LOG columns indicate if this source is capable of generating the event. (values are Yes/No/Maybe).

Event	**Event Source**			**Description**
	SKEL	HLLX	LOG	
00*	N	N	Y	major - initialize/terminate
01	Y	Y	Y	Backout Package
02	Y	N	Y	Install Package
03	Y	N	Y	Temporary Change Cycle
04	Y	N	Y	Distribute Package
05*	N	N	Y	Unauthorized Member Access
07*	N	N	Y	Generate Package Information
08	N	N	Y	Delete Package (Physical Delete)
09*	N	M	Y	Update Application Information
10	note1	Y	Y	Revert Package
11*	N	N	Y	Update Global Information
12	Y	N	Y	Activate Component
13	N	N	Y	Memo Delete Package
14	N	N	Y	Undelete Package
15	Y	N	Y	Baseline Ripple
16	Y	N	Y	Reverse Baseline Ripple
18*	N	N	Y	Age Installed Package
20	N	Y	Y	Approve Package
21*	N	N	Y	Re-sync Calendar
22*	N	N	Y	Age Staging Libraries
23	note4	N	Y	Backout Release
24	note4	N	Y	Install Release
25	note4	N	Y	Distribute Release
26	N	N	Y	Delete Release
27	note4	N	Y	Revert Release
28	N	N	Y	Approve Release
29	N	N	Y	Reject Release
30	N	Y	Y	Reject Package
31	N	N	Y	Memo Delete Release
32	N	N	Y	Undelete Release
33	note4	N	Y	Baseline Release
34	N	N	Y	Install Release Aged
35	N	N	Y	Block Release
36	N	N	Y	Unblock Release
37	N	N	Y	Create/Update Release
38*	N	N	Y	HLLX Administration
39*	N	N	Y	HLLX Commands
40	N	Y	Y	Freeze Package
42	N	Y	Y	Selectively Unfreeze Package
43	note2	note2	Y	Demote Component
44	Y	Y	Y	Demote Package
45	Y	N	Y	Promote Release Area
46	Y	N	Y	Demote Release Area
48	Y	Y	Y	Promote Package
49	note2	note 2	Y	Promote Component
50	Y	Y	Y	Audit Package
51	N	N	Y	Alter Audit Return Code

Event	Event Source			Description
52	Y	N	Y	Audit Release Area
53	N	N	Y	Approve Release Area
54	N	N	Y	Reject Release Area
55	N	N	Y	Block Release Area
56	N	N	Y	Unblock Release Area
57	Y	N	Y	Submit package audit Auto resolve
58	Y	N	Y	Submit Release Audit Auto Resolve
60	N	N	Y	Lock package before promote
62	N	N	Y	Unlock Package after promote
64	N	Y	Y	scratch a component
66	N	Y	Y	rename a component
67	Y	Y	Y	Relink component
68	N	N	Y	copied a component
69	N	N	Y	Recompile Component
70*	N	N	Y	file tailoring started
71*	N	N	Y	file tailoring failed
72*	N	N	Y	file tailoring completed
78	note3	Y	Y	Checkin to release area complete
79	N	N	Y	Retrieve from release area complete
80	N	Y	Y	Create Package
81	note2	note2	Y	Check component into release area
82	note3	Y	Y	Checkout Component
83*	N	N	Y	potential checkout conflict
84	N	Y	Y	Stage Component
85*	N	N	Y	overlay previous module
86	N	Y	Y	delete component from package
87	note3	Y	Y	Checkout Component from release
88	N	Y	Y	copy forward package
89	N	N	Y	Retrieve component from release
90*	N	N	Y	monitor limbo and internal scheduler
91	N	N	Y	Update Release Global Approvers
92	N	N	Y	Update Release definitions
93	N	N	Y	Update Release Applications
94	N	N	Y	attach package to release
95	N	N	Y	detach package to release
96*	N	N	Y	link a release (RLM function?)
97*	N	N	Y	unlink a release (RLM function)
100	Y	Y	N	Pre-Build
101	Y	Y	N	Build

note1: Depending on whether you are working with an ALL or DP/P sites there may not be a batch job associated with a package revert. The skels notification of this event takes place where a batch job is used.

note2: These events may apply to many thousands of components in a single action. It makes no sense to call the REST server for each component. The package (or area) level action event is supported and any process driven by the package level event can use ZMF REST services to query the package for individual components should that be necessary.

Note, also, that the individual component events will continue to be written to the log for post-processing.

note3: A batch job is only involved, and this event emitted via the skels notification, if the action is performed in batch mode.

note4: An ERO release ties together one or more ZMF packages. When you do anything at the back end of a release lifecycle (e.g. install a release etc.) you are actually causing the same action to be taken for the group of packages that make up the release. So each of these packages will already be generating standard (i.e. base ZMF) events for these actions (e.g. 02 - Install package). For these events, you should use the 'log' emitted event.

Event Variables

There are many 'Variables' in the ZMF/Jenkins integration. Standard variable names provide consistency for multiple services across multiple platforms. Jenkins jobs often use 'build parameters' to customize processing, supplied in name/value pairs. Incorrect parameter names passed to a Jenkins process will cause a failure. These processes must use the standard names.

- Parameters for Event Clients
- Parameters passed from Event Clients to REST Services (QUERY/JSON)
- Parameters passed from REST Services to JENKINS
- Parameters passed to ZMF REST Services (QUERY/JSON)

ZMF Specific VARIABLE NAMES:

- APPL
- PACKAGE
- LIBTYPE
- COMPONENT
- RELEASE
- RELEASEAREA
- PROMOTIONNAME
- SITE
- PROMOTIONLEVEL
- JOBNAME
- JOBNUMBER
- USERID
- RESTSERVER (REST Services URL to identify ZMF)
- EVENTSOURCE (skel, hllx, log)

ZMF Non-specific Variables

- EVENT
- WAIT
- RETURNCONTENT

Note: This forces user scripts to use the ZMF defined variable names.

Event Source

- Events can be generated from a number of sources. The Event Source may be:
- A HLLX routine
  A step in ZMF batch job (ISPF SKEL)
  LOG Task.

Event Clients

Event Clients forward specific Events to REST Services for processing. REST Services provides a centralized service for processing multiple subscribers and holds key information including subscriber security. The following variables should be included in all event requests to REST Services

- EVENT=XX
- WAIT=Y/N
- RETURNCONTENT=Y/N
- + All applicable ZMF Variables. See Event Variables for a list of variables

REST Services

REST Services is a centralized service for processing Events. It manages:

- Subscribers through a Web Application
- Receiving Event requests from Event Clients
- Sending Events to one of more subscribers
- Interpreting response from each subscriber
- Sending response to each Event Client

REST Services will return data in JSON format. The response data is held in a tag named EVENT_RESULT. EVENT_RESULT holds an array of JSON elements for each subscriber:

- TARGET_EVENT The subscriber Event
- TARGET_ID The Target ID
- TARGET_URL The Target URL
- TARGET_HTTP_CODE The HTTP Status code from the Target
- TARGET_HTTP_MESSAGE The HTTP Status message from the Target
- TARGET_JENKINS_JOB_NUMBER The Jenkins Job Number
- TARGET_RESULT_URL URL to display Jenkins JOB Console output
- TARGET_JOB_STATUS Jenkins Job Status
- TARGET_JOBCONTENT Content of (TARGET_RESULT_URL)
- TARGET_SONAR_QUBE_URL The URL to display Sonar Qube result

REST Services will be responsible for managing Event Subscribers.

Event Subscribers

A ZMF Subscriber represents an entity interested in a ZMF Event. The subscriber is identified by the following attributes:

General Attributes:

- Subscriber Name A user defined friendly name for the subscriber (64 Characters)
- Event The ZMF Event number (A Valid ZMF Event Number) - selected from a drop down list
- URL The Subscriber URL. (256 Characters)
- HTTP Method POST/GET. The HTTP Method for this subscriber
- Return Content - true/false. Flag to denote if content is expected back from the event notification to the subscriber
- Enabled true/false. Flag to enable/disable specific subscriber.
- HLLX source - true/false. Set if an HLLX action can trigger this event
- Skel Source - true/false. Set if the event can be triggered from an ISPF Skeleton
- Log Source - true/false. Set if the event can be triggered as a result of logged event

Setting more than one of the above 3 sources to true will result in multiple triggers for the same event. The source of the event will be supplied on the list of parameters sent to the subscriber.

- Parameters QUERY/JSON Deliver parameters through QUERY String or JSON Body

Security Attributes:

- Authorization - NONE/BASIC
- Userid The userid to be used for BASIC authentication (64 Characters)
- Password The password to use for BASIC authentication (64 Characters)

ZMF Filtering Attributes Filter

- Appl - filter definition that limits processing to this Application. (4 Characters)
- Lib Type - filter definition that limits processing to this Library Type. (3 Characters)

Filtering allows the subscriber to refine the events received. For example, they may only want to receive events for application "DEMO".

Jenkins Attributes:

- Jenkins true/false. Flag indicating a Jenkins target.
- Project Identifies the Jenkins JOB to run.
- CLI use Jenkins CLI interface
- Wait (true/false) Flag to signal that the process should wait for the Jenkins job to complete.

- Timeout Timeout in Milliseconds (if wait = true)

Miscellaneous Attributes:

- Parser flag or string indicating how to parse results. This allows for specific plugins to process subscriber results within REST Services.

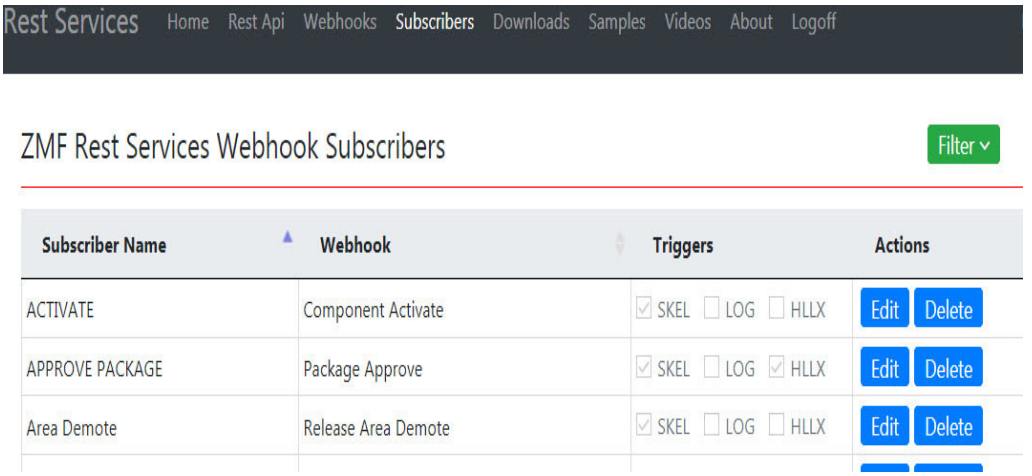
ZMF Variables: (List of Requested ZMF Variables)

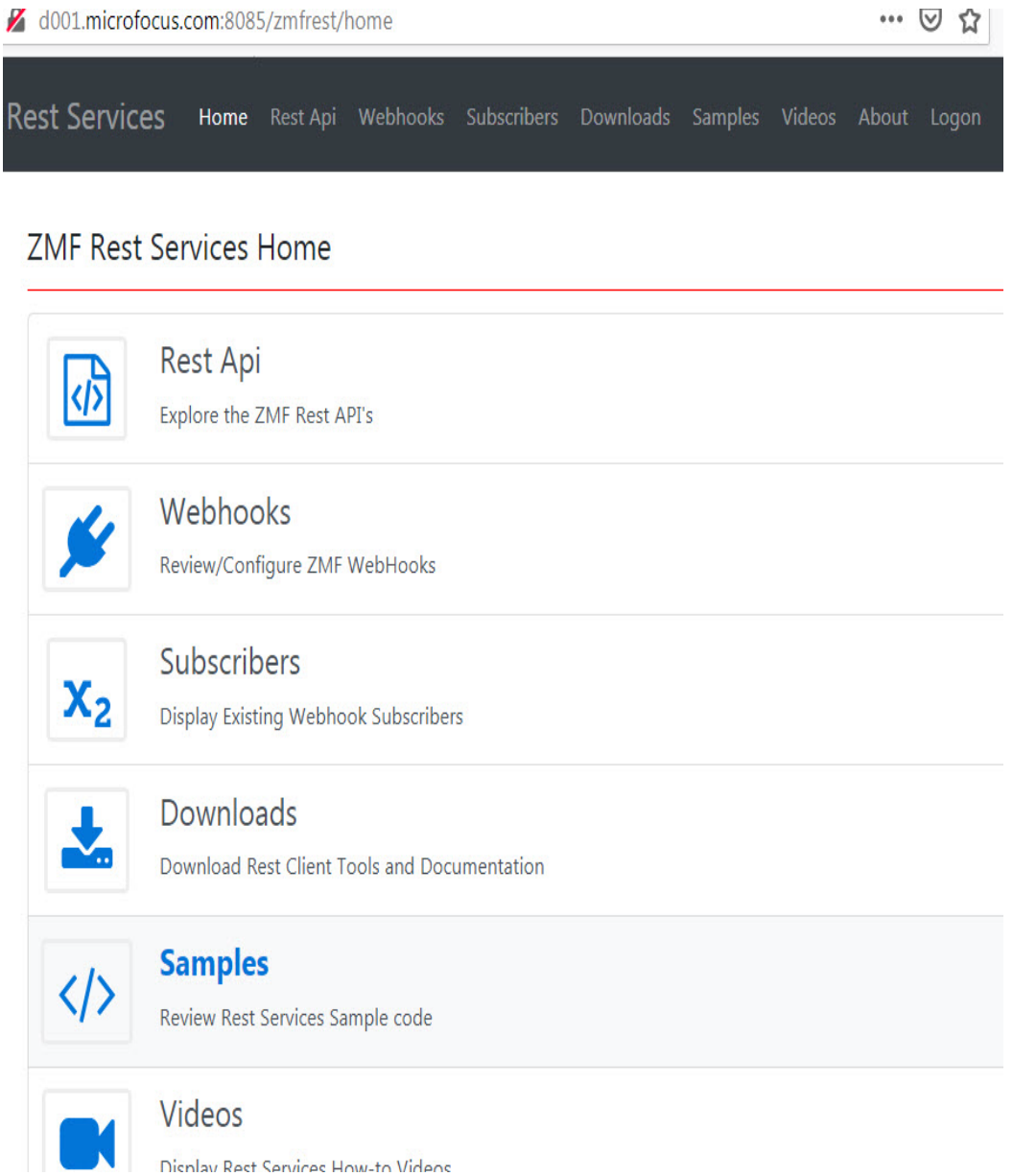
- Each ZMF Variable represented by a YES/NO selection (See Section on ZMF Variables)

Subscriber Flow Definition - Sample Screen Prints

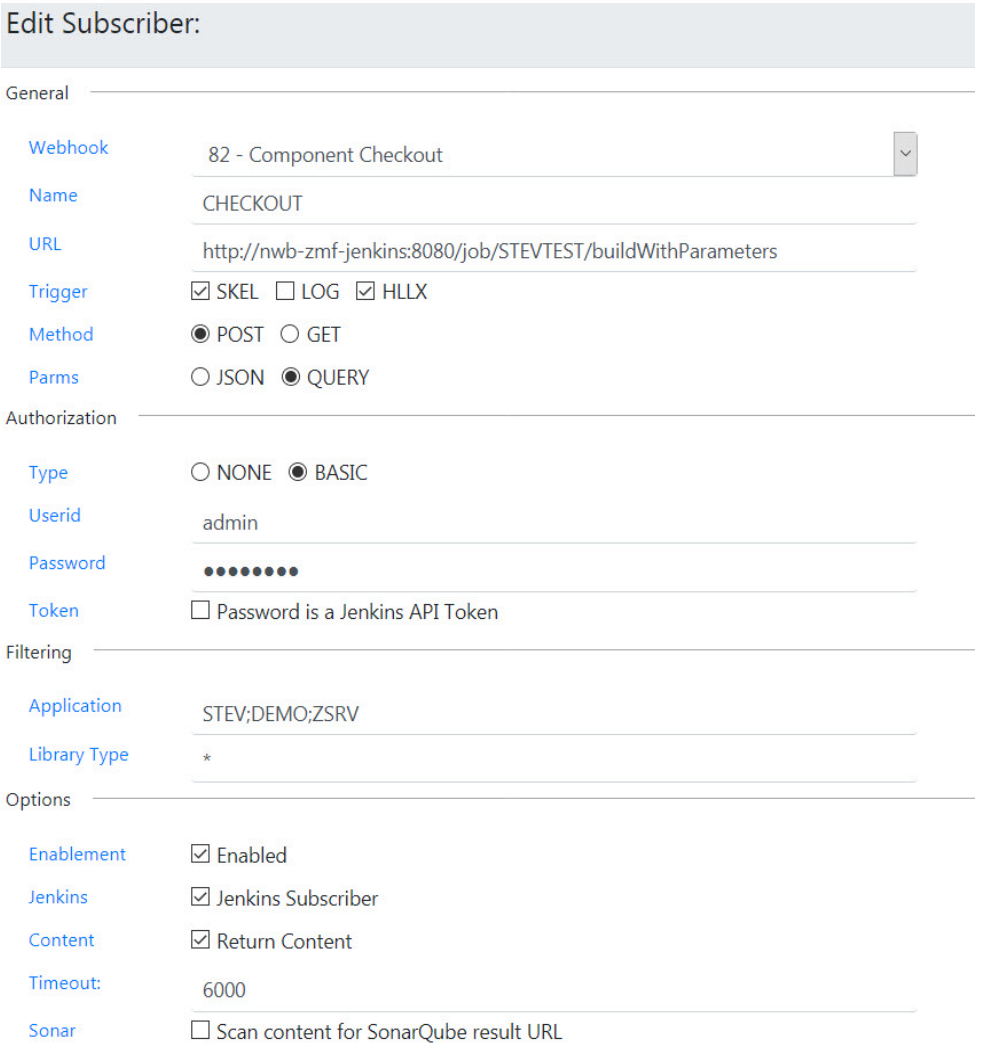
Sample screen prints of a Subscriber flow definition are shown below:

Partial List webhooks subscribers panel example

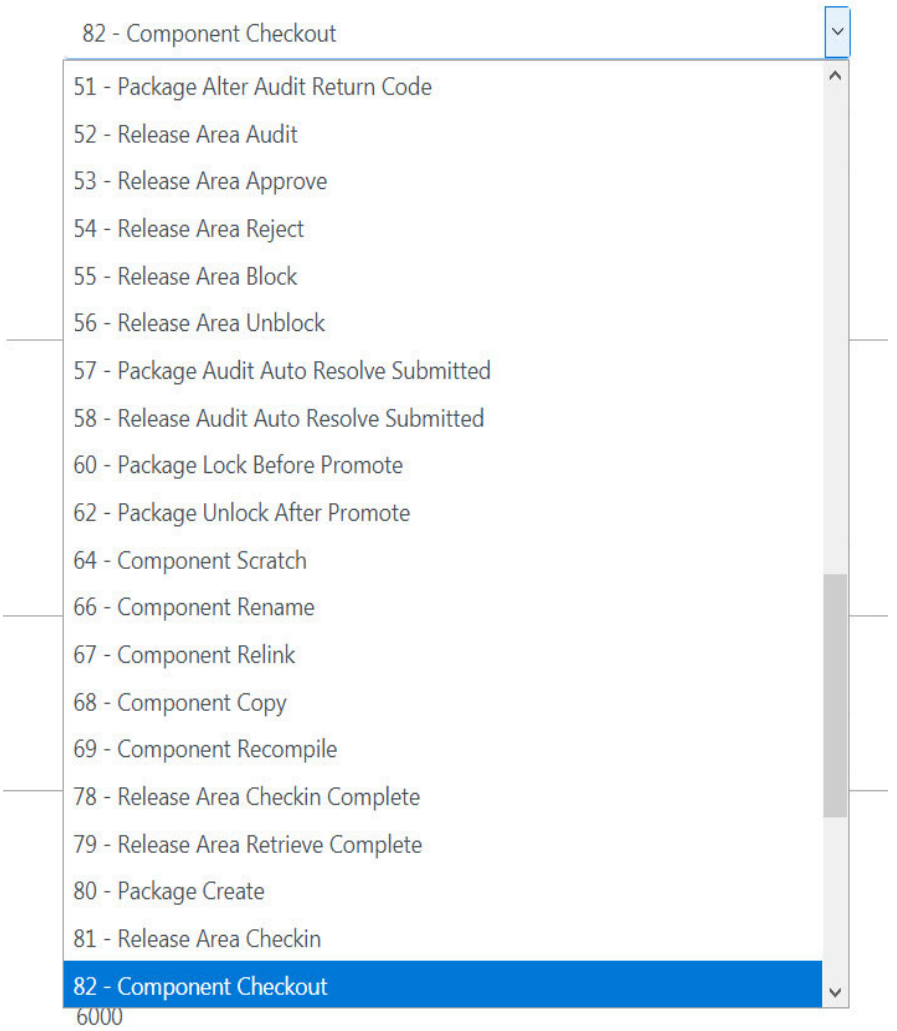



d001.microfocus.com:8085/zmfrest/home
Rest Services
Home
Rest Api
Webhooks
Subscribers
Downloads
Samples
Videos
About
Logon
ZMF Rest Services Home
Rest Api
Explore the ZMF Rest API's
Webhooks
Review/Configure ZMF WebHooks
Subscribers
Display Existing Webhook Subscribers
Downloads
Download Rest Client Tools and Documentation
Samples
Review Rest Services Sample code
Videos

Example of a Component Checkout Subscription



Example showing a partial drop down list of Events that might be subscribed to



Using application filtering with the REST server

Subscriptions to the REST server may be filtered on application and library type such that event notifications are only acted on if the application (or libtype) passed on the notification matches the filter. By default all applications (and libtypes) will pass filtering. It may be that you only wish to implement REST server notification for certain specific applications, this filtering can be implemented at the REST server by coding a list of applications (separated by a semi-colon) in the relevant filter field. However, this still results in unnecessary network traffic (and delay to the application which is not taking part in event notification) as the filtering is not done until the notification reaches the REST server. ZMF has been set up to avoid this unnecessary traffic/delay by separate mechanisms for HLLX and skeleton processing. For skeleton processing file tailoring only takes place for a specific package and, as such, the application is fixed for that particular file tailoring exercise. The file tailoring programs will pass the application when they query the REST server to see if an event has any subscribers. If the application does not pass

filtering at the REST server then the file tailoring program will mark that event as 'inactive' for this process and the relevant event notification steps will not be generated in the JCL created. For HLLX the decision on whether an event is 'active' or not is taken by the main program driving the exit calls. An HLL exit call could be for any application so the same test (as for skels processing) cannot be made. To avoid unnecessary traffic to the REST server from HLL exits you must code the application selectivity yourself in the exit code, i.e. check a list of 'REST server active' applications in your exit code prior to making a call to CMNURIRX (more information on the mechanisms used to call the REST server from both HLL exits and batch job steps is given below).

REST Services

Standard ZMF Web Services provides comprehensive coverage of ZMF Services. The learning curve is steep, as the client must implement session management to wrap requested transactions with logon and logoff requests. This is the proper tool to use for full-function clients. ZMF REST Services (ZRS) provide REST API's for ZMF Services. ZRS is a wrapper on ZMF XML Services and works in a manner similar to the XML Prototype tool in TSO. Like the prototype tool, each call includes authentication to wrap the call with logon and logoff requests. This is standard processing for REST, as by definition, each transaction is stateless.

ZRS Requests follow this standard URL Pattern: http://host:port/context/Request

- context - is the servlet context where ZRS is deployed (typically zmfrest)
- Request is the service name to call. This is an alias to the SERVICE/SCOPE/MESSAGE implemented in ZMF. For example, APPLPARMS represents PARMS/APP/LIST

ZRS accepts input parameters from a QUERY String or JSON body.

Parameters use standard variable names.

Output may be in XML or JSON format. The 'accept' header supplied by the client dictates the output format. Some services will override this as appropriate.

REST Services Extensions

A checkin request typically references a temporary file or data set that contains the source content to be stored in a ChangeMan package. For REST Services clients, content may now be included in the HTTP Request body.

The REST Services checkin extension:

1. Permits a new value to be specified for the chkinSourceLocation parameter. This parameter may be set to 'B' to indicate content is included in the request body.
2. Adds a new parameter "codepage" to the checkin request. This parameter identifies the code page of the included content. Content is translated to the ChangeMan codepage by REST Services. The default value is UTF-8.
3. Validates that the package component is locked by the user issuing the checkin request. This validation may be disabled by specifying CHECKIN_LOCK_REQUIRED=N as a REST Services parameter in the ZMFPARMS data set.

4 Validates that the package component is not being edited by another user. This validation may be disabled by specifying CHECKIN_TEST_ENQUEUE=N as a REST Services parameter in the ZMFPARMS data set.

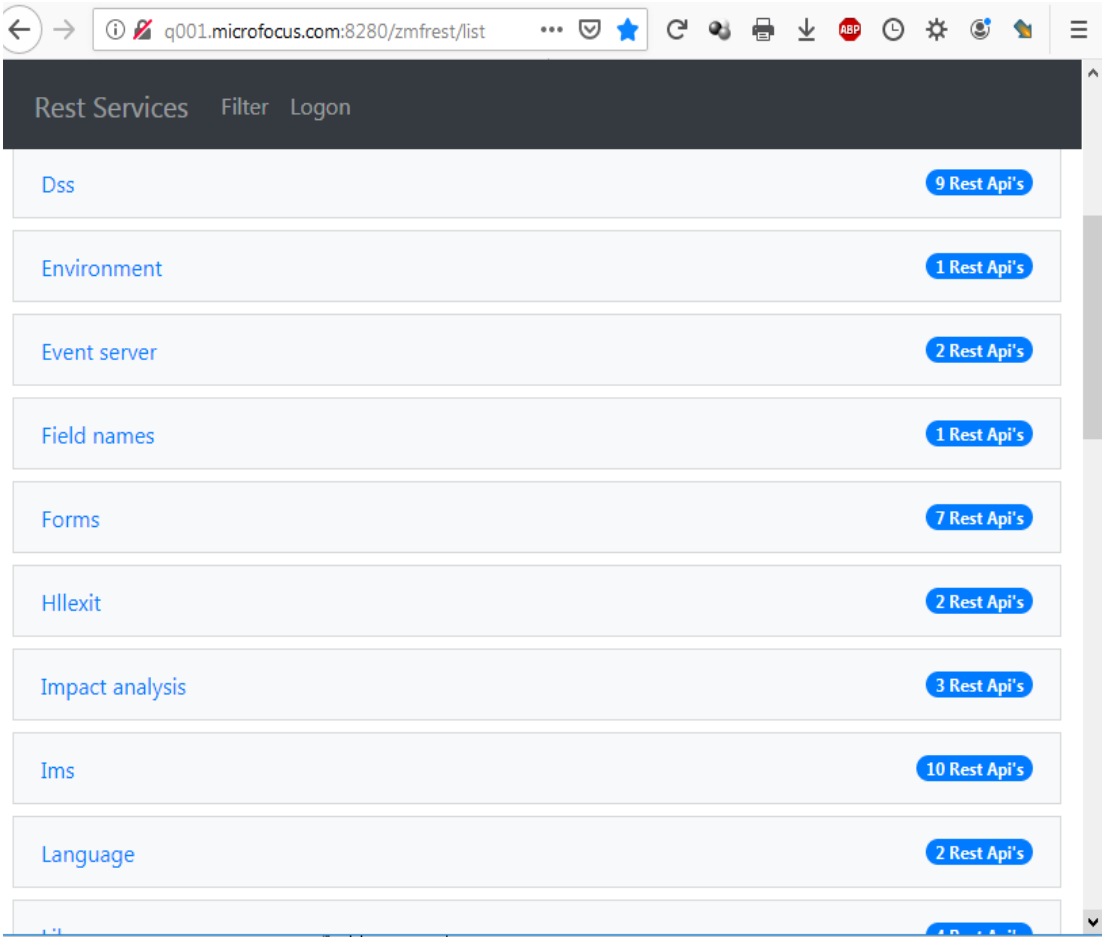
REST interface

To get a detailed list of the required/optional parameters (aka API variables) applicable to a specific REST API, this can be obtained with the following URI.

/zmfrest/LIST

For example within a browser: http://d001.microfocus.com:8085/zmfrest/list

From there you may filter the display to show additional required / optional parameters:



The REST API web application can be used to explore and prototype the ZMF REST API calls.

To place a call in a program or script you need just use the relevant url and supply the parameters either as query parms or as a JSON body.

To authenticate your request at the target ZMF you must place your RACF userid and password into the authentication header of the request being sent to the server.

The header value should look like this:

`'Authorization':'Basic <encoding of userid:password>'`

where <*encoding of userid:password*> is a base 64 encoding of <*userid*>:<*password*>.

REST Services Table

Method	Name	Category	Description
GET	approve/search	Package	Search for packages pending approval
GET	approver/appl	Approver, Application	Get application approver information
GET	approver/package	Approver, Package	Get package approver information
GET	calendar	Calendar	Get installation calendar detail
GET	calendar/summary	Calendar	Get installation calendar summary
GET	change-description	Component	Get component change description
GET	component	Component, Package	Get package component information (non-generated components, e.g. src,cpy,pds)
GET	component-description/appl	Component, Application	Get application level component description information
GET	component-description/appl/find	Component, Application	Find the application level description which applies to a component
PUT	component-description/appl/lripple	Component, Application	Get baselined application level component description information
PUT	component-description/appl/ripple	Component, Application	Baseline application level component description information
PUT	component-description/appl/rriple	Component, Application	Reverse baseline application level component description information
GET	component-description/global	Component, Global	Get global component description
GET	component-owner/appl	Component, Application	Get component application ownership rules
GET	component-owner/appl/check	Component, Application	Check component application ownership rules
GET	component-security/appl	Component, Application	Get application level component security rules
GET	component-security/appl/check	Component, Application	Check application level component security rules
GET	component-security/appl/find	Component, Application	Find the application level security rule which applies to a component
GET	component-security/global	Component, Global	Get global component security information

Method	Name	Category	Description
GET	component/browse	Component	Browse a component
PUT	component/build	Component	Build a like-src component
PUT	component/checkin	Component	Check a component into a package
PUT	component/checkout	Component	Check a component out into a package
GET	component/checkouv	Component	Validate package status prior to checkout
GET	component/compare	Component	Compare components
GET	component/load	Component, Package	Get package generated component information (load modules etc.)
PUT	component/lock	Component	Lock component
GET	component/packagelist	Component, Package	Get a list of package component information
GET	component/ promotionhistory	Component, Promotion	Get promotion history information for a component
PUT	component/rebuild	Component	Mass component rebuild
PUT	component/recompile	Component	Recompile a component
PUT	component/relink	Component	Relink a component
PUT	component/rename	Component	Add baseline component rename information to a package
PUT	component/scratch	Component	Add baseline component scratch information to a package
GET	component/source-include	Component	Get source includes (copybooks) information
GET	component/static-include	Component	Get static subcomponent information
PUT	component/unlock	Component	Unlock a component
GET	component/utility	Component, Package	Get package utility (scratch, rename) information
GET	component/version	Component, SSV	Get save staging versions (SSV) information
PUT	component/version	Component, SSV	Retrieve a prior version of a package component from the SSV store
GET	component/worklist	Component, Package	Get package worklist information
GET	db2aplactv	Db2, Application	Get application DB2 active library information
GET	db2attr	Db2	Get DB2 remote package attribute information
GET	db2logical/appl	Db2, Application	Get application DB2 logical subsystem information
GET	db2logical/global	Db2, Global	Get global DB2 logical subsystem information
GET	db2physical	Db2, Global	Get global DB2 physical subsystem information
GET	db2token/appl	Db2, Application	Get application DB2 general token information
GET	db2token/global	Db2, Global	Get global DB2 general token information

Method	Name	Category	Description
GET	dss	Dss	Lists member, hash token and directory entries for a dataset.
PUT	dss	Dss	Allocate a dataset
DELETE	dss	Dss	Delete a dataset
GET	dss/baseline/mbrstats	Dss	Get member stats for a baseline component
PUT	dss/expand	Dss	Expand baseline libraries
GET	dss/info	Dss	Get dataset information
GET	dss/ispfinfo	Dss	Get the allocation information to be used for allocating an ISPF file tailoring or other temporary dataset.
DELETE	dss/member	Dss	Delete a member of a dataset
GET	dss/stclist	Dss	Lists the datasets allocated to the requested ddname in the ZMF started task.
GET	fieldnames	Field names	Get ZMF field name information
GET	file	File	List Files
PUT	file	File	Change File Access
POST	file	File	Create File
DELETE	file	File	Delete File
GET	file/access	File	List File Access
GET	file/copy	File	Copy File
GET	file/dirlist	File	List Directories
GET	file/download	File	Download File
PUT	file/export	File	Export File
GET	file/filelist	File	List Files
PUT	file/import	File	Import File
PUT	file/link	File	Link Files
PUT	file/lock	File	Lock File
POST	file/mkdir	File	Make Directory
GET	file/realpath	File	Realpath
PUT	file/rename	File	Rename File
DELETE	file/rmdir	File	Remove Directory
GET	file/scan	File	Scan File(s)
PUT	file/unlock	File	Unlock File
PUT	file/upload	File	Upload Files
GET	forms/global	Forms, Global	Get global forms information
GET	forms/package	Forms, Package	Get package forms information
PUT	forms/package/approve	Forms, Package	Approve package forms

Method	Name	Category	Description
GET	forms/package/ comment	Forms, Package	Updates the comments for an online form in a package.
GET	forms/package/detail	Forms, Package	Detail package forms
PUT	forms/package/reject	Forms, Package	Reject package forms
PUT	forms/package/submit	Forms, Package	Submit package forms
GET	hfs/hash	Hfs	Get hfs file hash token
GET	history	Component	Get component history
GET	history/base	Component	Get latest baselined component history
GET	history/concurrent	Component	Get history for a component active concurrently in different packages
GET	history/current	Component	Get current history for a component
GET	history/language	Component	Get standard language identifier for a component
GET	history/listload	Component	Get history for target components associated with a source component
GET	history/listname	Component	Get a component name-libtype list
GET	history/package	Component	Get a list of component history for a package
GET	history/short	Component	Get a list of history for components in motion
GET	impact/bun	Impact analysis	Get impact analysis Baseline Unique Number (BUN) information
GET	impact/component	Impact analysis	Get impact analysis component information
GET	impact/row	Impact analysis	Lists component impact analysis information
GET	imscrgn/appl	Ims, Application	Get application level IMS control region information
GET	imscrgn/global	Ims, Global	Get global IMS control region information
GET	imsdbd/appl	Ims, Application	Get application level DBD override information
GET	imsdbd/global	Ims, Global	Get global DBD override information
GET	imsdbd/package	Ims, Package	Get package level DBD override information
GET	imspsb/appl	Ims, Application	Get application level PSB override information
GET	imspsb/global	Ims, Global	Get global PSB override information
GET	imspsb/package	Ims, Package	Get package level PSB override information
GET	language/appl	Language, Application	Get application language information
GET	language/global	Language, Global	Get global language information
GET	library/baseline	Library	Get baseline library information
GET	library/production	Library	Get production library information
GET	library/promotion	Library	Get promotion library information
GET	library/promotion/site	Library	Get promotion library site information

Method	Name	Category	Description
GET	libtype/appl	Libtype, Application	Get application level library type information
GET	libtype/global	Libtype, Global	Get global library type information
GET	libtype/package	Libtype, Package	Get package level library type information
GET	log	Log	Lists activity log entries
POST	log	Log	Creates an activity log entry
GET	notifyfile/download	Notify file	Downloads the global notification file.
PUT	notifyfile/upload	Notify file	Uploads the global notification file.
POST	package	Package	Create a package
DELETE	package	Package	Delete a package
GET	package/affapls	Package, Application	Get package affected applications information
PUT	package/approve	Package	Approve/reject a package
PUT	package/attach	Package	Attach a package to a release
PUT	package/audit	Package	Submits a job to audit a package
PUT	package/backout	Package	Backout a package
PUT	package/check/promote	Package	Checks if a promote request is valid without performing the promotion.
PUT	package/cleanup/ promote	Package	Performs promotion cleanup at installation. This is effectively a full demotion from all levels for a particular site.
GET	package/cmpdesc	Package	Get component descriptions from package records
GET	package/component/ integrity	Package	Check the integrity of component metadata for a package
PUT	package/demote	Package	Demote a package
PUT	package/detach	Package	Detach a package from a release
PUT	package/forms/refreeze	Package	Refreeze package forms
PUT	package/forms/refreeze	Package	Unfreeze package forms
PUT	package/freeze	Package	Freeze a package
GET	package/gendesc	Package	Get package level general descriptions
GET	package/genparms	Package	Get package parameters
GET	package/implinst	Package	Get package implementation instructions
GET	package/imsacb	Package, IMS	Get package IMS ACB information
GET	package/imscrgn	Package, IMS	Get package IMC control region information
GET	package/participating	Package	Get participating package information
GET	package/prmcmp	Package	Get package promoted components information
PUT	package/promote	Package	Promote a package

Method	Name	Category	Description
GET	package/ promotionhistory	Package	Get package promotion history
PUT	package/promotionlock	Package	Obtain a package promotion lock
GET	package/ promotionoverlay	Package	Lists components that would be overlayed if a package was promoted.
PUT	package/ promotionunlock	Package	Release a package promotion lock
GET	package/reasons	Package	Get backout/revert reasons for a package
PUT	package/refreeze/ nonsource	Package	Refreeze non_src package components
PUT	package/refreeze/ parameters	Package	Refreeze package parameters
PUT	package/refreeze/sites	Package	Refreeze package site information
PUT	package/refreeze/ source	Package	Refreeze package src-lod information
PUT	package/refreeze/utility	Package	Refreeze utility (scratch, rename) package information
PUT	package/refreeze/utility	Package	Unfreeze utility (scratch, rename) package information
PUT	package/revert	Package	Revert a package
GET	package/schrecs	Package	Get CMN scheduler information
GET	package/search	Package	Search for packages using selection criteria
GET	package/search/limbo	Package	Search for packages in limbo
PUT	package/submit	Package	Request rebuild of installation jobs for package
GET	package/summary	Package	Get summary totals for packages
GET	package/syslib	Package	Get package SYSLIB information
PUT	package/unfreeze/ nonsource	Package	Unfreeze non_src package components
PUT	package/unfreeze/ parameters	Package	Unfreeze package parameters
PUT	package/unfreeze/sites	Package	Unfreeze package site information
PUT	package/unfreeze/ source	Package	Unfreeze package src-lod information
GET	package/userrecords	Package	List package user records
GET	parameters/appl	Parms, Application	Get application parameters
GET	parameters/global	Parms, Global	Get global parameters
GET	procedures/appl	Procs, Application	Get component compile procedures defined to an application
GET	procedures/designated/ appl	Component, Application	Get application level component designated procedure rules

Method	Name	Category	Description
GET	procedures/designated/appl/check	Component, Application	Check application level component designated procedure rules
GET	procedures/designated/appl/find	Component, Application	Find the application level designated procedure rule which applies to a component
GET	procedures/designated/global	Component, Global	Get global component designated procedure information
GET	procedures/global	Procs, Global	Get component compile procedures defined globally
GET	reasons	Reasons	Get global backout or revert reasons
GET	release	Release	Get release information
GET	release/appl	Release, Application	Get release application setup information
GET	release/appl/promotion-definition	Release, Application	Get release application promotion setup information
GET	release/appl/search	Release, Application	Release Release Application Release Definitions
GET	release/appl/syslib	Release, Application	Get release application SYSLIB information
GET	release/approver	Release, Approver	Get release approvers information
GET	release/approver/area	Release, Approver, Area	Get release area approvers information
GET	release/approver/associated	Release, Approver	Get release area associated approvers information
GET	release/approver/global	Release, Approver, Global	Get release global approvers information
GET	release/area	Release, Area	Get release area information
GET	release/area/check-area	Release, Area	List components which may be eligible for check-in from an area
GET	release/area/check-package	Release, Area	List components which may be eligible for check-in from a package
GET	release/area/cim	Release, Area	Get release area component-in-motion metadata
GET	release/area/component-lock	Release, Area	List release area component locks
GET	release/area/component/demoted	Release, Area	List release area demoted components
GET	release/area/component/detail	Release, Area	List Release Area Component Details
GET	release/area/component/promoted	Release, Area	Get release area promoted component information
GET	release/area/component/scan	Release, Area	Get list of release area components (summary)

Method	Name	Category	Description
GET	release/area/ component/scan-all	Release, Area	Get list of release area components (detail)
GET	release/area/ component/summary	Release, Area	Sumary list of latest components in a release area
GET	release/area/hst	Release, Area	Get release area component history metadata
GET	release/area/iat	Release, Area	Get release area impact analysis metadata
GET	release/area/integrity/ detail	Release, Area	Assess the integrity of release area metadata (detailed results)
GET	release/area/integrity/ summary	Release, Area	Assess the integrity of release area metadata (summary results)
GET	release/area/start	Release, Area	Get release start area information
GET	release/area/syslib	Release, Area	Get release area SYSLIB information
GET	release/area/syslib/ allchk	Release, Area	Get release area SYSLIB libraries (check allocations)
GET	release/area/syslib/ allnoc	Release, Area	Get release area SYSLIB libraries (no allocation check)
GET	release/area/syslib/cpy	Release, Area	Get release area copybook SYSLIB information
GET	release/area/syslib/link	Release, Area	Get release area link deck (LCT) SYSLIB information
GET	release/area/syslib/load	Release, Area	Get release area program binder SYSLIB information
GET	release/area/syslib/no-src	Release, Area	Get release area SYSLIB information for all but like-SRC build mechanisms
GET	release/area/syslib/ source	Release, Area	Get release area SYSLIB information for source compiles
GET	release/area/test	Release, Area	Test a release area for consistency
GET	release/area/test/detail	Release, Area	Test a release area for consistency (detailed results)
GET	release/area/version-regression	Release, Area	Get release area component version regression information
GET	release/cim	Release	Get release component-in-motion metadata
GET	release/component/ check	Release	Check package components presence in all areas of a release
GET	release/hst	Release	Get release component history metadata
GET	release/iat	Release	Get release impact analysis metadata
GET	release/library	Release	Get release library information
GET	release/libtype	Release, Libtype	Get release area library type information
GET	release/libtype/bun	Release, Libtype	Get release libtype and BUN correlation information
GET	release/package	Release	List packages attached to a release
GET	release/package/check	Release	Check all package components presence in all areas of a release

Method	Name	Category	Description
GET	release/package/search	Release	No REST Service Description
GET	release/prior-release	Release	Get prior release information for this release
GET	release/reasons	Release	Get release backout and reject reason information
GET	release/release-link	Release	List Release LINK Definitions
GET	release/search	Release	Search release components
GET	release/sites	Release, Site	Get release site information
GET	release/test	Release	Test a release for consistency
GET	release/test/detail	Release	Test a release for consistency (detailed results)
GET	schedule	Schedule	Get CMN scheduler information
PUT	schedule/hold	Schedule	Hold a package in the CMN scheduler
GET	schedule/release	Schedule	Release a package in the CMN scheduler
GET	site/appl	Site, Application	Get application level site information
GET	site/global	Site, Global	Get global site information
GET	site/package	Site, Package	Get package site information
GET	skels/header	3Dskels, Global	Get 3D-skels header information
GET	skels/variables	3Dskels, Global	Get 3d-skels variable information
GET	system/db2	System, Db2	Lists DB2 subsystem names
GET	system/environment	System, Environment	Lists current system environment information
GET	system/product/users	System	Lists active users of a given product
GET	system/properties	System	Lists a variety of system type information
GET	system/security-group	System	Lists connected security groups

Method	Name	Category	Description
GET	user/notify	User	Sends a message to users. There a 2 general types of message; generic and predefined. With a generic message, the entire message text is supplied in the request. With predefined messages, the message text contains predefined text and substitutable variables. The variables are substituted with values either from the request or may be retrieved by the service. Messages may be sent to users using 4 methods: MVS/TSO send (MVSSEND), E-mail via Sernet and ECP Web server (SERNET), batch jobs (BATCH), or directly to a standard SMTP email server (EMAIL). MVSSEND: Sends a single line message to TSO users on the same LPAR as the ZMF started task. SERNET: Sends an e-mail via an ECP Web server. The ECP Web server must be installed and running. The TCP/IP address of the ECP Web server must be defined to the ZMF started task or supplied in the service request. BATCH: Creates a job using ISPF file tailoring and then submits the job. The appropriate request values and package information are made available to the file tailoring skelton in ISPF variables. File tailoring skelton CMN$$NTF is supplied as an example. EMAIL: Creates a formatted email letter and calls the Sernet SMTP agent (SERSMPTC).
GET	zmf-environment	Environment	Lists information about the ZMF environment.

ZMF supplied skeleton changes

General note on usage: The call to the REST server will wait for a response. The REST server will wait for a response from the process which it has been asked to initiate (i.e. a Jenkins job etc.). If the target process returns immediately then response times need not be an issue. If you wish the target process to perform significant processing (e.g. standards checking/testing etc.) then the response may be a long time in coming. In these batch processes this may not be a huge concern. However, contrast this with the similar note made at the start of the following section on HLL exit use of the REST server. Also note that the CMNURIBA program will return CC=12 for any http response from the REST server other than one of the 2xx series. Again, contrast this with the out-of-the-box support supplied for HLLX.

Two sample skeletons are provided to allow batch jobs generated by ZMF to call the REST server. The call itself is coded in *CMN$$EVT* and this skeleton can be imbedded anywhere. The majority of the out-of-the-box support for ZMF generated batch job event notification is supplied via skeleton *CMNEVT* (which imbeds CMN$$EVT).

Existing skeletons related to 'success' notification have been modified to imbed CMNEVT as required, these are:

CMN00
CMN00INS
CMNRPMB0

Further changes have been needed to a group of skeletons which imbed CMN00INS twice, so that the REST server call is only made for a success notification. These are:

CMN20, 20I, 20T, 20TI
CMN55, 55I

Certain event notifications are generated directly via the ZMF file tailoring mechanism (i.e. the file tailoring includes CMN$$EVT directly). This is to avoid changing more existing skeletons than is absolutely necessary, these event ids are:

50 Package audit
52 Release area audit
57 Package audit autoresolve submitted
58 Release area autoresolve submitted
78 Checkin to release area
100 Build begins
101 Build ends

It may be that one wishes to move the location of the event notification away from that generated by ZMF file tailoring into a skeleton of their choice. To do this the original file tailoring must be neutralized and a sample skeleton, CMN$$EVX, has been provided to show how this can be done. Comparison of CMN$$EVX and CMN$$EVT will reveal the changes required.

New variables are defined to indicate whether the REST server is active in general and active for specific events. These variables are set by the job generation program (i.e. CMNVFTLR, CMNVPRFT, CMNVRPFT, CMNVPIJB) and they will be available to all skeletons. If the user has chosen not to enable support for the REST server in general or for a specific event then the imbedded skeleton CMN$$EVT will do nothing. The call to the REST server is made using CMNURIBA (see example above) and every standard variable is passed (whether it is available or not), the REST server will work out which variables, from the list, it will use for a specific event.

The new ISPF variables are these:

EVTACTV The REST server is active if this is set to Y
EVTADDR The DNS/ip address of the REST server
EVTPORT The port number on which the REST server is listening
EVTCTX The context for the event servlet (default is zmfevent)
EVTNOxx xx or xxx is the event number (room for 3 digits if necessary). This variable will be set to Y if the event is active (e.g. EVTNO12=Y)

The ZMF program which is generating these variables will check with the REST server to see if the specific event we are processing is active. If it is active then the EVTNOxx variable will be set to Y, else it will be set to N.

Example of SKEL to IMBED the CMN$$EVT SKEL:

```
)CM
)CM NOTIFICATION 12
)CM
)SEL &EVTACTV EQ Y AND &EVTNO12 EQ Y
)SETF &EVENTID = &STR(12)
)IM CMN$$EVT
)ENDSEL &EVTACTV EQ Y AND &EVTNO12 EQ Y
)CM
)CM  End Of Notification 12

*Example SKEL For Event Client:*


//*)IM CMN$$EVT &EVENTID
)SEL &LISTNO EQ &Z
)SET LISTNO = 0
)ENDSEL &LISTNO EQ &Z
)SET LISTNO = &LISTNO + 1
//*
//* Call the REST server for ZMF event number &EVENTID
//*
//EVENT&EVENTID EXEC PGM=CMNURIBA
//*
)SEL &EVENTID NE 100 AND &EVENTID NE 12
//SYSPRINT DD  SYSOUT=*
)ENDSEL &EVENTID NE 100 AND &EVENTID NE 12
)SEL &EVENTID EQ 100 OR &EVENTID EQ 12
//SYSPRINT DD  DISP=(,PASS),DSN=&&&&LIST9&LISTNO,
//             &DEFNVKW=&DEFNVUN,SPACE=(TRK,(1,3),RLSE),
//             DCB=(RECFM=FA,LRECL=133,BLKSIZE=0)
)ENDSEL &EVENTID EQ 100 OR &EVENTID EQ 12
//JSONIN   DD  DATA,DLM=@@
{
  "EVENT"          : "&EVENTID.",
  "USERID"         : "&USER.",
  "APPL"           : "&PROJECT.",
  "PACKAGE"        : "&PKGNAME.",
  "SITE"           : "&RMTSITE.",
  "RELEASE"        : "&RLSNAME.",
  "RELEASEAREA"    : "&ARENAME.",
  "PROMOTIONNAME" : "&PROMNME.",
  "PROMOTIONLEVEL": "&PROMLVL.",
  "LIBTYPE"        : "&CMPTYPE.",
  "COMPONENT"      : "&CMPNAME."
}
@@
//SYSIN    DD  *
Server=&EVTADDR
Port=&EVTPORT
Context=&EVTCTX
Method=POST
/*
//*
```

The supplied skeleton, CMN$$EVT, can be further modified should one wish to save the JSON response body as part of the job output. This is not implemented as delivered as, in most cases, one will not wish to do this and we want to keep the skeletons as simple as possible. The way to do this is, as mentioned in an earlier section, to add the CMNRSPNS

DD statement to the CMNURIBA step and add a following PRETTY print step. Sample CMN$$EVT modifications are shown here:

```
//*)IM CMN$$EVT &EVENTID
//* Modified to produce JSON body output
//* Modified to produce JSON body output
//* Modified to produce JSON body output
)SEL &LISTNO EQ &Z
)SET LISTNO = 0
)ENDSEL &LISTNO EQ &Z
)SET LISTNO = &LISTNO + 1
//*
//* Call the REST server for ZMF event number &EVENTID
//*
//EVENT&EVENTID EXEC PGM=CMNURIBA
//*
)SEL &EVENTID NE 100 AND &EVENTID NE 12
//SYSPRINT DD  SYSOUT=*
)ENDSEL &EVENTID NE 100 AND &EVENTID NE 12
)SEL &EVENTID EQ 100 OR &EVENTID EQ 12
//SYSPRINT DD  DISP=(,PASS),DSN=&&&&LIST9&LISTNO,
//             &DEFNVKW=&DEFNVUN,SPACE=(TRK,(1,3),RLSE),
//             DCB=(RECFM=FA,LRECL=133,BLKSIZE=0)
)ENDSEL &EVENTID EQ 100 OR &EVENTID EQ 12
//CMNRSPNS  DD DISP=(,CATLG),DSN=CMNDEV.&USER..JSON.TEMP&EVENTID.,
//             SPACE=(CYL,(1,1)),UNIT=SYSDA
//JSONIN   DD  DATA,DLM=@@
{
  "EVENT"          : "&EVENTID.",
  "USERID"         : "&USER.",
  "APPL"           : "&PROJECT.",
  "PACKAGE"        : "&PKGNAME.",
  "SITE"           : "&RMTSITE.",
  "RELEASE"        : "&RLSNAME.",
  "RELEASEAREA"    : "&ARENAME.",
  "PROMOTIONNAME" : "&PROMNME.",
  "PROMOTIONLEVEL": "&PROMLVL.",
  "LIBTYPE"        : "&CMPTYPE.",
  "COMPONENT"      : "&CMPNAME."
}
@@
//SYSIN    DD  *
Server=&EVTADDR
Port=&EVTPORT
Context=&EVTCTX
Method=POST
/*
//*
//PRETTY   EXEC PGM=IKJEFT01,REGION=0M
//REMOVE    DD DISP=(OLD,DELETE),DSN=CMNDEV.&USER..JSON.TEMP&EVENTID
//SYSEXEC   DD  DISP=SHR,DSN=SYS1.SAMPLIB
)SEL &EVENTID NE 100 AND &EVENTID NE 12
//SYSTSPRT DD  SYSOUT=*
)ENDSEL &EVENTID NE 100 AND &EVENTID NE 12
)SEL &EVENTID EQ 100 OR &EVENTID EQ 12
//SYSTSPRT  DD DISP=(,PASS),DSN=&&&&LIST8&EVENTID.,
//             UNIT=SYSALLDA,SPACE=(CYL,(1,5),RLSE),
//             DCB=(RECFM=FBA,LRECL=133,BLKSIZE=27930)
)ENDSEL &EVENTID EQ 100 OR &EVENTID EQ 12
//SYSTSIN   DD  *
HWTJSPRT CMNDEV.&USER..JSON.TEMP&EVENTID
//*
```

The file tailoring programs have been changed to allow for the supplied skeletons to call the REST server as necessary. They each call the REST server during initialization to see whether the REST server is active in general and if the events they will be generating are

subscribed to. The triggers to imbed the calls to the REST server will only be active if the REST server is active and the specific event is subscribed to. Those who don't use the REST server will see no changes to the generated JCL.

CMNVFTLR prepends the build job JCL stream with a call to the REST server for event 100 (build job begins), it appends the JCL with a call to the REST server for event 101 (build job ends). It also sets variables for event no 12 (component activation) which prompt the generated JCL to imbed CMN$$EVT alongside the SUCCESS step. It also appends the package audit JCL stream with a call for event no 50 (package audit).

CMNVPRFT and CMNVRPFT set variables for event no 44 (package demote) and 48 (package promote).

CMNVPIJB sets variables immediately prior to file tailoring the relevant job stream into the 'x' dataset member. It does this for events 01 (package backout), 02 (package install), 03 (temporary package cycle), 10 (package revert), 15 (baseline ripple), and 16 (reverse ripple).

Sample REXX HLL exit code

General note on usage: All HLL exits may have an impact on the client user interface. Especially, for example, if an HLL exit does a significant amount of processing the user will be ‘locked’ in their interaction with ZMF in general. This may cause a frustrating end-user experience. For that reason it is recommended (and the samples supplied follow this recommendation) that HLL exits be used to simply notify the REST server of events and not to expect significant synchronous processing by the target process before returning to the REST server. Use of the post-service HLL exits (supplied in the samples) is recommended for event notification purposes. The target process should return immediately to the REST server even if significant processing has been initiated. The user will remain locked by HLLX until the target process has returned to the REST server and the REST server has, in turn, returned to the HLL exit. Also note that the out-of-the-box support is placed after the function service has completed (i.e. we are notifying the REST server that something has already happened). No check is (or should be) made on the success or otherwise of the call to the REST server. There is no point as the ZMF function has already been completed and whatever happens the other side of the REST server is of no consequence to that ZMf action. Note that the REST server will differentiate between the call origins for the same event so that the target process can decide whether to undergo significant synchronous processing for the event (e.g. as driven from a batch job step via the zmfevent/event/skel urn) or not (e.g. when driven from an HLL exit via the zmfevent/event/hllx urn).

When the HLLX address space starts up (and when a HLLX RELOAD is requested) the ZMF settings for the REST server are passed to it. If the REST server is active then it will query all HLLX supported events to see if there are subscribers. For all subscribed-to events the relevant (HLLX TCA) variable will be set to Y. When the HLL REXX exit is called CMNREXCI (our REXX initialization exit) has access to all these variables and will set the relevant REXX variables for use by the target HLL exit.

The supplied sample exit points for calling the REST server from an HLL exit are these:

Event	HLL exit name	Description	Sample exit name
01	RVRT01XB	Backout Package	HXRRVEV
10	RVRT01XM	Revert Package	HXRRVEV
20	APRV01XM	Approve Package	HXRAPEV
30	APRV01XM	Reject Package	HXRAPEV
40	FREZ01XM/FREZ01XR	Freeze Package	HXRFREV
42	FREZ01XU	Selectively Unfreeze Package	HXRFREV
44	PRDM01XD	Demote Package	HXRPREV
48	PRDM01XP	Promote Package	HXRPREV
50	AUDT01JB	Audit Package	HXRAUEV
64	SCRN01XM	Scratch component	HXRSCEV
66	SCRN01XM	Rename component	HXRSCEV
67	BULD01XL	Relink component	HXRBUEV
78	RCKI01CI	Checkin to area is complete	HXRRCEV
80	PCRE01XM	Create Package	HXRPCEV
82	CKOT01XM	Checkout Component	HXRCKEV
84	BULD01XC	Stage Component	HXRBUEV
86	BULD01XD	Delete component from pkg	HXRBUEV
87	CKOT01XM	Checkout from release	HXRCKEV
88	PCRE01XM	Copy forward package	HXRPCEV
100	BULD00XB	Pre-build of component	HXRBUEV
101	BULD01XB	Post-build of component	HXRBUEV

The REXX variable '**evSrvActive**' is defined for all HLL exits and set to Y or N to denote whether HLLX has found the REST server to be active. REST server exit code should only ever be executed if evSrvActive='Y'.

The following REXX variables are defined and set *only if* evSrvActive='Y':

evSrvAddress The DNS/IP address of the REST server (e.g. in our test cases this was set to 'd001.microfocus.com')

evSrvPort The port on which the REST server is listening (e.g. 09992 in our test case)

evSrvContext The context for the target event servlet (default is zmfevent)

evSrvEvent/*nn*/ where /*nn*/ or /*nnn*/ is set to the specific event id (e.g. evSrvEvent01). These variables are set to Y or N depending on whether or not the event is active (i.e. subscribed to at the eventserver).

Look at the sample code for an HLL exit that is invoked at 5 HLL exit points relating to 5 different events, in member HXRBUEV of the SAMPLE library.

Support for custom processes

Note that there is nothing to prevent one from placing a call to the REST server in any skeleton or HLL exit if they so wish. We are providing out-of-the-box solutions for what we consider to be the most useful events but one may have requirements that we haven't catered for.

If the supplied sample skeletons and/or HLL exits do not provide the support that a site is looking for then they can use the supplied /examples/ and place calls to the REST server wherever they like. It would be in the interest of the user, and users in general, if they communicated what they are doing to us so that we can take a view on including that support as a sample in future releases.

External 3rd Party Dependencies

IBM z/OS Client Web Enablement toolkit

The CMNURIxx utilities rely on the use of the z/OS Client Web Enablement toolkit which is supplied as part of z/OS. However, this use also has certain requirements of the environment in which it runs. The userid under which it is running needs to have an OMVS segment defined. The toolkit code itself runs under a POSIX(ON) LE enclave (which it will establish itself if not present).

For further information on the IBM z/OS Client Web Enablement toolkit refer to the IBM documentation: z/OS Client Web Enablement toolkit https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.ieac100/ieac1-client-web-enablement.htm

IBM Application Transparent Transport Layer Security AT-TLS

All secure communication (SSL) on z/OS must be implemented using AT-TLS. This may include communication between:

Event Clients to REST Services, REST Services to Subscribers, Clients to REST Services, REST Services to ChangeMan ZMF.

JAVA V8 for Z/OS

JAVA V8 is required to run Tomcat and ZMF Servlets on Z/OS.

JZOS Batch Launcher

The JZOS Batch Launcher is required to run Tomcat and Java programs on Z/OS

Jenkins 2.164 (Minimum)

Minimum version is 2.164

Jenkins should be run with a V8 JRE.

Appendix A

ZMF Utilities Notes

This appendix presents more information about facilities available.

CMNURIBA (Easy access to http methods from ZMF batch processes)

This program is designed to allow easy access to HTTP methods (GET, POST, etc) from traditional batch processes. It makes use of the z/OS HTTP Web Enablement Toolkit (supplied as part of the operating system). As such, it requires the userid under which it is running to be defined with an OMVS segment. It also establishes (or re-uses) a POSIX(ON) LE enclave. It works with URI's, HTTP headers, and JSON bodies.

The utility can be used to request an external action via an HTTP request, wait for the response, and decide whether to continue with the job based on that response.

Direction on what CMNURIBA is to do is given via SYSIN parameters, these are described below:

Parameter	Value
SERVER=	Specify the IP address or DNS name which will process the HTTP method
PORT=	Defines the port on which this server is listening for us
CONTEXT=	REST servlet context, default is zmfrest
HTTPTIMEOUT=	Defines how long, in seconds that we should wait for a response (default is 300, i.e. 5 mins)
TRACE=	YES/NO. Verbose output produced with TRACE=YES - also requires the HTPTRACE ddname to be allocated. Internally produced trace output is written to SYSPRINT, HTTP toolkit generated trace output is written to HTPTRACE.
URN=	Specifies the target resource name for the operation (default is/<context>/event/skel or /zmfevent/event/skel if no context specified, i.e. the REST server as used by skeleton processing)
METHOD=	Defines the HTTP method to be used (only GET and POST needed at this time)
PARM=	Defines the query parameter to be appended to the header, there can be many of these (see example below).

Also, if a JSON body is required on the request (e.g. for a POST method) this is input (as is) via the JSONIN dd statement (see example below). Note that each JSON clause must be completed within 80 bytes at this time (e.g. like a card image). This may change in future should the need for longer clauses be identified.

Processing overview:

SYSIN is read to establish the parameters to be used in this request. All sysin keywords must start on a new line and must not extend beyond column 72. Some of the parameters have the potential to be longer than this allows for, these are SERVER, URN, and any PARMs. This potential is catered for by using an asterisk as a continuation character. All

text including and after the asterisk is ignored and the next sysin card image is read. All text from the beginning of the card image (including spaces) is appended to the text already read in for this keyword. For example:

SERVER=d001.micro*
focus.com

is the same as

SERVER=d001.microfocus.com

Any query parameters are appended to the URN prior to issuing the HTTP request. For a METHOD=POST request, if the JSONIN DD statement is present then we build a JSON body to passed along with the POST headers.

An attempt is made to connect to the target server:port and, if successful, the relevant request is sent and we await confirmation from the server. Any response is checked for a 'good' status code (2xx) which will result in a RC=0 for the job step, else we have an RC=12. If there is a bad response then the response body (if any) is echoed out in SYSPRINT. If TRACE=YES is on then the response body is written to SYSPRINT regardless of the result. Note that a future enhancement could be to allow the user to define what is an acceptable response.

Example of JCL for GET request:

This example issues a GET request to a server with query parms. Here's some JCL for the 'activate component' event, this would be inserted as a batch job step in the build job:

```
//GOGOGO   EXEC PGM=CMNURIBA,REGION=0M
//*
//SYSPRINT  DD  SYSOUT=*
//SYSUDUMP  DD  SYSOUT=*
//HTPTRACE  DD  DISP=SHR,DSN=WSER58.HTTP.TRACE.OUTPUT
//SYSIN     DD  *
 Server=d001.microfocus.com
 Port=8085
 Trace=YES
 Method=GET
 Parm=EVENT=12
 Parm=PACKAGE=ZSRV000123
 Parm=APPL=ZSRV
 Parm=LIBTYPE=JAV
 Parm=COMPONENT=TESTSRC
```

The HTPTRACE dataset is a sequential file with RECFM=V,LRECL=1028,BLKSIZE=1032

The above request is converted into a connection, to
http://d001.microfocus.com:8085
and the HTTP GET method is issued using this connection, targeted at the following URN:
/zmfevent/event/skel?EVENT=12&PACKAGE=ZSRV000123&APPL=ZSRV&LIBTYPE=JAV&COMPONENT=TESTSRC

Example of JCL for POST request with JSON body:

```
//GOGOGO   EXEC PGM=CMNURIBA,REGION=0M
//*
//SYSPRINT  DD  SYSOUT=*
//SYSUDUMP  DD  SYSOUT=*
//HTPTRACE  DD  DISP=SHR,DSN=WSER58.HTTP.TRACE.OUTPUT
//JSONIN    DD  DATA,DLM=@@
 {
   "EVENT"    : "12",
   "PACKAGE"  : "ZSRV000123",
   "APPL"     : "ZSRV",
   "LIBTYPE"  : "JAV",
   "COMPONENT": "TESTSRC"
 }
 @@
//SYSIN     DD  *
 Server=d001.microfocus.com
 Port=8085
 Trace=YES
 Method=POST
```

Checking the availability of the REST server to receive event notifications

Sample JCL member RSTCHECK can be used to, in general, check the availability of the REST server and, specifically, check whether a webhook for a specific event id is

subscribed to. See the listing of RSTCHECK below for further details:

```
//jobname  JOB (account),'Check REST Server',  <=== Change Accordingly
//             CLASS=?,NOTIFY=?,               <=== Change Accordingly
//             MSGCLASS=?                      <=== Change Accordingly
//*******************************************************************
//*
//* This job tests the connection to the REST server in general and,
//* specifically, whether a particular event is active (i.e. is
//* subscribed to).
//*
//* The operation is traced (in case there are problems to resolve)
//* and the trace output is written to the HTPTRACE ddname.
//*
//* <your.server.address> and <its port> must be replaced with values
//* for your particular implementation.
//*
//* The supplied JCL tests whether the skel notified event id 52 is
//* active, but you can test of any event id you wish by changing
//* the number.
//*
//* If the event is active then the step rc will be 0, else 12.
//*
//* Replacing skel in the URN by hllx or log will test whether
//* hllx or log notified events are active, i.e. one of
//*
//* URN=/zmfrest/query/skel/52
//* URN=/zmfrest/query/hllx/52
//* URN=/zmfrest/query/log/52
//*
//* You can test whether an event is subcribed to for a specific
//* application by adding the appl as a filter, e.g.
//*
//* URN=/zmfrest/query/skel/52?appl=DEMO
//*
//*******************************************************************
//JOBLIB   DD  DISP=SHR,DSN=somnode.CMNZMF.LOAD
//         DD  DISP=SHR,DSN=somnode.SERCOMC.LOAD
//*
//*
//DELTRACE EXEC PGM=IEFBR14
//DD1       DD DISP=(MOD,DELETE),UNIT=SYSDA,SPACE=(TRK,0),
//             DSN=yourhlq.HTTP.TRACE.OUTPUT
//*
//TSTEVSRV EXEC PGM=CMNURIBA,REGION=0M
//*
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//HTPTRACE  DD DISP=(,CATLG),DSN=yourhlq.HTTP.TRACE.OUTPUT,
//             UNIT=SYSDA,SPACE=(CYL,(1,10),RLSE),
//             DCB=(RECFM=V,LRECL=1028,BLKSIZE=0)
//SYSIN     DD *
Server=<your.server.address>
Port=<its port>
Context=<REST servlet context, default is zmfrest>
Trace=YES
Method=GET
URN=/zmfrest/query/skel/52
```

The following shows the same job steps with specific values. Note that if the REST server is available and the specific event id is subscribed to then the job will receive a 200 http code from the server and the step will end with cc=0. If the event id is not subscribed to

then http code 418 will be received and the job step will end with cc=12. If the REST server cannot be contacted then some other http code may be presented and further information in the trace dataset may be of use.

```
//*
//DELTRACE EXEC PGM=IEFBR14
//DD1       DD DISP=(MOD,DELETE),UNIT=SYSDA,SPACE=(TRK,0),
//             DSN=WSER58.HTTP.TEMP.OUTPUT
/*
//*
//TSTEVSRV EXEC PGM=CMNURIBA,REGION=0M
//*
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//HTPTRACE  DD DISP=(,CATLG),DSN=WSER58.HTTP.TEMP.OUTPUT,
//             UNIT=SYSDA,SPACE=(CYL,(1,10),RLSE),
//             DCB=(RECFM=V,LRECL=1028,BLKSIZE=0)
//SYSIN     DD *
Server=d001.microfocus.com
Port=09992
Trace=YES
Method=GET
URN=/zmfevent/query/skel/52
```

Formatting JSON responses from the REST server

Any response from the REST server is, by default, echoed in SYSPRINT via 100 byte wrapped-around output. Normally you may not be interested in anything other than the return code from the REST server. However, in some circumstances you may have invoked a process that returns a result set/messages that you wish to keep as part of, for example, the build output for a component. If the response is supplied as JSON then we can use the IBM supplied 'pretty print' mechanism (SYS1.SAMPLIB(HWTJSPRT)) to format the JSON into something more readable. To do this you need only add a CMNRSPNS dd statement to the CMNURIBA step to write the response to a named temporary file and following this with an execution of HWTJSPRT on this named temporary file. Note that the response is no longer written to SYSPRINT in this case. An example of doing this for a

specific event 100 invocation is shown below, the extra JCL statements are the CMNRSPNS DD statement in the EVENT100 step and the whole of the PRETTY step.

```
//*
//* Call the REST server for ZMF event number 100
//*
//EVENT100 EXEC PGM=CMNURIBA
//*
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//CMNRSPNS  DD DISP=(,CATLG),DSN=WSER58.JSON.TEMPOUT,
//             SPACE=(CYL,(1,1)),UNIT=SYSDA
//JSONIN   DD  DATA,DLM=@@
{
  "EVENT"          : "100",
  "USERID"         : "WSER58",
  "APPL"           : "ZSRV",
  "PACKAGE"        : "ZSRV000007",
  "SITE"           : "",
  "RELEASE"        : "",
  "RELEASEAREA"    : "",
  "PROMOTIONNAME" : "D002DEV",
  "PROMOTIONLEVEL": "10",
  "LIBTYPE"        : "JAV",
  "COMPONENT"      : "com/serena/sercmn/zmf/constants/IAccessTypes.java"
}
@@
//SYSIN    DD  *
Server=d001.microfocus.com
Port=09992
Context=zmfevent
Method=POST
/*
//*
//PRETTY   EXEC PGM=IKJEFT01,REGION=0M
//REMOVE    DD DISP=(OLD,DELETE),DSN=WSER58.JSON.TEMPOUT
//SYSEXEC   DD  DISP=SHR,DSN=SYS1.SAMPLIB
//SYSTSPRT  DD DISP=(,PASS),DSN=&&LIST05,
//             UNIT=SYSALLDA,SPACE=(CYL,(1,5),RLSE),
//             DCB=(RECFM=FBA,LRECL=133,BLKSIZE=27930)
//SYSTSIN   DD  *
HWTJSPRT WSER58.JSON.TEMPOUT
```

CMNURIRX (Easy access to http methods from a REXX exec)

This program is a wrapper to the same engine as driven by CMNURIBA (note: this is common module CMNURI00), it does the same things except in a more REXX exec 'friendly' fashion. Originally intended for execution from a ZMF HLL exit written in REXX, but it could be executed from any REXX exec.

Note that the default URN implemented by CMNURIRX is /<context>/event/hllx or /zmfevent/event/hllx if context is not specified, i.e. the REST server as used by HLL exits. The equivalent SYSIN and JSONIN parameters are passed to CMNURIRX via stem variables. Output from the program (like sysprint from the batch version) is also passed

back via a stem variable. It's easiest to see how this works from an example. The following was implemented into an HLLX REXX exec:

```
/*  REXX  */
proceed = 'YES'
inStem   = 'ZMFUriParm'
outStem  = 'ZMFUriMsg'
jsonStem = 'ZMFUriJson'

Say '-------------------------------------------------------'
Say 'HLL exit point FREZ00XM - prior to package freeze service'
Say ' This exit is being called prior to the freeze of'
Say ' package: 'packageId
Say '-------------------------------------------------------'
Say ' '

ZMFUriParm.0 = 4
ZMFUriParm.1   = 'Server=d001.microfocus.com'
ZMFUriParm.2   = 'Port=8085'
ZMFUriParm.3   = 'Trace=NO'
ZMFUriParm.4   = 'Method=POST'
ZMFUriJson.0 = 4
ZMFUriJson.1   = '{'
ZMFUriJson.2   = '  "EVENT"    : "40",'
ZMFUriJson.3   = '  "PACKAGE"  : "'packageId'" '
ZMFUriJson.4   = '}'

Call SYSCALLS 'SIGOFF'

address LINKMVS 'CMNURIRX inStem outStem jsonStem'

If RC = 0 then
  Do
    Say 'Pre-freeze Jenkins pipeline has completed successfully'
  End
Else
  Do
    Say 'Pre-freeze Jenkins pipeline was unsuccessful, messages follow'
    If ZMFUriMsg.0 <> 0 then
      Do i = 1 to ZMFUriMsg.0
        Say ZMFUriMsg.i
      End
    proceed  = "NO"
    shortMsg = "Jenkins process failed"
    longMsg  = "Failure of Jenkins pipeline has caused this freeze to fail."
  End

exit 0
```

Here we have three stem variables, zmfUriParm (for input parameters), zmfUriMsg (for output messages) and zmfUriJson (for input JSON body statements). The input stem variables are populated as if you were supplying sysin and JSON to the batch CMNURIBA utility. The output message stem variable is accessed as you would any stem variable and you can see it being 'SAY'ed in the above REXX.

The rootnames of these stem variables (i.e. without the ending period) are set in the three simple variables inStem, outStem, jsonStem and passed, in that order, as parameters to a LINKMVS call to CMNURIRX.

Index

Z