IBM MQ for z/OS Generate MQSC Commands plugin for IBM Urban Code Deploy

Mayur Raja,
IBM MQ Development,
Hursley Park,
mayur_raja@uk.ibm.com

March 2017

Abstract

This paper describes the IBM MQ for z/OS Generate MQSC Commands plugin for IBM Urban Code Deploy (UCD), and shows how to use it to provision IBM MQ resources represented in files in a Source Control Management (SCM) system, to a target IBM MQ for z/OS queue manager environment.

Knowledge of IBM MQ for z/OS and IBM UCD are assumed.

Acknowledgements

I'd like to extend my thanks to:

- **Staff** at a major European Bank**,**
- **Zhang Hong Chen**, STSM, Urban Code Deploy for System z,
- **Xiao Bin Chen**, Software Engineer, Urban Code Deploy,
- **Nicholas Mathison**, Software Developer, Urban Code Deploy Plugins,
- **Michael Samano,** Development Manager, Urban Code Deploy,
- **Carl Maslen,** System Z Support, IBM Cloud,
- **Pete Siddall**, STSM and IBM MQ for z/OS Architect

for their support and expertise during the development of the plugin, and to:

- **Pete Siddall**, STSM and IBM MQ for z/OS Architect,
- **Sue Robertson**, IBM MQ Information Development Team Lead, and
- **Bill Oppenheimer**, IBM MQ Information Developer

for reviewing the paper and providing feedback and suggestions for improvement.

Background Product Information

IBM Urban Code Deploy

IBM® UrbanCode™ Deploy orchestrates and automates application deployments, middleware configurations, and database changes to on premise or cloud-based development, test, and production environments. Your team can deploy as often as needed—on demand or on schedule, with security-rich, self-service release management. Whether you require on-premises or as-a-service, IBM UrbanCode Deploy helps you accelerate your time to market, drive down deployment costs, reduce risks, and achieve continuous delivery.

For further information please see: http://www-03.ibm.com/software/products/en/ucdep

IBM MQ

IBM® MQ is messaging middleware that simplifies and accelerates the integration of diverse applications and business data across multiple platforms. It uses message queues to facilitate the exchanges of information and offers a single messaging solution for cloud, mobile, Internet of Things (IoT), and on-premises environments.

By connecting virtually everything from a simple pair of applications to the most complex business environments, IBM MQ helps you improve business responsiveness, control costs, reduce risk—and gain real-time insight from mobile, IoT and sensor data.

IBM MQ is available as standalone distributed software, as a physical appliance, and on IBM z/OS®. IBM MQ workloads can also be pushed to the cloud. An enhanced version of IBM MQ—IBM MQ Advanced—is available as distributed software and on z/OS to meet a greater set of integration requirements, including extended encryption, file transfer capabilities and a telemetry option for access to data from sensors and mobile devices.

For further information please see: http://www-03.ibm.com/software/products/en/ibm-mq

Existing IBM UCD plugin for IBM MQ on distributed platforms

The existing IBM UCD plugin for IBM MQ is designed to create IBM MQ resources on distributed platforms and not on the z/OS platform.

For further information, please see:
https://developer.ibm.com/urbancode/plugin/websphere-mq/

New IBM MQ for z/OS Generate MQSC Commands plugin for IBM UCD

The new IBM MQ for z/OS Generate MQSC Commands plugin for IBM UCD allows selected MQSC commands to be generated so that they can be defined on a target IBM MQ for z/OS queue manager to provision IBM MQ resources.

Note: The new plugin does not support the creation of IBM MQ for z/OS queue managers.

For further information, please continue to read this paper.

Introduction

IBM UCD is a great tool for orchestrating and automating application deployments, middleware configurations, and database changes to on premise or cloud-based development, test and production environments. The following diagram illustrates a typical deployment flow:

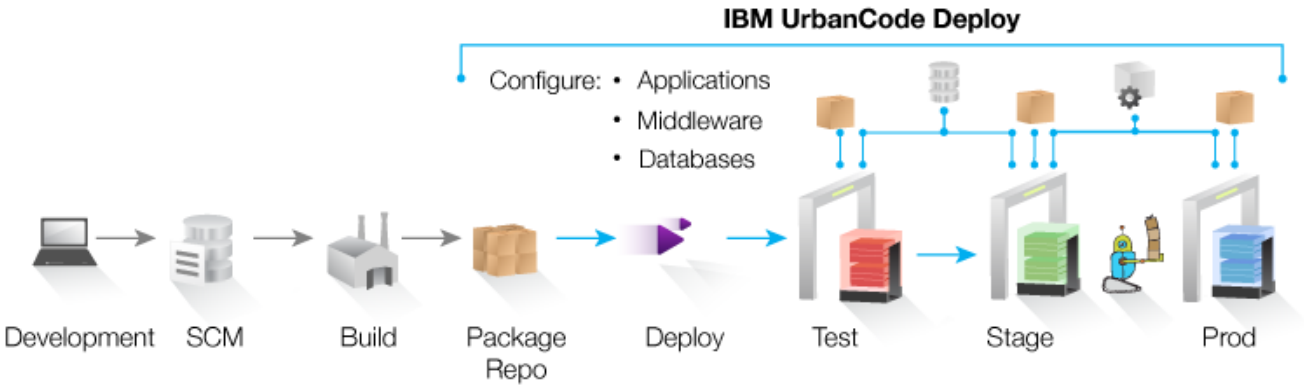


Figure 1: A typical IBM UCD Flow

Essentially, the development or systems administration teams would create a representation of the resources to be defined in files held in a Source Control Management (SCM) system, build and package the resources and deploy them into a target environment management by IBM UCD. The above diagram shows deployment from development onto a Test environment, followed by controlled deployments into Staging and Production environments but there might well be additional deployment target environments in the pipeline. In essence, the idea is to deploy resources into one environment and once the resources have been tested and found to be stable, to progressively deploy them into further environments down the line until reaching Production. IBM UCD provides the ability to set properties on numerous IBM UCD artifacts (for example, resource, application, component, environment, and so on) so that, if required, the characteristics of the resources (for example, names and attribute values) can be rapidly varied as resources are progressively deployed into different target environments. This is generally the recommended approach with IBM UCD and provides a great deal of flexibility.

Some IBM MQ users have however expressed a need to define resource characteristics for each deployment environment in the files held in the SCM system. This is so they can deploy both the application source code (also held in the SCM system) and the associated resource files. The main advantage of this is that resource attribute values are defined in one place and not scattered across numerous IBM UCD artifacts. It also means that all users do not need to gain a deep understanding of IBM UCD artifacts and properties. So, to address this need, the new **IBM MQ for z/OS Generate MQSC Commands plugin** for **IBM UCD** has been created.

IBM MQ for z/OS Generate MQSC Commands plugin for IBM UCD

The **IBM MQ for z/OS Generate MQSC Commands plugin** for **IBM UCD** processes REST form IBM MQ resource representations and generates MQSC form IBM MQ resource definitions.

The REST form IBM MQ resource representations are defined as a triplet of files in a SCM system:

- **base** MQ resource definitions
- **overrides** values per target deployment environment
- **properties** values

Note: The plugin expects the triplet of files to be present. Valid values must be defined in the base and overrides files however, the properties file can be left blank if no properties are in use.

The plugin is designed to be defined as a step in an IBM UCD process. Details of the process are described later in this paper.

Note: If there is a need to frequently change the values of selected resource attributes during runtime, it is advantageous to define properties for these on IBM UCD artifacts as these can be rapidly updated. To update values in the triplet of files and to then deploy them to the runtime environment could take several minutes.

Overall design of the IBM MQ for z/OS Generate MQSC Commands plugin

The IBM MQ for z/OS Generate MQSC Commands plugin is designed to process the triplet of files as follows:

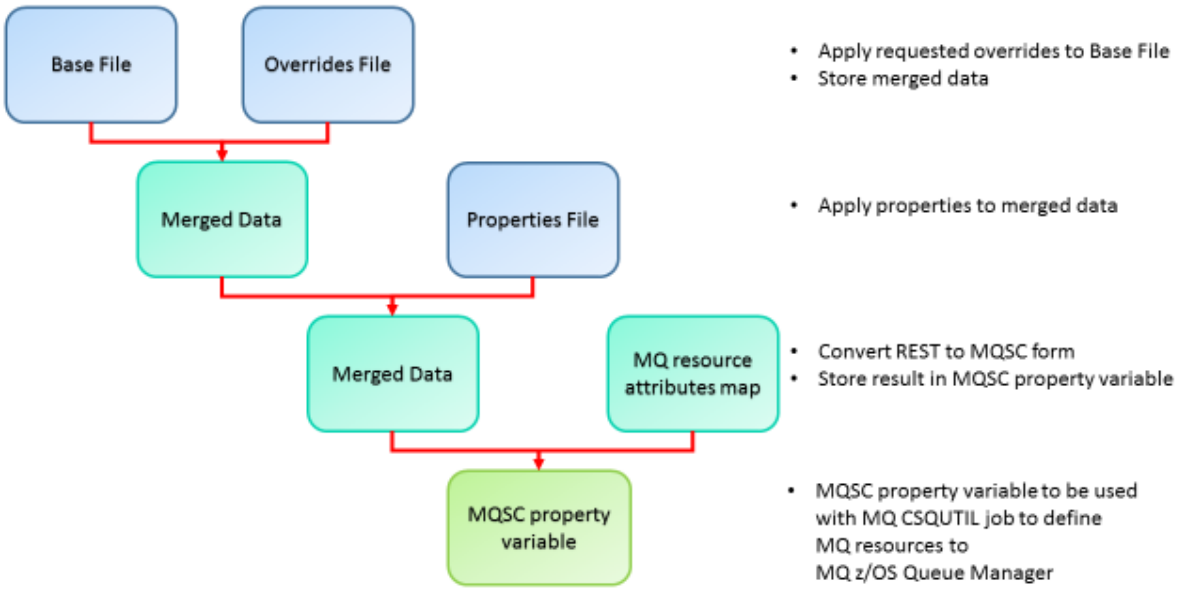


Figure 2: Processing the Triplet of Files

Notes:

1) The **Merged Data** is held in a string variable.
2) The **MQ resource attributes map** is an internal file, defined in **encoding UTF-8,** which is used by the IBM MQ for z/OS Generate MQSC Commands plugin to map rest form attributes to MQSC form attributes.
3) The **MQSC property variable** is an output variable from the plugin that is used in a subsequent step in the IBM UCD process to run the **IBM MQ CSQUTIL job** to define the MQ resources on the MQ queue manager for the target deployment environment.

IBM MQ Applications and Resource Deployment

Before IBM MQ applications can put or get messages from IBM MQ queues, queue resources need to be defined to an IBM MQ for z/OS queue manager. Additionally, before applications can transfer data from one IBM MQ for z/OS queue manager to another, channel resources and further queues need to be defined on the local and/or remote IBM MQ for z/OS queue managers. The exact type and number of resources required will vary, depending on the specific needs of applications. Provided a reasonable separation of resources can be achieved, say by application, resources can be represented in the triplet of files as follows:

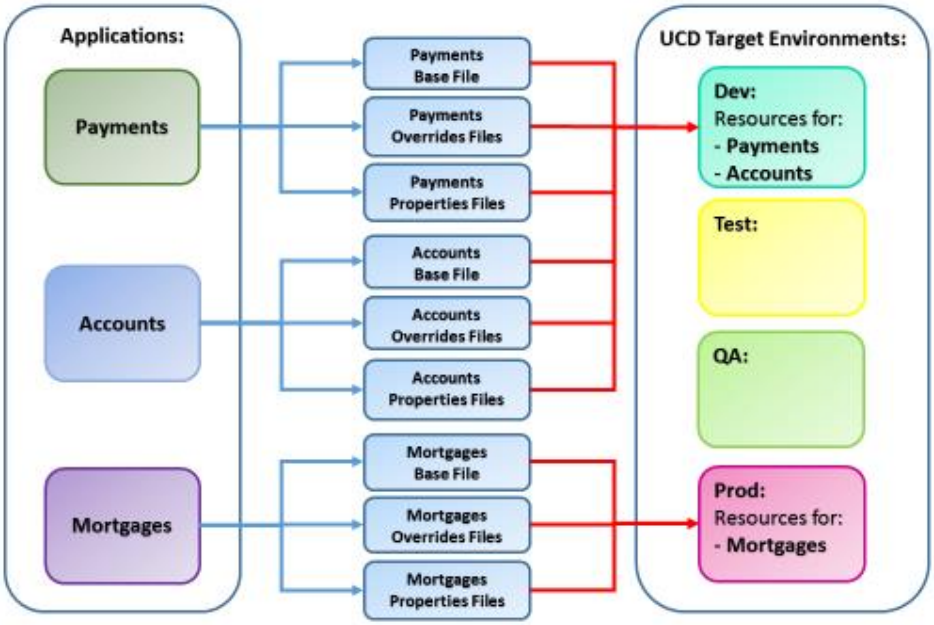


Figure 3: MQ Applications and Resource Deployment

Resources for the **Payments** and **Accounts** applications are deployed to the **Dev** environment while resources for the **Mortgages** application are deployed to the **Prod** environment.

Resources in the triplet of files need to be defined in JSON format in **encoding UTF-8.** Each file contains the following resources:

1. **Base –** the base definition of resource attributes and values. Resource attribute values in this file may contain tokens.
2. **Overrides –** overriding attributes and values for each target deployment environment. Resource attribute values in this file may contain tokens.
3. **Properties –** tokens and values to be replaced in resource attribute values of resources defined in the base and overrides files.

Depending on the target environment being deployed to, the IBM MQ for z/OS Generate MQSC Commands plugin applies the respective override values and generates the appropriate IBM MQ resource definitions. The resources are subsequently defined on the target environment. An example to define a local queue is shown below. A simplified syntax is used for clarity:

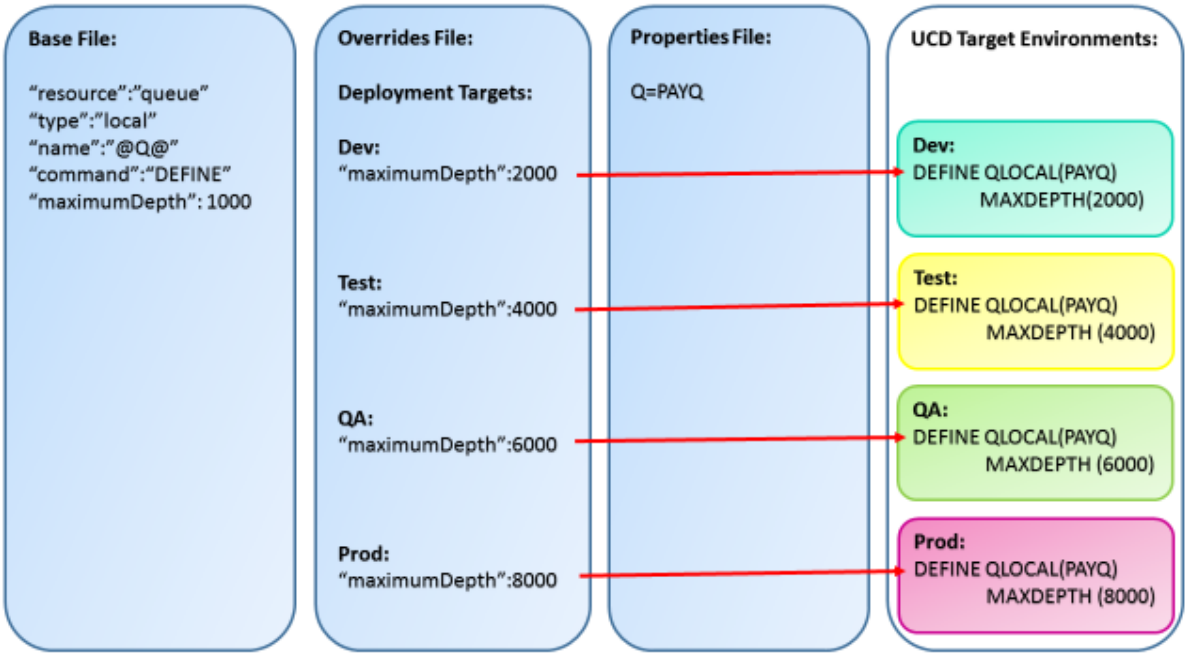


Figure 4: Deployment of resources to target environments

Notice how the **maximum depth** of Queue **PAYQ** varies depending on the target deployment environment.

More detailed examples, with specific syntax, are shown below.

Overall Design for the representation of resources in the triplet of files

As IBM MQ has plans to expose a new REST API to allow users to interact with IBM MQ objects, the representation of resources is based on the IBM MQ REST API implementation.

Queues

The representation of queues is based on the REST API for IBM MQ Queues. Details of this can be found in the IBM MQ V9 Knowledge Center (see: http://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.0.0/com.ibm.mq.adm.doc/q127590_.htm).
IBM MQ vNext beta customers can also obtain the latest beta driver, enable API discovery (see: API Discovery at: http://www.ibm.com/support/knowledgecenter/SSFKSJ_9.0.0/com.ibm.mq.adm.doc/q128290_.htm) and get SWAGGER documentation for the REST API for IBM MQ Queues.

The following table lists the REST attribute groups, the REST attributes and the MQSC attributes they map to, the REST attribute values and the corresponding MQSC attribute values, for queues. This information is required when defining queue resource representations in the base and overrides files (see examples below):

REST Attribute Group Name	REST Attribute Name	MQSC Attribute Name	REST Attribute Value	MQSC Attribute Value
optionalParms	commandScope	CMDSCOPE	*String*	*string*
	like	LIKE	*String*	*string*
	noReplace	NOREPLACE	noReplace	NOREPLACE
	force	FORCE	Force	FORCE
	keepAuthorityRecords	AUTHOREC	keepAuthorityRecords	NO
	purge	PURGE	Purge	PURGE
	queueSharingGroupDisposition	QSGDISP	Copy	COPY
			Group	GROUP
			Private	PRIVATE
			Qmgr	QMGR
			shared	SHARED
alias	targetName	TARGET	*string*	*string*
	targetType	TARGTYPE	queue	QUEUE
			topic	TOPIC
applicationDefaults	clusterBind	DEFBIND	notFixed	NOTFIXED
			onOpen	OPEN
			onGroup	GROUP
	messagePropertyControl	PROPCTL	all	ALL
			compatible	COMPAT
			force	FORCE
			none	NONE
			version6Compatible	V6COMPAT
	messagePersistence	DEFPSIST	nonPersistent	NO
			persistent	YES
	messagePriority	DEFPRTY	*int*	*int*
	putResponse	DEFPRESP	asynchronous	ASYNC
			synchronous	SYNC
	readAhead	DEFREADA	disabled	DISABLED
			no	NO
			yes	YES
	sharedInput	NOSHARE	false	NOSHARE
		SHARE	true	SHARE
cluster	transmissionQueueForChannelName	CLCHNAME	*string*	*string*
	name	CLUSTER	*string*	*string*
	namelist	CLUSNL	*string*	*string*
	workloadPriority	CLWLPRTY	*int*	*int*

	workloadQueueUse	CLWLUSEQ	any	ANY
			asQmgr	QMGR
			local	LOCAL
	workloadRank	CLWLRANK	*int*	*int*
dataCollection	accounting	ACCTQ	asQmgr	QMGR
			off	OFF
			on	ON
	monitoring	MONQ	asQmgr	QMGR
			high	HIGH
			low	LOW
			medium	MEDIUM
			off	OFF
	statistics	STATQ	asQmgr	QMGR
			off	OFF
			on	ON
events.depth	highEnabled	QDPHIEV	false	DISABLED
			true	ENABLED
	highPercentage	QDEPTHHI	*int*	*int*
	lowEnabled	QDPLOEV	false	DISABLED
			true	ENABLED
	lowPercentage	QDEPTHLO	*int*	*int*
	fullEnabled	QDPMAXEV	false	DISABLED
			true	ENABLED
events.serviceInterval	duration	QSVCINT	*int*	*int*
	highEnabled	QSVCIEV	false	NONE
			true	HIGH
	okEnabled	QSVCIEV	false	NONE
			true	OK
extended	allowSharedInput	DEFSOPT	false	EXCL
			true	SHARED
	backoutRequeueQueueName	BOQNAME	*string*	*string*
	backoutThreshold	BOTHRESH	*int*	*int*
	custom	CUSTOM	*string*	*string*
	hardenGetBackout	NOHARDENBO	false	NOHARDENBO
		HARDENBO	true	HARDENBO
	supportDistributionLists	DISTL	false	NO
			true	YES
general	description	DESCR	*string*	*string*
	inhibitGet	GET	false	DISABLED
			true	ENABLED
	inhibitPut	PUT	false	DISABLED
			true	ENABLED
	isTransmissionQueue	USAGE	false	NORMAL
			true	XMITQ
model	type	DEFTYPE	permanentDynamic	PERMDYN
			sharedDynamic	SHRDDYN
			temporaryDynamic	TEMPDYN
queueSharingGroup	disposition	QSGDISP	copy	COPY
			group	GROUP
			private	PRIVATE
			qmgr	QMGR
			shared	SHARED
	structureName	CFSTRUCT	*string*	*string*
remote	queueName	RNAME	*string*	*string*
	qmgrName	RQMNAME	*string*	*string*
	transmissionQueueName	XMITQ	*string*	*string*
storage	indexType	INDXTYPE	correlationId	CORRELID
			groupId	GROUPID
			messageId	MSGID
			messageToken	MSGTOKEN
			none	NONE
	maximumDepth	MAXDEPTH	*int*	*int*
	maximumMessageLength	MAXMSGL	*int*	*int*
	messageDeliverySequence	MSGDLVSQ	fifo	FIFO
			priority	PRIORITY
	nonPersistentMessageClass	NPMCLASS	normal	NORMAL
			high	HIGH
	storageClass	STGCLASS	*string*	*string*
trigger	enabled	TRIGGER	false	NOTRIGGER
			true	TRIGGER

	data	TRIGDATA	*string*	*string*
	depth	TRIGDPTH	*int*	*int*
	initiationQueueName	INITQ	*string*	*string*
	messagePriority	TRIGMPRI	*int*	*int*
	processName	PROCESS	*string*	*string*
	type	TRIGTYPE	depth	DEPTH
			every	EVERY
			first	FIRST
			none	NONE

Table 1: Attribute groups, attributes and attribute values for Queues

Channels

The representation of channels is based on an initial version of the IBM MQ REST API for channels. However, as this is still under design/development and hence is **subject to change**, full details are not yet available. However, the following table lists the REST attribute groups, REST attributes and the MQSC attributes they map to, the REST attribute values and the corresponding MQSC attribute values, for channels. This information is required when defining channel resource representations in the base and overrides files (see examples below):

REST Attribute Group Name	**REST Attribute Name**	**MQSC Attribute Name**	**REST Attribute Value**	**MQSC Attribute Value**
optionalParms	commandScope	CMDSCOPE	*string*	*string*
	like	LIKE	*string*	*string*
	noReplace	NOREPLACE	noReplace	NOREPLACE
	queueSharingGroupDisposition	QSGDISP	copy	COPY
			group	GROUP
			qmgr	QMGR
amqp	keepAliveInterval	AMQPKA	*int*	*int*
	port	PORT	*int*	*int*
	topicRoot	TPROOT	*string*	*string*
	useClientID	USECLTID	false	NO
			true	YES
batch	dataLimit	BATCHLIM	*int*	*int*
	heartbeat	BATCHHB	*int*	*int*
	interval	BATCHINT	*int*	*int*
	size	BATCHSZ	*int*	*int*
clientConnection	affinity	AFFINITY	false	NONE
			true	PREFERRED
	defaultReconnect	DEFRECON	disabled	DISABLED
			no	NO
			qmgr	QMGR
			yes	YES
	qmgrName	QMNAME	*string*	*string*
	weight	CLNTWGHT	*int*	*int*
cluster	name	CLUSTER	*string*	*string*
	namelist	CLUSNL	*string*	*string*
	workloadPriority	CLWLPRTY	*int*	*int*
	workloadWeight	CLWLWGHT	*int*	*int*
	networkPriority	NETPRTY	*int*	*int*
compression	header	COMPHDR	none	NONE
			system	SYSTEM
	message	COMPMSG	any	ANY
			none	NONE
			rle	RLE
			zlipFast	ZLIBFAST
			zlibHigh	ZLIBHIGH
dataCollection	monitoring	MONQ	asQmgr	QMGR
			high	HIGH
			low	LOW
			medium	MEDIUM
			off	OFF
	statistics	STATQ	asQmgr	QMGR
			high	HIGH

			low	LOW
			medium	MEDIUM
			off	OFF
exits	message	MSGEXIT	*string*	*string*
	messageUserData	MSGDATA	*string*	*string*
	receive	RCVEXIT	*string*	*string*
	receiverUserData	RCVDATA	*string*	*string*
	security	SECEXIT	*string*	*string*
	securityUserData	SECDATA	*string*	*string*
	send	SENDEXIT	*string*	*string*
	sendUserData	SENDDATA	*string*	*string*
extended	senderDataConversion	CONVERT	false	NO
			true	YES
	disconnectInterval	DISCINT	*int*	*int*
	maximumInstances	MAXINST	*int*	*int*
	maximumInstancesPerClient	MAXINSTC	*int*	*int*
	nonPersistentMessageSpeed	NPMSPEED	fast	FAST
			normal	NORMAL
	messagePropertyControl	PROPCTL	all	ALL
			compatible	COMPAT
			none	NONE
	sequenceNumberWrap	SEQWRAP	*int*	*int*
	useDeadLetterQueue	USEDLQ	false	NO
			true	YES
general	description	DESCR	*string*	*string*
	heartbeatInterval	HBINT	*int*	*int*
	keepAliveInterval	KAINT	*int*	*int*
	localAddress	LOCALADDR	*string*	*string*
	maximumMessageLength	MAXMSGL	*int*	*int*
	sharedConversations	SHARECNV	*int*	*int*
	transmissionQueueName	XMITQ	*string*	*string*
mca	type	MCATYPE	process	PROCESS
			thread	THREAD
	userID	MCAUSER	*string*	*string*
messageRetry	count	MRRTY	*int*	*int*
	interval	MRTMR	*int*	*int*
	exit	MREXIT	*string*	*string*
	userData	MRDATA	*string*	*string*
networkConnection	connectionName	CONNAME	*string*	*string*
queueSharingGroup	disposition	QSGDISP	copy	COPY
			group	GROUP
			private	PRIVATE
			qmgr	QMGR
	defaultChannelDisposition	DEFCDISP	fixShared	FIXSHARED
			private	PRIVATE
			shared	SHARED
retry	longCount	LONGRTY	*int*	*int*
	longInterval	LONGTMR	*int*	*int*
	shortCount	SHORTRTY	*Int*	*int*
	shortInterval	SHORTTMR	*Int*	*int*
security	certificateLabel	CERTLABL	*String*	*string*
	cipherSpecification	SSLCIPH	*String*	*string*
	clientAuthentication	SSLCAUTH	optional	OPTIONAL
			required	REQUIRED
	peerName	SSLPEER	*String*	*string*
	putAuthority	PUTAUT	alternateOrMCA	ALTMCA
			Context	CTX
			Default	DEF
			onlyMCA	ONLYMCA

Table 2: Attribute groups, attributes and attribute values for Channels

Note: Only channels with a transmission protocol type of **tcp** are supported so the **TRPTYPE** on MQSC **DEFINE CHANNEL** is automatically set to **TRPTYPE(TCP)**.

Structure of data in the base file

The data starts with the **“resource”** field. Resources in the file can be of **two types**: **“queue”** and **“channel”**.

With the IBM MQ REST API, the IBM MQ command to be issued is defined by the REST API call being issued. In the base file, the command to be issued is specified by the “command” field. Valid commands are:

- “**DEFINE**”
- “**DELETE**”

The name of the resource being created is specified by the **“name”** field.

The type of the resource being created is specified by the **“type”** field.

For queues, valid types are:

- **“alias”**
- **“remote”**
- **“local”**
- **”model”**

For channels, valid type are:

- **“amqp”**
- **"clientConnection"**
- **"clusterReceiver"**
- **"clusterSender"**
- **"receiver"**
- **"requester"**
- **"sender"**
- **"server"**
- **"serverConnection"**

With the IBM MQ REST API, the **optional query parameters** are specified on the URL for the REST API call being issued. In the base file, the optional query parameters are specified in the **“optionalParms”** group. Other parameters are specified in their respective groups as defined in the above tables.

A **skeleton structure** for the base file is:

```
{
    "resource": {
        "queue": [{
            "command": "DEFINE",
            "name": "<queue name>",
            "type": "<queue type>",
            "optionalParms": {
                "<attribute>": "<value>",
                "<attribute>": <value>
```



```
            },
            "<queue type group>": {
                "<attribute>": "<value>",
                "<attribute>": <value>
            },
            "<other attribute group>": {
                "<attribute>": "<value>",
                "<attribute>": <value>
            }
        }],
        "channel": [{
            "command": "DEFINE",
            "name": "<channel name>",
            "type": "<channel type>",
            "optionalParms": {
                "<attribute>": "<value>",
                "<attribute>": <value>
            },
            "<other attribute group>": {
                "<attribute>": "<value>",
                "<attribute>": <value>
            }
        }]
    }
}
```

Figure 5: Base file skeleton structure

Where fields defined within meta-brackets need to be set as per the values specified in this section and in the tables above.

Note: string attribute values are specified within double quotes while **boolean** and **integer** values are not.

So, for example, a local queue and a receiver channel can be represented as:

```
{
    "resource": {
        "queue": [{
            "command": "DEFINE",
            "name": "PAYMENT.QUEUE",
            "type": "local",
            "general": {
                "description": "Queue for bank payments"
            },
            "storage": {
                "maximumDepth": 1000,
                "maximumMessageLength": 32000
            },
            "extended": {
                "allowSharedInput": true
            },
            "trigger": {
                "enabled": false,
                "type": "none"
            }
        }],
        "channel": [{
            "command": "DEFINE",
```



```
            "name": "DRQM.TO.DLQM",
            "type": "receiver"
        }]
    }
}
```

Figure 6: Base file example structure

Structure of data in the overrides file

The data in the overrides file has a very similar structure to the data in the properties file. The reason for this is that the overrides file allows you to specify overriding values, by the target deployment environment, for every resource represented in the base file.

So, the **"resource"**, resource types ("**queue"** and **"channel"**), **"command"**, **"name"** and **"type"** fields are exactly as described above. It is important to ensure that the values of these fields match the values of the same fields specified in the base file because they are used when determining the overrides to apply.

The overrides for each target deployment environment are specified in the **"deploymentTargets"** group. Any number of target deployment environments can be specified as comma separated maps and you can decide on the deployment environment names that are suitable for your environment. So, for example, you can specify different override values for **"Dev"**, **"Test"**, **"QA"** and **"Prod"** environments.

For each target deployment environment, the target environment name is specified in the **"targetEnvName"** field.

For each target deployment environment, the **"deploy"** field indicates whether the corresponding resource defined in the base file, along with an override values, are to be deployed or not. If set to **true**, the resource with the override values applied is deployed and, if set to **false**, the resource and any corresponding override values are not deployed.

The **"overrideName"** group is an optional group that can be used to specify an overriding value for the name of the resource in the **"name"** field. This is discussed in more detail in section '**Overriding the name of a resource**' below.

The other values to be overridden are specified in their respective groups as defined in the above tables.

A **skeleton structure** for the overrides file is:

```
{
	"resource": {
		"queue": [{
			"command": "DEFINE",
			"name": "<queue name>",
			"type": "<queue type",
			"deploymentTargets": [{
				"targetEnvName": "<environment name>",
				"deploy": <true/false>,
				"overrideName": {
					"name": "<overriding queue name>"
```



```
                        },
                        "<attribute group>": {
                                "<attribute>": "<value>",
                                "<attribute>": <value>
                        }
                },
                {
                        "targetEnvName": "<environment name>",
                        "deploy": <true/false>,
                        "<attribute group>": {
                                "<attribute>": "<value>",
                                "<attribute>": <value>
                        }
                }]
        }],
        "channel": [{
                "command": "DEFINE",
                "name": "<channel name>",
                "type": "<channel type>",
                "deploymentTargets": [{
                        "targetEnvName": "<environment name>",
                        "deploy": <true>/false>,
                        "attribute group": {
                                "<attribute>": "<value>",
                                "<attribute>": <value>
                        }
                },
                {
                        "targetEnvName": "<environment name>",
                        "deploy": <true/false>,
                        "<attribute group>": {
                                "<attribute>": "<value>",
                                "<attribute>": <value>
                        }
                }]
        }]
    }
}
```

Figure 7: Overrides file skeleton structure

Where fields defined within meta-brackets need to be set as per the values specified in this section and in the tables above.

Note: string attribute values are specified within double quotes while **boolean** and **integer** values are not.

So, for example, override values for a local queue and receiver channel can be represented as:

```
{
    "resource": {
        "queue": [{
            "command": "DEFINE",
            "name": "PAYMENT.QUEUE",
            "type": "local",
            "deploymentTargets": [{
                "targetEnvName": "Dev",
```

```
                "deploy": true,
                "storage": {
                    "maximumDepth": 10000,
                    "maximumMessageLength": 16384
                }
            },
            {
                "targetEnvName": "Test",
                "deploy": true,
                "storage": {
                    "maximumDepth": 20000,
                    "maximumMessageLength": 32768
                }
            },
            {
                "targetEnvName": "QA",
                "deploy": false,
                "storage": {
                    "maximumDepth": 30000,
                    "maximumMessageLength": 65536
                }
            },
            {
                "targetEnvName": "PROD",
                "deploy": false
            }]
        }],
        "channel": [{
            "command": "DEFINE",
            "name": "DRQM.TO.DLQM",
            "type": "receiver",
            "deploymentTargets": [{
                "targetEnvName": "Dev",
                "deploy": true,
                "extended": {
                    "useDeadLetterQueue": true
                }
            },
            {
                "targetEnvName": "Test",
                "deploy": true,
                "general": {
                    "description": "Receiver Channel for Test Environment",
                    "maximumMessageLength": "32768"
                }
            }]
        }]
    }
}
```

Figure 8: Overrides file example structure

So, if the triplet of files are being deployed to target environment:

a) **Dev**, for local queue **PAYMENT.QUEUE**, maximum depth is set to 10,000 and maximum message length is set to 16384. And, for receiver channel **DRQM.TO.DLQM**, place undelivered messages to the dead letter queue is enabled. Notice that deploy is set to true in both cases.

b) **Test**, for local queue **PAYMENT.QUEUE**, maximum depth is set to 20,000 and maximum message length is set to 65536. And, for receiver channel **DRQM.TO.DLQM**, the description is set to 'Receiver Channel for Test Environment' and the maximum message length that the channel can receive is set to 32768. Notice that deploy is set to true in both cases.
c) **QA**, for local queue **PAYMENT.QUEUE**, the specified overrides are not applied since deploy is set to false. And, for receiver channel **DRQM.TO.DLQM**, no overrides are applied since none are specified.
d) **Prod,** for local queue **PAYMENT.QUEUE** and for receiver channel **DRQM.TO.DLQM,** no overrides are applied since none are specified.

If the overrides file is not defined or empty, a **NullPointerException** is raised.

Structure of data in the properties file

The properties file essentially contains token and value pairs.

A **skeleton structure** for the properties file is:

```
{
    "<property name>":"<property value>",
    "<property name>":"<property value>"
}
```

Figure 9: Properties file skeleton structure

Where properties and property values defined within meta-brackets can be set to any token and value that you want to use in your resource names.

So for example, you can define the following properties if you want to define token value pairs for your local and remote IBM MQ for z/OS queue manager names:

```
{
    "LQMGR":"DLQM",
    "RQMGR":"DRQM"
}
```

Figure 10: Properties file example structure

Tokens can be specified in resource definitions in the base and overrides files by wrapping the tokens within the 'at ('@')' symbol. For example, you can specify the above two tokens in the channel name as:

```
{
    "resource": {
        "channel": [{
            "command": "DEFINE",
            "name": "@RQMGR@.TO.@LQMGR@",
            "type": "receiver"
        }]
    }
}
```

Figure 11: Example structure showing use of tokens in a channel name

Note: If tokens are used in names, tokens used in the names of corresponding resources in the base and overrides files must be identical. The reason for this is that the names are matched when determining the override values to be applied.

When the files are processed by the IBM MQ for z/OS Generate MQSC commands plugin, tokens in the channel name are automatically resolved to form channel name **DRQM.TO.DLQM.**

The plugin design **expects the properties file** to exist. However, if no token value pairs are included in the resource definitions, the **properties file** can be left **completely blank** or set to an **empty map** as follows:

```
{

}
```

If tokens are included in resource definitions but the corresponding token value pairs are not specified in the properties file, an **IllegalArgumentException** is thrown because there is no way to resolve the tokens. Also, in this case, the IBM MQ CSQUTIL job is not submitted because an attempt to define such resources will clearly fail.

Overriding the name of a resource

If you want to define resources so that they have different names in different target environments, there are a couple of ways to achieve this.

Using the "overrideName" group

In the **base file,** you can, for example, define an **alias** queue:

```
{
    "resource": {
        "queue": [{
            "command": "DEFINE",
            "name": "PAYMENTQ.ALIAS",
            "type": "alias",
            "alias": {
                "targetName": "PAYMENTQ.BASE",
                "targetType": "queue"
            }
        }]
    }
}
```

Figure 12: Example structure showing an alias queue definition in the base file

In the **overrides file**, you can define **overrides** for the **alias** queue:

```
{
    "resource": {
        "queue": [{
            "command": "DEFINE",
            "name": "PAYMENTQ.ALIAS",
            "type": "alias",
            "deploymentTargets": [{
```



```
                "targetEnvName": "Dev",
                "deploy": true,
                "overrideName": {
                    "name": "DVQM.PAYMENTQ.ALIAS"
                },
                "alias": {
                    "targetName": "DVQM.PAYMENTQ.BASE",
                    "targetType": "queue"
                }
            },
            {
                "targetEnvName": "Test",
                "deploy": true,
                "overrideName": {
                    "name": "TSQM.PAYMENTQ.ALIAS"
                },
                "alias": {
                    "targetName": "TSQM.PAYMENTQ.BASE",
                    "targetType": "queue"
                }
            },
            {
                "targetEnvName": "QA",
                "deploy": true,
                "overrideName": {
                    "name": "QAQM.PAYMENTQ.ALIAS"
                },
                "alias": {
                    "targetName": "QAQM.PAYMENTQ.BASE",
                    "targetType": "queue"
                }
            },
            {
                "targetEnvName": "Prod",
                "deploy": true,
                "overrideName": {
                    "name": "PRQM.PAYMENTQ.ALIAS"
                },
                "alias": {
                    "targetName": "PRQM.PAYMENTQ.BASE",
                    "targetType": "queue"
                }
            }]
        }]
    }
}
```

Figure 13: Example structure showing override names for an alias queue

In the **properties file**, you can define an empty set (since no properties are in use):

```
{
}
```

Figure 14: Example empty structure in the properties file showing no tokens and values are in use

Depending on target environment being deployed to, this would result in the following named alias queues being defined:

Environment	Alias Queue Name	Base Queue Name
Dev	DVQM.PAYMENTQ.ALIAS	DVQM.PAYMENTQ.BASE
Test	TSQM.PAYMENTQ.ALIAS	TSQM.PAYMENTQ.BASE
QA	QAQM.PAYMENTQ.ALIAS	QAQM.PAYMENTQ.BASE
Prod	PRQM.PAYMENTQ.ALIAS	PRQM.PAYMENTQ.BASE

Table 3: Alias and base queue names

So, if the target deployment environment is **Test**, the following MQSC command is generated:

```
DEFINE QALIAS('TSQM.PAYMENTQ.ALIAS') +
       TARGET('TSQM.PAYMENTQ.BASE') +
       TARGTYPE(QUEUE) +
       REPLACE
```

Using the "overrideName" group, and tokens and values

In the **base file,** you can define:

```
{
    "resource": {
        "queue": [{
            "command": "DEFINE",
            "name": "PAYMENTQ.ALIAS",
            "type": "alias",
            "alias": {
                "targetName": "PAYMENTQ.BASE",
                "targetType": "queue"
            }
        }]
    }
}
```

Figure 15: Example structure showing alias queue definition in the base file

In the **overrides file**, you can define:

```
{
    "resource": {
        "queue": [{
            "command": "DEFINE",
            "name": "PAYMENTQ.ALIAS",
            "type": "alias",
            "deploymentTargets": [{
                "targetEnvName": "Dev",
                "deploy": true,
                "overrideName": {
                    "name": "@DevEnv@.@DevQMgr@.PAYMENTQ.ALIAS"
```



```
                },
                "alias": {
                    "targetName": "@DevEnv@.@DevQMgr@.PAYMENTQ.BASE",
                    "targetType": "queue"
                }
            },
            {
                "targetEnvName": "Test",
                "deploy": true,
                "overrideName": {
                    "name": "@TestEnv@.@TestQMgr@.PAYMENTQ.ALIAS"
                },
                "alias": {
                    "targetName": "@TestEnv@.@TestQMgr@.PAYMENTQ.BASE",
                    "targetType": "queue"
                }
            },
            {
                "targetEnvName": "QA",
                "deploy": true,
                "overrideName": {
                    "name": "@QAEnv@.@QAQMgr@.PAYMENTQ.ALIAS"
                },
                "alias": {
                    "targetName": "@QAEnv@.@QAQMgr@.PAYMENTQ.BASE",
                    "targetType": "queue"
                }
            },
            {
                "targetEnvName": "Prod",
                "deploy": true,
                "overrideName": {
                    "name": "@ProdEnv@.@ProdQMgr@.PAYMENTQ.ALIAS"
                },
                "alias": {
                    "targetName": "@ProdEnv@.@ProdQMgr@.PAYMENTQ.BASE",
                    "targetType": "queue"
                }
            }]
        }]
    }
}
```

Figure 16: Example structure showing override names, with tokens, for an alias queue

In the **properties file**, you can define:

```
{
    "DevEnv":"Dev",
    "DevQMgr":"DVQM",
    "TestEnv":"Test",
    "TestQMgr":"TSQM",
    "QAEnv":"QA",
    "QAQMgr":"QAQM",
    "ProdEnv":"Prod",
    "ProdQMgr":"PRQM"
}
```

Figure 17: Example structure showing token and value pairs to be used when resolving names for an alias queue

Depending on target environment being deployed to, this would result in the following named alias, and base, queues being defined:

Environment	Alias Queue Name	Base Queue Name
Dev	Dev.DVQM.PAYMENTQ.ALIAS	Dev.DVQM.PAYMENTQ.BASE
Test	Test.TSQM.PAYMENTQ.ALIAS	Test.TSQM.PAYMENTQ.BASE
QA	QA.QAQM.PAYMENTQ.ALIAS	QA.QAQM.PAYMENTQ.BASE
Prod	Prod.PRQM.PAYMENTQ.ALIAS	Prod.PRQM.PAYMENTQ.BASE

Table 4: Alias and base queue names (with token and value replacement)

So, if the target deployment environment is **Test**, the following MQSC command is generated:

```
DEFINE QALIAS('Test.TSQM.PAYMENTQ.ALIAS') +
       TARGET('Test.TSQM.PAYMENTQ.BASE') +
       TARGTYPE(QUEUE) +
       REPLACE
```

Security

Before you can proceed to install the IBM MQ for z/OS Generate MQSC Commands plugin you need to login to the IBM UCD WUI using the security credentials defined to you.

Once you have logged in, you can follow the steps described in the following section to install the plugin.

To make use of the IBM MQ for z/OS Generate MQSC Commands plugin you need to add it as a step in an IBM UCD process (described below). This means that the User ID you use to login into IBM UCD with must have sufficient authority to create an IBM UCD process (for further details of Security with IBM UCD, see: http://www.ibm.com/support/knowledgecenter/SS4GSP_6.2.0/com.ibm.udeploy.admin.doc/topics/security_ch.html).

The generated MQSC commands are issued to IBM MQ using a Job Submit step (described below) to submit an IBM MQ CSQUTIL job. This job runs with the authority of the user (for example, this can be your User ID) defined in the Job Step. The Submit Job step also requires that variables **jes.host**, **jes.user**, **jes.monitor.port** and **jes.password** are set as environment properties on an IBM UCD component, for example (see further details of this below).

Installing the IBM MQ for z/OS Generate MQSC Commands plugin

The IBM MQ for z/OS Generate MQSC Commands plugin is shipped as a zip file. The zip file can be installed from the IBM UCD Web User Interface:

Home->Settings->Automation Plugins:

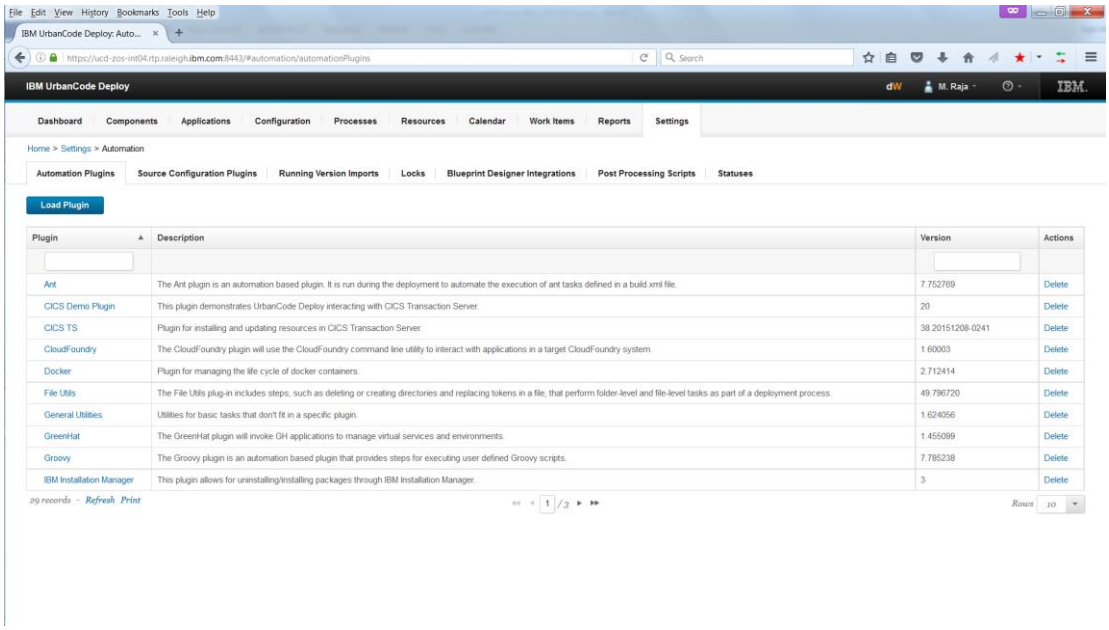


Figure 18: Loading the plugin – 1

Select **Load Plugin** followed by **Browse.** Enter the location of the zip file followed by **Submit**:

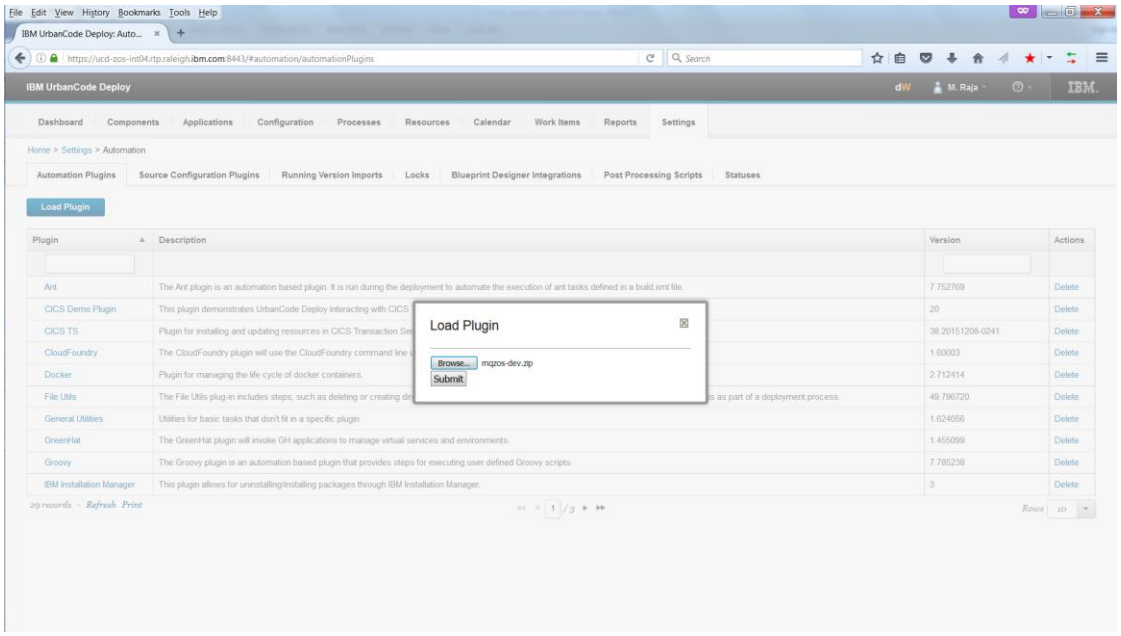


Figure 19: Loading the plugin – 2

When the plugin has loaded, it will appear in the list as **MQ for z/OS**:

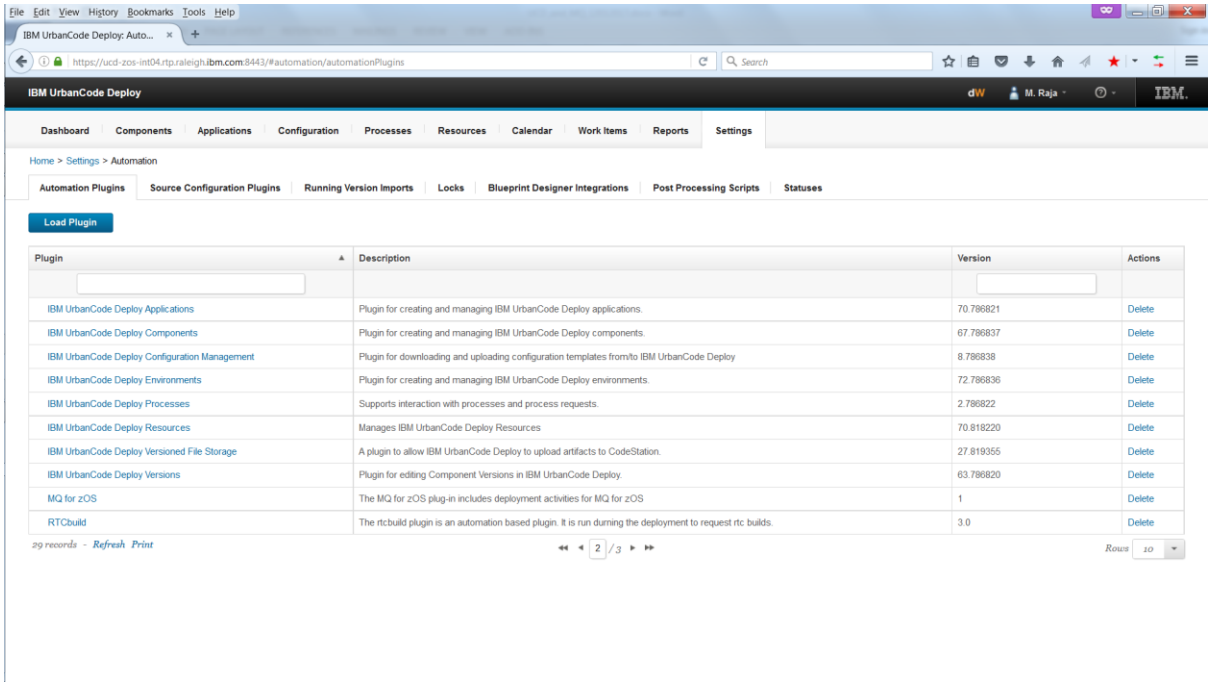


Figure 20: Plugin loaded

Using the IBM MQ for z/OS Generate MQSC Commands plugin

To use the IBM MQ for z/OS Generate MQSC Commands plugin, define a step for it in an IBM UCD process and associate it with a component of an application. Then create deployment targets that map to IBM MQ for z/OS queue managers, to run the application on.

For the purposes of testing:

- An application called Bank Payments was created in IBM UCD.
- A single component, to define IBM MQ resources, was created for the application. The properties of the SCM system were defined to the component.
- Target deployment environments Dev, Test, QA and Prod were created to deploy the application to.
- Two IBM UCD processes were created.

  A process associated with the application
  To define a step to install the application.

  A process associated with the component
  To define the steps required to process the triplet of files and issue MQSC commands to define the IBM MQ resources to the target deployment environment (and hence the target IBM MQ for z/OS queue manager).

This paper focuses on the latter of the two processes. It also does not go into the details of the other IBM UCD configuration listed above. Details of this can be found in the IBM UCD

Knowledge Center (see: http://www.ibm.com/support/knowledgecenter/en/SS4GSP/ucd_welcome.html)

Defining and packaging the triplet of files for version import into UCD

Before the IBM MQ for z/OS Generate MQSC Commands plugin can be used, resources need to be represented in the triplet of files (as described above) and the files need to be packaged and made available for import, as a component version, into IBM UCD.

You can use either **Standard** or **z/OS** components to import and deploy the triplet of files.

A **Standard** component supports version import from various sources, for example, git, TFS, RTC etc. It supports deployment of HFS files.

A **zOS** component imports versions from an LPAR or from an RTC Exterprise Extensions package. It supports deployment of HFS files and data sets, batch processing artifacts based on types or attributes, and rollback of versions.

Refer to the Deploying to the z/OS platform topic in the IBM UCD Knowledge Center to learn more about zOS component.

For the purposes of testing, RTC was used as the SCM system. A triplet of files were created in a stream in RTC and the stream was built into a **snapshot** and imported as a component version into **Codestation**, the IBM UCD repository.

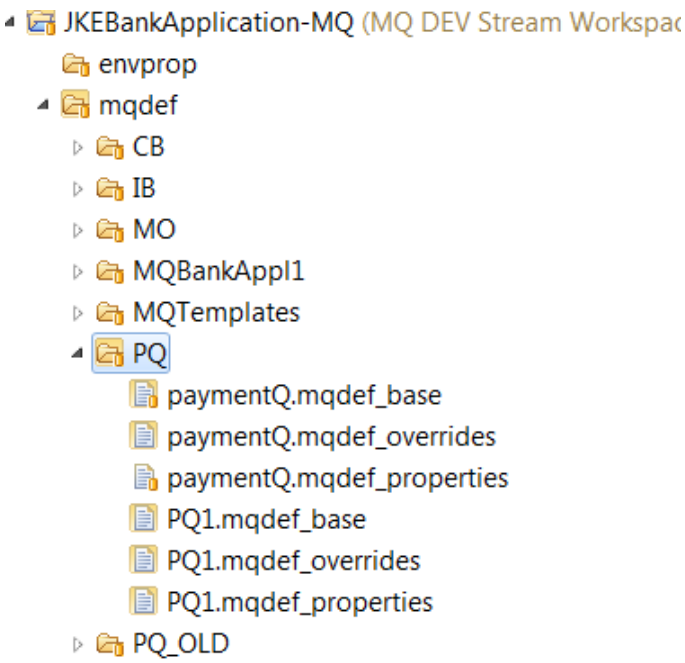


Figure 21: Triplets of files in RTC

Building a snapshot in RTC

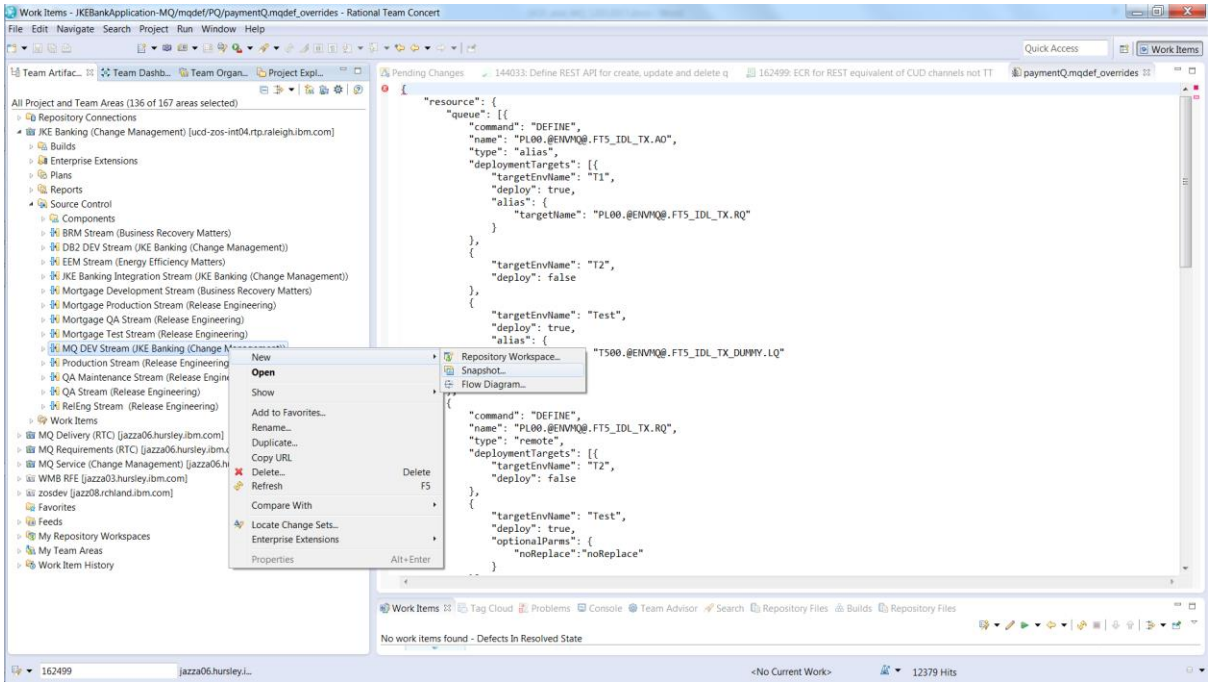


Figure 22: Building a snapshot

Defining the IBM UCD Process to process the triplet of files and issue MQSC commands

The process can be defined as follows:

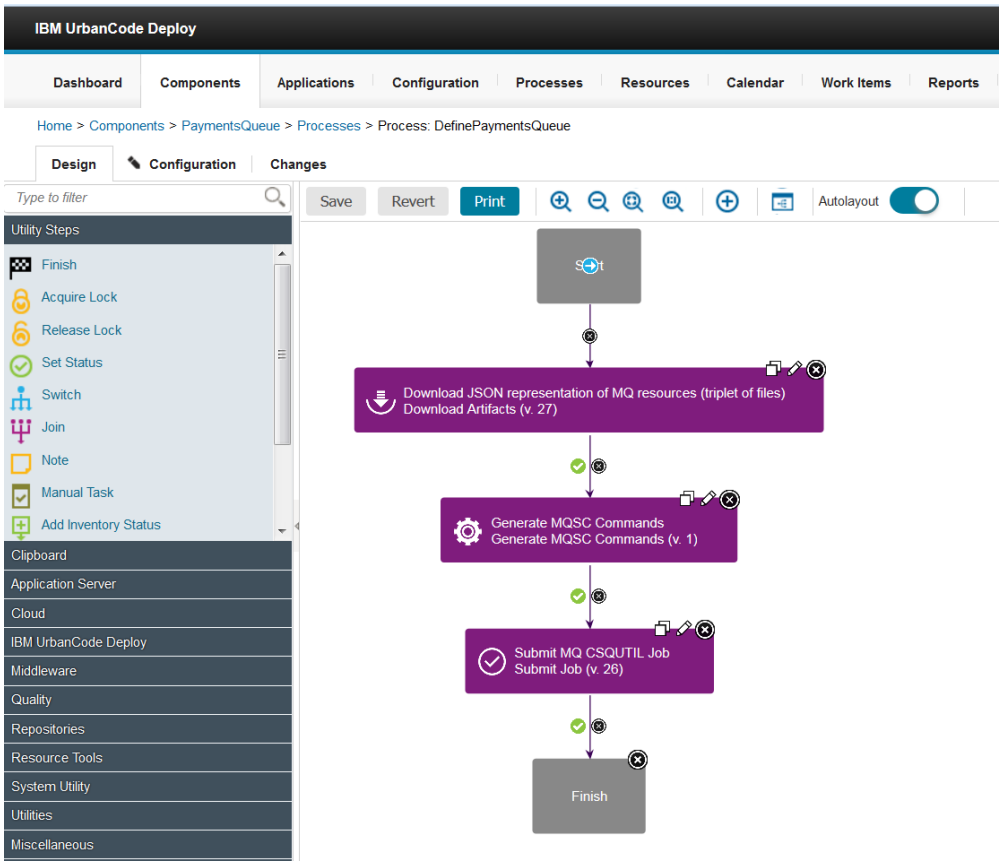


Figure 23: Defining the IBM UCD Process to include the IBM MQ for z/OS Generate MQSC Commands plugin

With steps:

1) Download JSON representation of IBM MQ resources (triplet of files)

Download the triplet of files from the IBM UCD repository.

*The **Download Artifacts** step is used for this:*

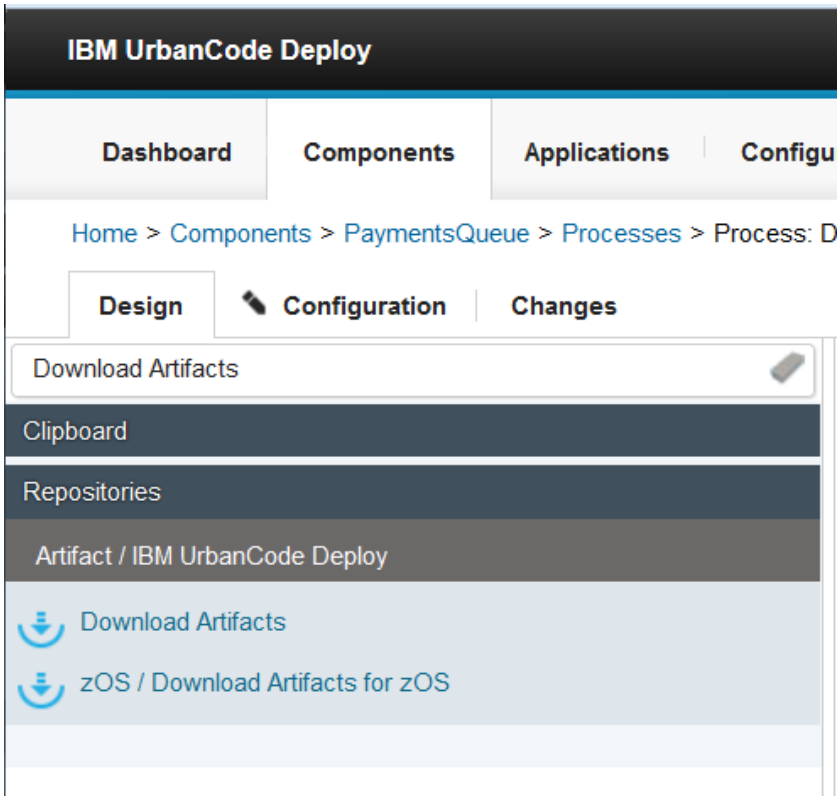


Figure 24: Download Artifacts step

Sample properties for this step are:

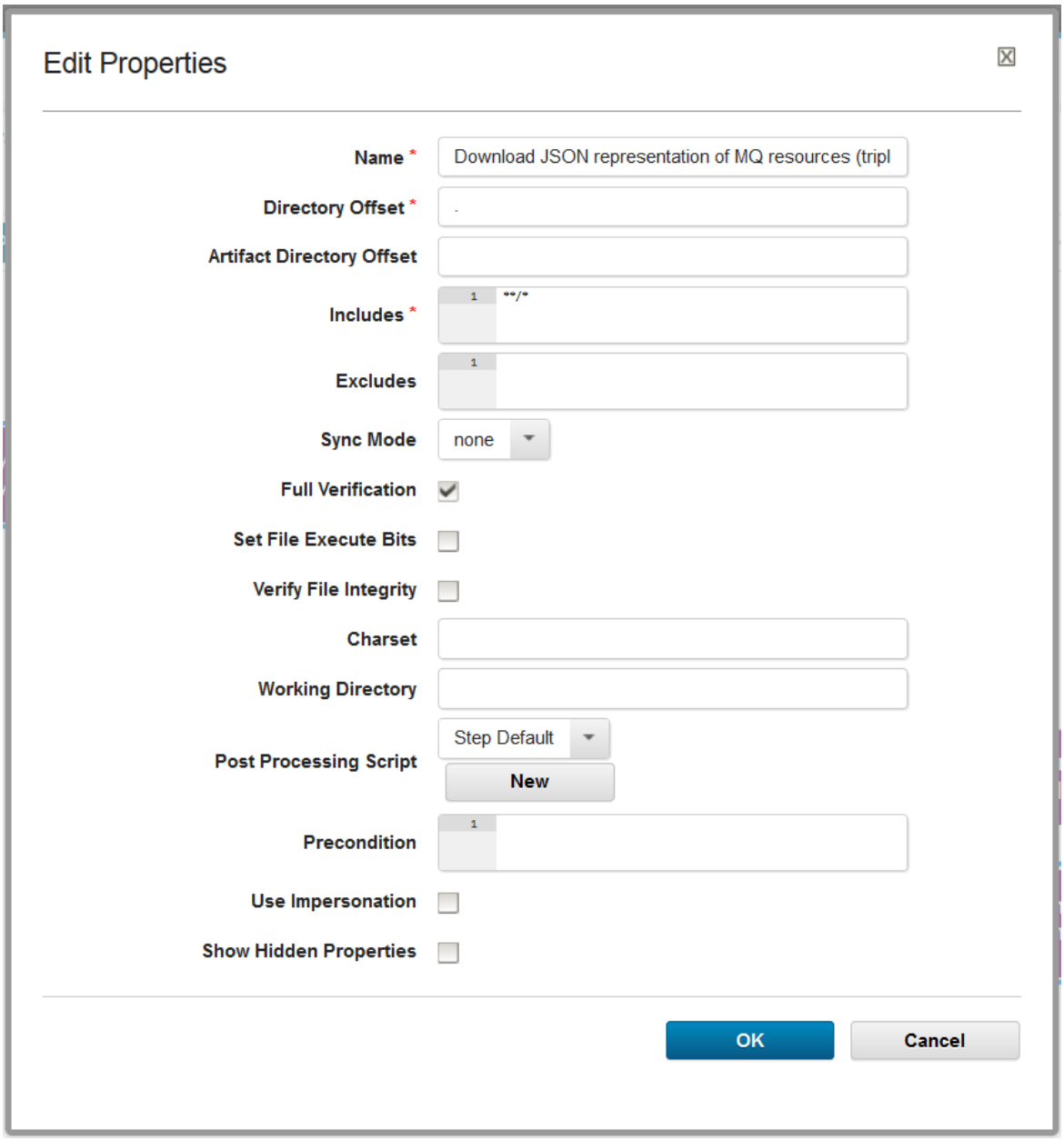


Figure 25: Sample properties of the Download Artifacts step

2) Generate MQSC Commands

Processes the triplet of files (as described above) to generate an output property that contains the MQSC form commands.

The **IBM MQ for z/OS / Generate MQSC Commands** step is used for this:

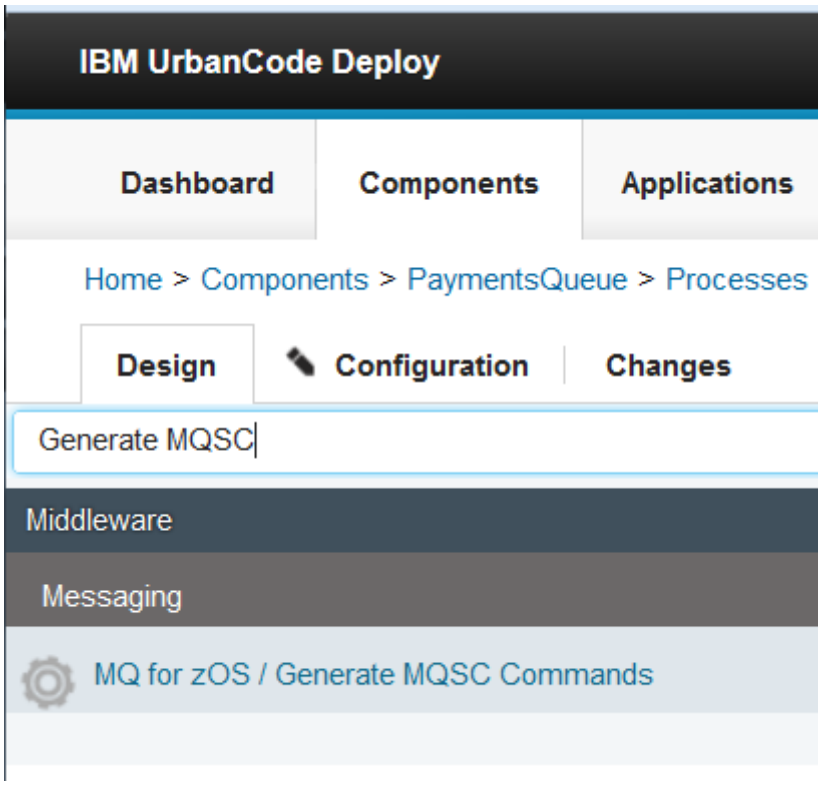


Figure 26: IBM MQ for zOS / Generate MQSC Commands step

Sample properties for this step are:

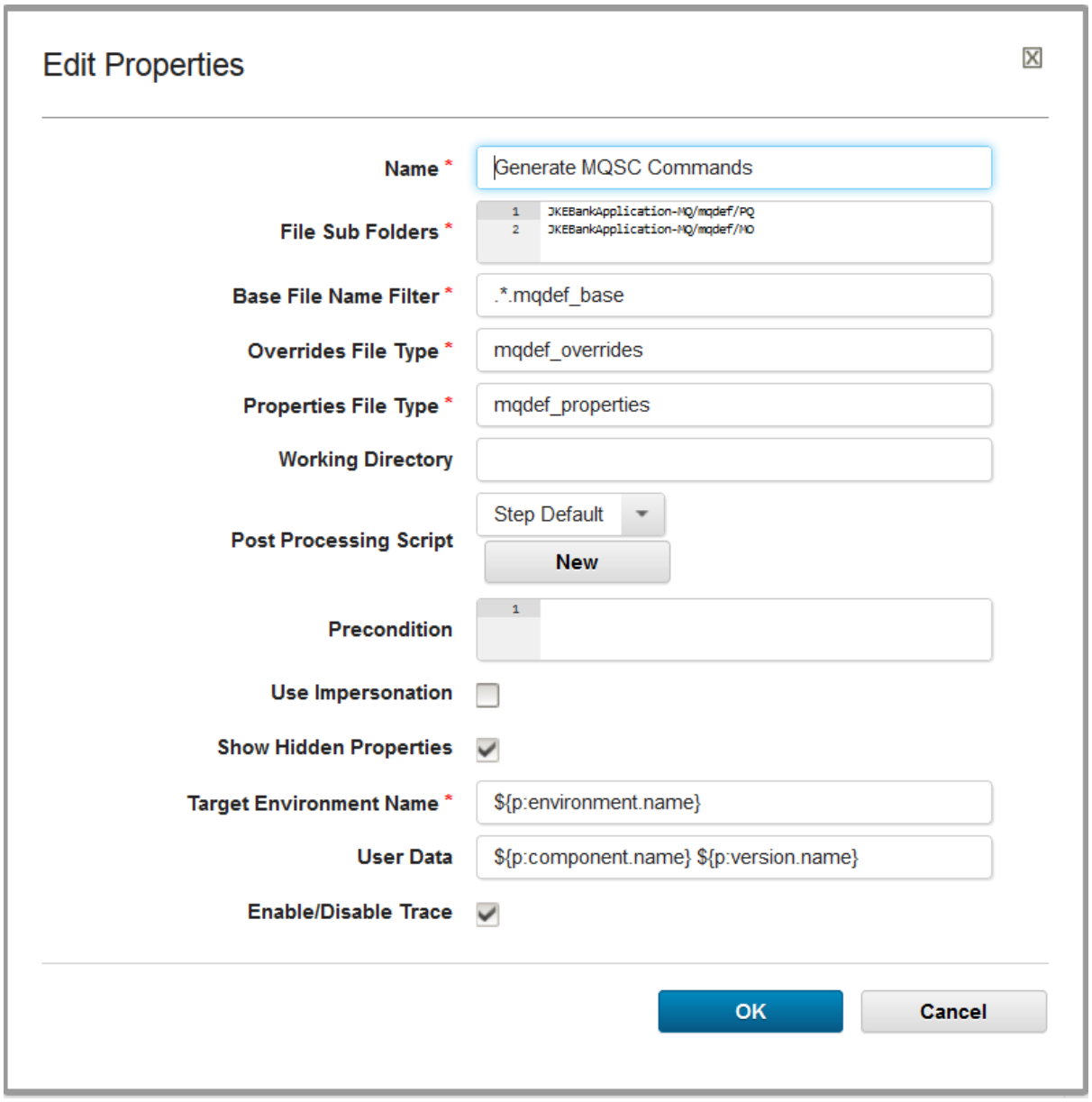


Figure 27: Sample Properties of the IBM MQ for zOS / Generate MQSC Commands step

Where the properties are:

Property	Description
Name	Name of the plugin. The default name is **Generate MQSC Commands**. This is the name that will appear when the plugin is added as a step into a IBM UCD process. There is no need to change this name but you can change it if you wish.
File Sub Folders	A list of application specific folders defined in the SCM system that contain the triplet of files to be deployed. Do not include leading or trailing file separators. Files are generally packaged (e.g. snapshot) in the SCM system with the application specific sub folders in place. Snapshots, for example, are imported, as **Versions**, into the IBM UCD artifact repository called CodeStation. By default, Versions are imported into IBM UCD into the current default working directory with the application specific sub folders in place. For a given application, the plugin needs to know the file sub folders in order to locate the triplet of files. **Note:** More than one set of triplets of files can be defined in an application specific folder as long as each triplet of files has a different name.
Base File Name Filter	A filter to select base files. You can specify any file name and type you want for the base files so set the filter as required. The default filter value is set to: **.*.mqdef_base**, which will result in a search (in the current working directory plus the specified file sub folders) for base files with a file type of **mqdef_base**.
Overrides File Type	The file type for overrides files. You can specify any file type you want for the overrides files. The file name however **must be**

	the same as the file name of the corresponding base file. The default file type is **mqdef_overrides.** The plugin searches for the overrides file in the current working directory plus the specified file sub folders (see above).
Properties File Type	The file type for properties files. You can specify any file type you want for the properties files. The file name however **must be the same** as the file name of the corresponding base file. The default file type is **mqdef_properties.** The plugin searches for the overrides file in the current working directory plus the specified file sub folders (see above).
Working Directory	An alternative absolute path to the working directory for the step. If left blank, the default process specific working directory is used.
Post Processing Script	The post processing script to use to determine pass/fail and lines of interest.
Precondition	A script to determine whether this step should run. This must evaluate to true or false if not left blank.
Use Impersonation	Run this task as a different user.
Show Hidden Properties	A check box which indicates whether hidden properties are to be displayed or not. If checked, hidden properties are displayed.

Target Environment Name	The name of the target environment that resources are to be deployed to. By default, this field is set to variable: **${p:environment.name}** so that its value is automatically resolved to the name of the target environment at deployment time. **There is no need to change this value.**
User Data	Any user data that is to be included with the generated MQSC. By default, this field is set to variables: **${p:component.name} ${p:version.name}** so that the name and version of the component being deployed are included as comments in generated MQSC. This is to help identify which version of a component was deployed to generate the MQSC commands.
Enable/Disable Trace	A check box which indicates whether additional trace data is to be displayed during execution of the plugin. This can generally be left unchecked but it can be checked to obtain trace data if diagnosing issues. By default, this field is **unchecked**.

Table 5: Properties of the IBM MQ for z/OS Generate MQSC Commands plugin

The notation for referencing properties in IBM UCD is ${p:…}. Properties can be defined on various IBM UCD artifacts. For further information, please refer to: http://www.ibm.com/support/knowledgecenter/en/SS4GSP_6.0.1/com.ibm.udeploy.reference.doc/topics/ud_properties.html

Following execution of this step output variable **mqscResourceDefinitions** is set to the MQSC form of the resource definitions. This variable is used as input to the next step in the UCD process.

3) *Submit MQ CSQUTIL Job*

Submits the IBM MQ CSQUTIL batch job to define the MQSC form resources to the target IBM MQ for z/OS queue manager.

*The **Submit Job** step is used for this:*

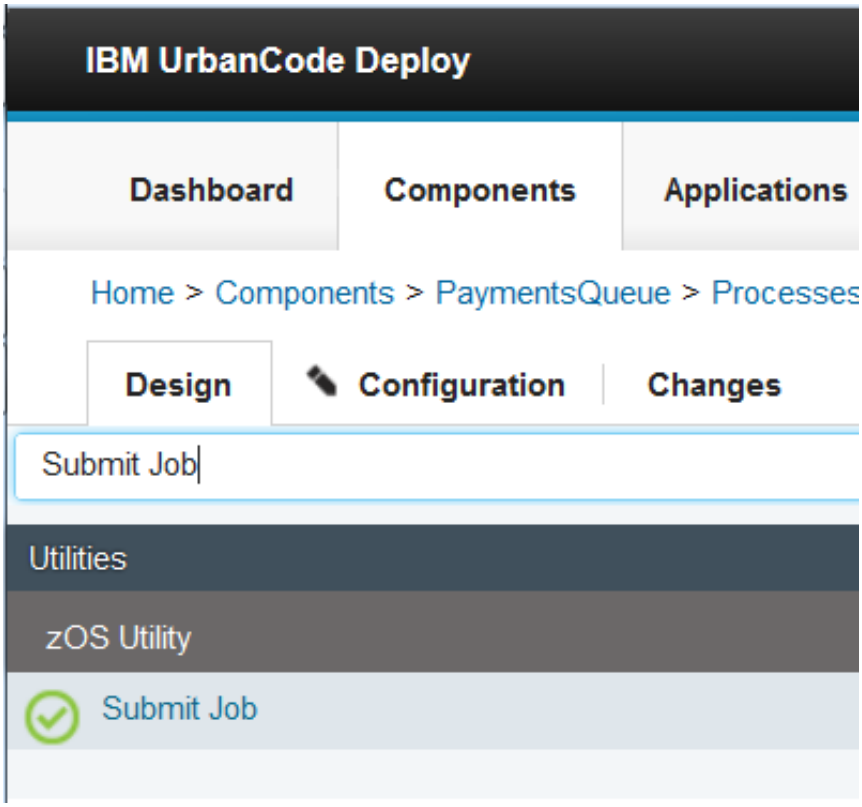
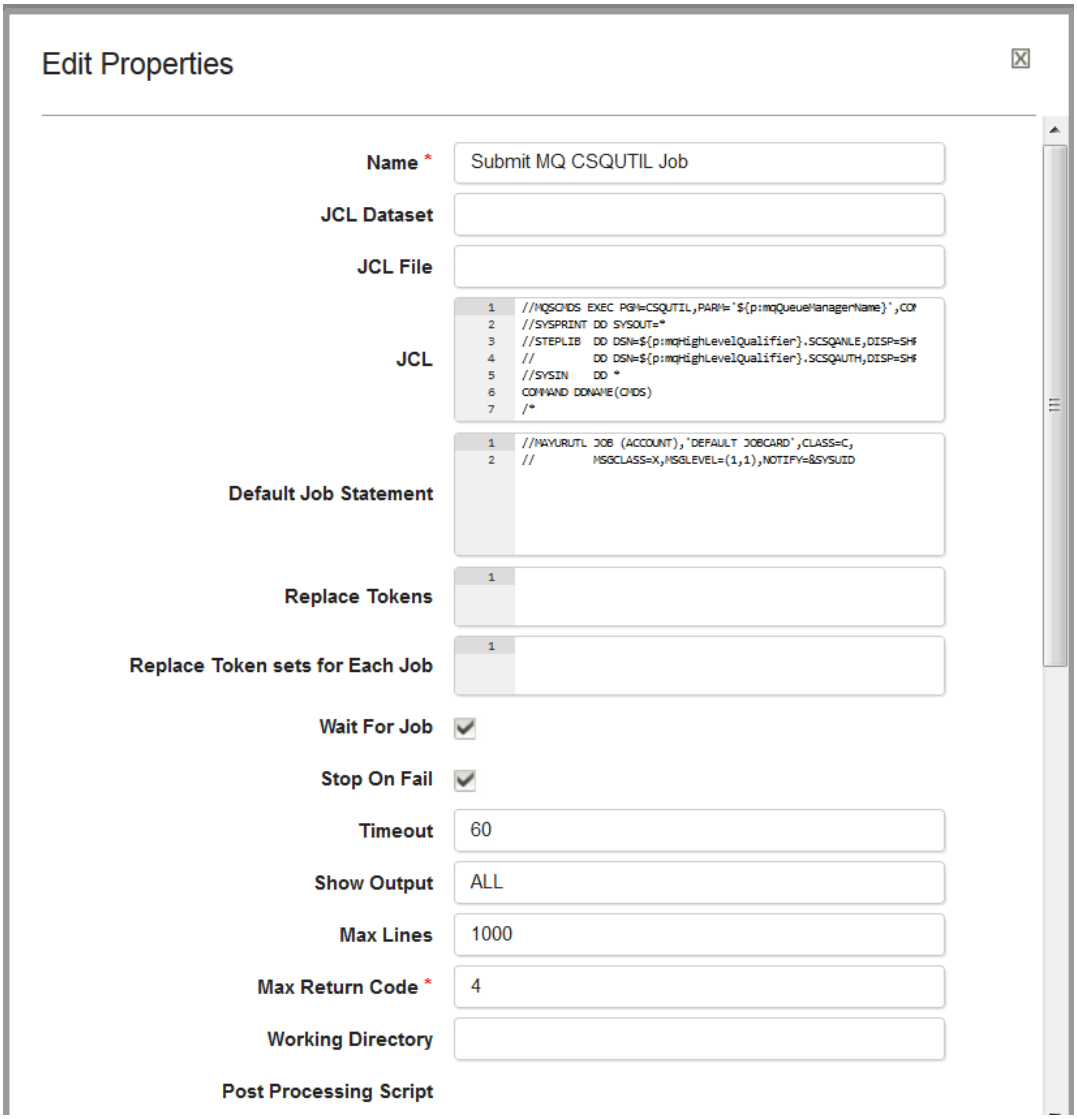


Figure 28: Submit Job step

Sample properties for this step are:



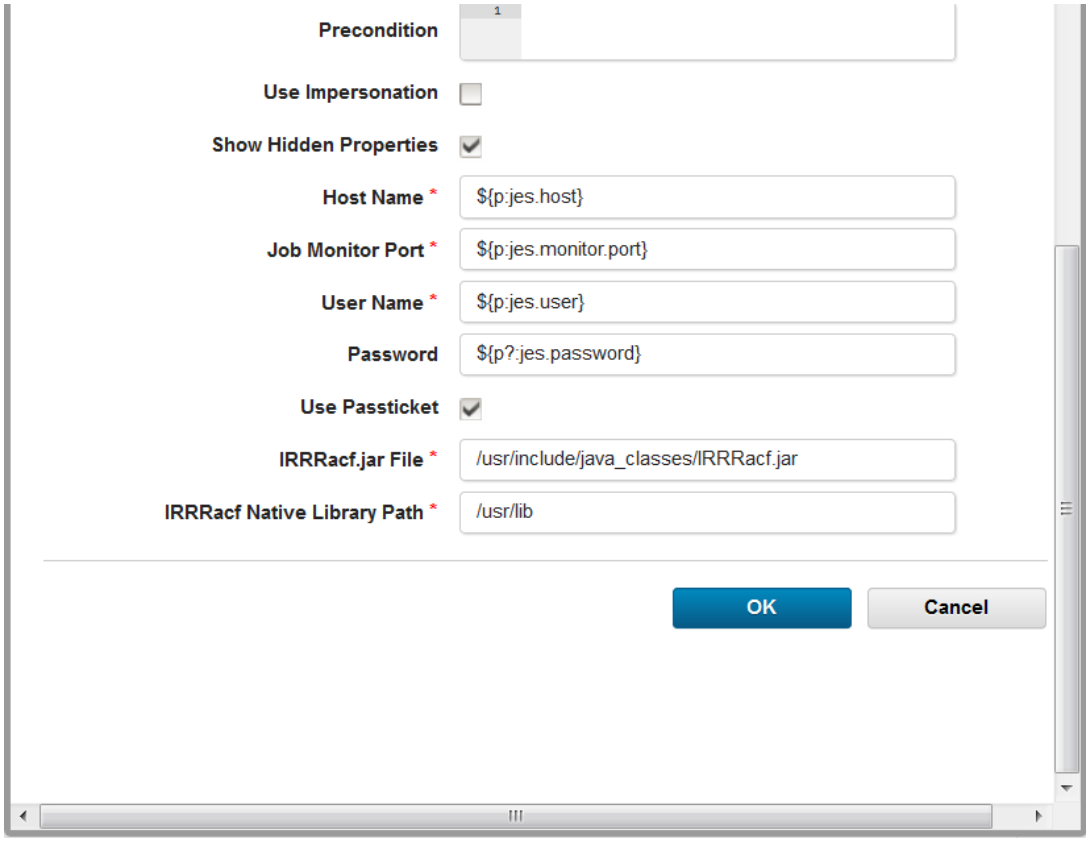


Figure 29: Sample properties of the Submit Job step

Set field **JCL** to the value of the JCL for the IBM MQ CSQUTIL job and concatenate to it **mqscResourceDefinitions**, the output variable from the previous step in the UCD process. For example:

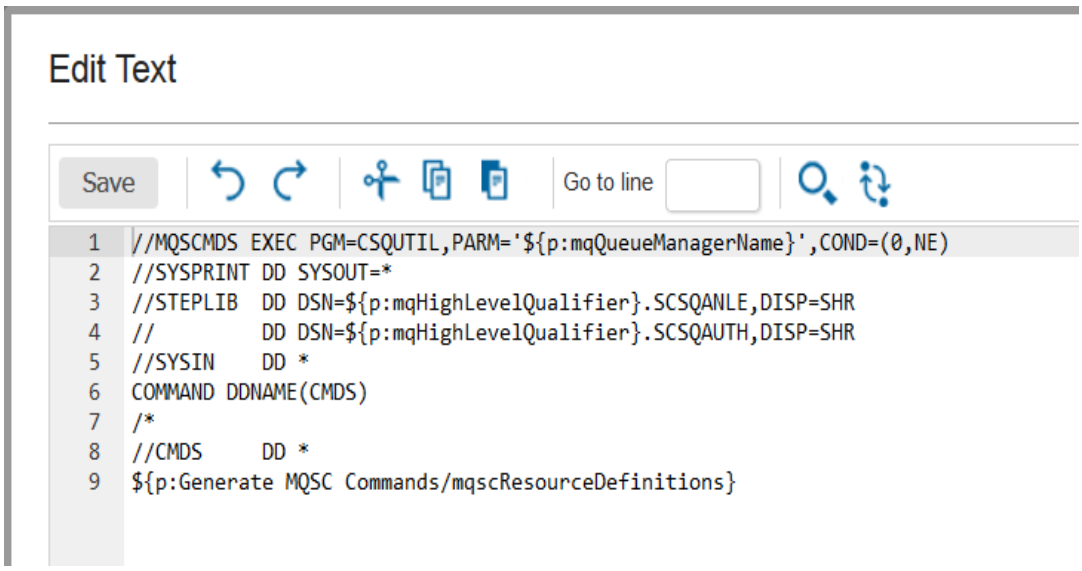


Figure 30: Sample IBM MQ CSQUTIL JOB JCL with output variable mqscResourceDefinitions

Note: Define variables ${p:mqQueueManagerName} and ${p:mqHighLevelQualifier} on the IBM UCD Component associated with the application. Set them to the name of the

IBM MQ for z/OS queue manager that resources are to be deployed to, and to the high level qualifier of the IBM MQ load libraries (see example below).

Set field **Default Job Statement** to the default job card to be used. For example:

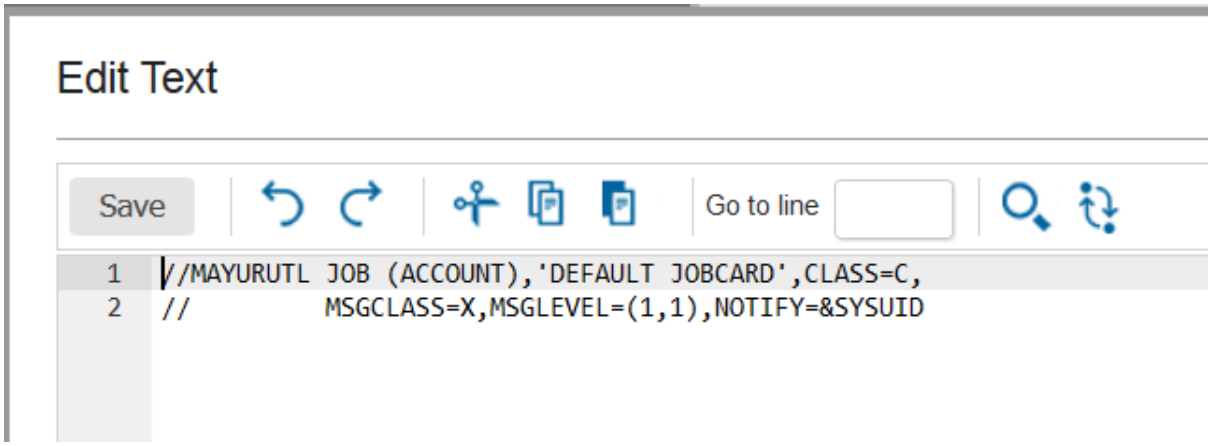


Figure 31: Sample default Job statement for the Submit Job step

Note: You can also define the default job statement in a variable on an IBM UCD artefact and set the Default Job Statement field to the name of the variable. For example, you can define a component variable called **${p?:deploy.env.jobstatement}** and set it to the default job statement.

The submit Job step requires that variables **jes.host**, **jes.user**, **jes.monitor.port** and **jes.password** are set as environment properties (on the component, for example):

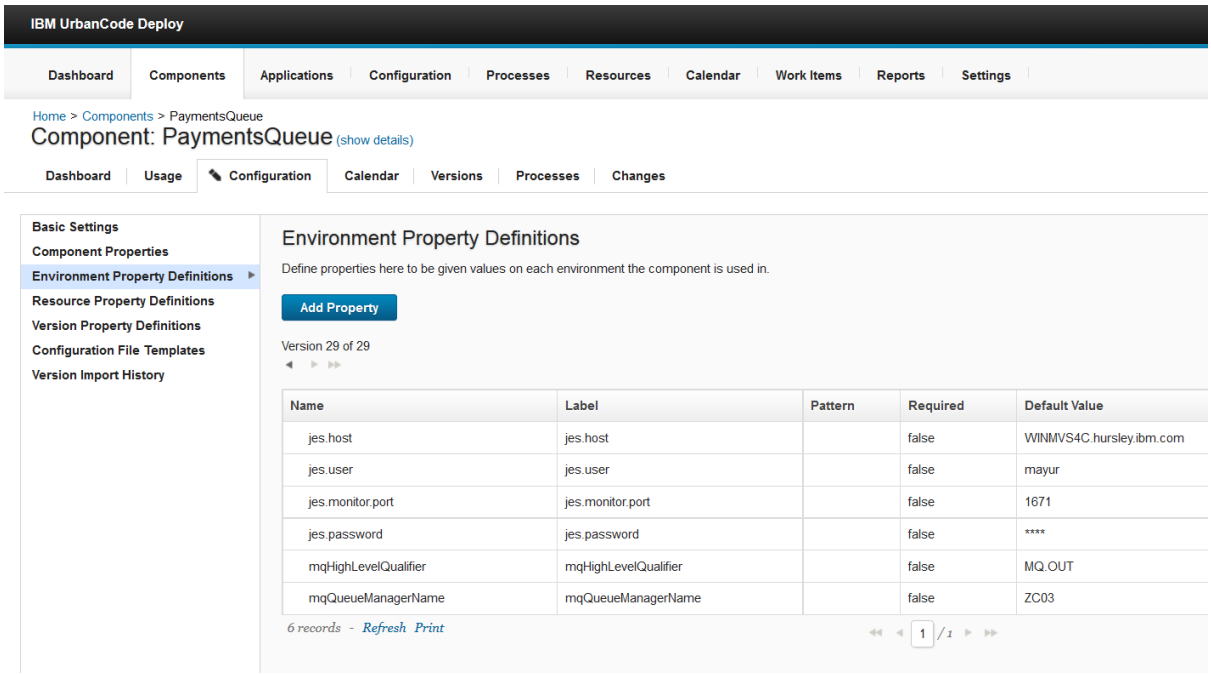


Figure 32: Environment properties required by Job Step

Output generated by running the above process

When the above process is run, as part of an IBM UCD application component, against environment **Test** and with the triplet of files documented above, an example of the **CSQUTIL JCL** generated is:

```
//MAYURUTL JOB (ACCOUNT),'DEFAULT JOBCARD',CLASS=C,JOB19336
//         MSGCLASS=X,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//MQSCCMDS EXEC PGM=CSQUTIL,PARM='ZC03',COND=(0,NE)
//SYSPRINT DD SYSOUT=*
//STEPLIB  DD DSN=MQ.OUT.SCSQANLE,DISP=SHR
//         DD DSN=MQ.OUT.SCSQAUTH,DISP=SHR
//SYSIN    DD *
//CMDS     DD *
```

Figure 33: Sample generated CSQUTIL JCL

The **component name** and the **version name** are included as comments in generated MQSC. This is to help identify which version of a component was deployed to generate the MQSC commands. For example:

```
*
* User Data: PaymentsQueue PaymentQ47
*
```

The following MQSC commands are generated and submitted as part of the CSQUTIL job:

```
*
* PQ1.mqdef_base, PQ1.mqdef_overrides, PQ1.mqdef_properties
*
DEFINE CHANNEL('DRQM.TO.DLQM') +
       CHLTYPE(RCVR) +
       DESCR('Receiver Channel for Test Environment') +
       MAXMSGL(32768) +
       TRPTYPE(TCP) +
       REPLACE

DEFINE QLOCAL('PAYMENT.QUEUE') +
       DEFSOPT(SHARED) +
       DESCR('Queue for bank payments') +
       MAXDEPTH(20000) +
       MAXMSGL(32768) +
       NOTRIGGER +
       TRIGTYPE(NONE) +
       REPLACE
```

Figure 34: Sample generated MQSC commands

Note: The names of the triplet of files used to generate the MQSC commands appear in comments above the MQSC commands. If a second set of files exist for a single component,

their names will appear above the respective MQSC commands generated from them.

Messages issued by the IBM MQ for z/OS Generate MQSC Commands plugin

- **INFORMATION** messages start with '**** INFORMATION:**' and end with '******'. Information messages are just issued as plain text when the plugin step is run.
- **WARNING** messages start with '**** WARNING:**' and end with '******' characters. Warning messages are generally issued if unexpected data is encountered in the triplet of files.
- **ERROR** messages start with '**** ERROR:**' and end with '******' characters. Error messages are generally issued if the triplet of files cannot be located or if there are errors in the representations in any of the triplet of files.

If required, the starting text can be searched for in the output generated by the IBM MQ for z/OS Generate MQSC Commands plugin.

Exceptions issued by the IBM MQ for z/OS Generate MQSC Commands plugin

The IBM MQ for z/OS Generate MQSC Commands plugin can throw:

- **IllegalArgumentException**
- **AssertionError**

Exceptions start with text **'** ERROR: Error generating mqsc commands from input**'.

If an error is encountered when running the IBM MQ for z/OS Generate MQSC Commands plugin, this text can be searched for in the generated output to verify if an exception was thrown and, if thrown, the reason for it being thrown.

A **JsonException** can be encountered if there are errors in the JSON structure in any of the triplet of files:

```
Data: fileName: /MV4C/udeployagentr/opt/ibm-ucd/agent/var/work/PaymentsQueue/
JKEBankApplication-MQ/mqdef/PQ/paymentQ.mqdef_overrides
** ERROR: Error generating mqsc commands from input : Expected a value or a closing
square bracket ']' on line: 131 column: 3.
But got '}' instead. **
groovy.json.JsonException: Expected a value or a closing square bracket ']' on line: 131
column: 3.
But got '}' instead.
at groovy.json.JsonSlurper.parseArray(JsonSlurper.java:147)
```

If this occurs, validate the JSON structures for correctness. Free to use JSON structure validation tools are available on the web.

A **NullPointerException** can be encountered if the overrides file has been left **completely blank.** The overrides file is required and must be set to valid values.

Note: IBM UCD may of course throw other exceptions for other errors.

Trace issued by the IBM MQ for z/OS Generate MQSC Commands plugin

As described above, one of the properties of the plugin is a checkbox to Enable/Disable Trace. If checked, trace is enabled and the following trace entries are produced:

1) **`'Entry:'`** – to trace entry into a method.
2) **`' Data:'`** – to trace any data.
3) **`'Exit :'`** – to trace exit from a method.

Trace can be enabled if required.

