# MDCMS Jenkins Interface Manual

Published December 28, 2021

## Contents



## Overview

The primary objective of the Jenkins interface is to invoke pipeline jobs defined on Jenkins servers to build, test, deploy, review and rollback artifacts on remote platforms from within an MDCMS RFP and to then capture the

result and console logs from those jobs. This way, MDCMS acts as a centralized deployment manager across all platforms for an organization.

## Prerequisites

- MDCMS v8.3 or higher must be installed and licensed on at least one IBM i partition
- MDOpen v8.3 or higher must be licensed on the same IBM i partitions
- MDWorkflow Base + Pipeline must be licensed on the same IBM i partitions
- The user(s) responsible for configuring the interface must have MDOpen v8.3 or higher installed in a version of the Eclipse IDE, such as the standard, free Eclipse IDE from Eclipse.org or Rational Developer for i from IBM
- The user(s) responsible for configuring the interface in MDOpen must be authorized to MDSEC Code 5 (Attribute Maintenance) and MDSEC Code 10 (Server Location Maintenance).
- The user(s) responsible for configuring the interface in Jenkins must have access to the Jenkins Dashboard and sufficient knowledge and privileges in Jenkins to:
  - create service users and add API tokens
  - add and update Jenkins plugins
  - add and update Declarative Pipelines
- The network firewall must allow bi-directional http(s) traffic between the MDCMS partitions and Jenkins servers. The port to allow on the Jenkins side is the port that the Jenkins server listens to. The port to allow on the MDCMS side is either the port defined for the MD REST API server or the port defined for the http server that forwards requests to the MD REST API server.
- The following Jenkins Plugins are required to be installed for the bi-directional interface with MDCMS to work:
  - Pipeline: Basic Steps
  - Pipeline: Declarative
  - Pipeline: Nodes and Processes
  - Http Request
  - Timestamper
  - Skip-certificate-check
- The following Jenkins plugins are optional
  - Declarative Pipeline Migration Assistant – if projects you need to trigger from MDCMS are Freestyle Jobs, you will need to migrate them to declarative pipeline jobs. This “Declarative Pipeline Migration Assistant” plugin can be used to create the necessary job scripts.

# Configure Connection to Pipeline Servers

Before MDCMS can communicate with a Jenkins server, the location and credentials must be defined. The user that will configure this must have MDSEC authority to code md/10 – Server Location Maintenance.

Take the following steps to add a Pipeline Server connection:

1. Within MDOpen, expand the repository for a partition and then expand Settings
2. Left-click on Pipeline Servers. The Pipeline Servers view will open and list any already defined servers
3. Within the view, right-click and select option Add (or Copy if a similar item already exists)
4. A dialog will pop up with the following fields:

| Property | Description |
|---|---|
| Server ID | A 10-character field to uniquely identify the server definition. The ID must be unique amongst all FTP servers and Pipeline servers. The rename option can be used to change the ID on the definition and any setting that depends on the definition. |
| Description | A description of the server to make it easy to identify from a list |
| Pipeline Type | The type of pipeline server. Select Jenkins |
| Pipeline Server URL | The URL of the server that MDCMS will use to communicate with using RESt APIs. The URL should include the http://, the address and the port number if not the default http or https port.<br>For example: https://jenkins.mycompany.com:1234 |
| User | A user id that is registered in Jenkins and has an API Token defined for it |
| Set New Token | The API token value registered for the user that is to be used by MDCMS. A token can be created by logging on in Jenkins as that user and then clicking the Add new Token button within the API Token section of the Configure panel for the user. |
| Proxy Address | The address of a proxy server to route the HTTP connection through, if necessary |
| Proxy Port | The port number of the proxy server to route the HTTP connection through |

| Property | Description |
| --- | --- |
| Proxy Type | • HTTP<br>• SOCKS5 |
| Proxy User | The user id for the connection to the Proxy Server, if necessary |
| Set New Proxy Password | The password for the Proxy User, if necessary |

1. Once the field values have been entered, click the Save button

### Test Connection to Pipeline Server

The connection can be tested by right-clicking on the row in the Pipeline Server view and selecting option Test Connection. The results of the test, and any interaction from MDCMS to the server, are available in the pipeline logs.

To view the Pipeline logs, right-click on a row in the Pipeline Server view and select option Logs. Then left-click on the log for the current date to view the contents.

The pipeline consumer logs are stored as IFS files (one per day per pipeline server) in folder /MDCMS/logs/<instance>/pipeline. A log file is deleted after n days, based on the retention settings in the MDCMS Log maintenance screen. The default is to retain these IFS files for 10 days.

## Configure *PIPE Attributes

There is a special MDCMS Object Type, ***PIPE**, that can be assigned to an attribute. Attributes of this type indicate to MDCMS that 0 or more Pipeline jobs will be invoked for object requests assigned to the attribute when an RFP runs for a target level.

The same attribute ID should be defined for each level in a migration path so that the object requests will migrate from level to level. Then, for each level, Attribute Pipelines will be defined if any pipeline jobs should be invoked for that level. If no pipeline execution is required for a specific level, MDCMS will simply migrate the request records in the RFP without performing activity on them.

If certain artifacts will require a different set of Pipeline jobs then other artifacts, then a separate attribute should be created for them.

The user that will configure an attribute must have MDSEC authority to code 5 – Attribute Maintenance for the target Appl/Lvl.

Take the following steps to add a *PIPE Attribute:

1. Within MDOpen, expand the repository for a partition and then expand Settings
2. Left-click on Attributes. The Attributes view will open and list any already defined attributes based on the filter values.
3. Within the view, right-click and select option Add (or Copy if a similar item already exists)
4. A dialog will pop up with many fields. Only the following fields are relevant for *PIPE attributes:

| Property | Description |
|---|---|
| Application | The Application ID |
| Level | The target promotion level in the application |
| Object Type | Set to *PIPE |
| MDCMS Attribute | A 10-character ID to identify the group of artifacts that will be deployed by pipeline jobs using this attribute |
| Description | An optional description of the attribute |
| Target Library | An optional value that can be used as a wildcard to pass to pipeline job parameters. This, along with other parameters, may make it easier to reuse the same pipeline jobs for multiple environments. |
| Require Approval | Set to true if an authorized user must approve an RFP after the Build/Test phase is complete before the artifacts can be deployed. If an RFP contains object requests for this attribute, it will wait for approval when it otherwise wouldn’t, if approval isn’t always required for the target level. |
| Acceptance Group Type | A user group type can be defined for the attribute and when defined, once objects of the given attribute are installed into the target level, a member of the group defined for the impacted project(s) must review and then accept the results of the installation before the RFP can continue to the next level in the migration path.<br>This group type will be in addition to any Acceptance Group Types defined for all objects for the level. This way, this additional acceptance is only necessary for these special attributes. For example, |

| Property | Description |
| --- | --- |
| | a pipeline job may need to perform automated testing on the deployed objects or a release manager may need to manually review the state of the artifacts on the remote platform in the test environment before the RFP can be promoted to Production. |

1. Once the field values have been entered, click the OK button

## Configure Attribute Pipelines

Once a Pipeline Server and a *PIPE attribute is defined, Attribute Pipelines can be defined to specify which Pipeline Jobs should run for which attribute at each promotion level.

The user that will configure an Attribute Pipeline must have MDSEC authority to code 5 – Attribute Maintenance for the target Appl/Lvl.

Take the following steps to add an Attribute Pipeline:

1. Within MDOpen, expand the repository for a partition and then expand settings
2. Left-click on Attribute Pipelines. The Attribute Pipelines view will open and list any already defined pipelines based on the filter values. Alternatively, right-click on a Pipeline Server and Select option Attribute Pipelines to list all Pipelines for that server or right-click on a *PIPE attribute and Select option Attribute Pipelines to list all Pipelines for that attribute.
3. Within the view, right-click and select option Add (or Copy if a similar item already exists)
4. A dialog will pop up with the following fields:

| Property | Description |
| --- | --- |
| Application | The Application ID |
| Level | The target promotion level in the application |
| MDCMS Attribute | A *PIPE attribute to attach a pipeline job to |
| Phase | The phase of the RFP during which the pipeline job should be executed<br><br>• Build/Test – this phase coincides with the Submit or Bundle phase of the RFP, which is when objects are compiled and validated, but before |

| Property | Description |
| --- | --- |
| | any objects should be updated in the target environment.<br><br>• Deploy – this phase coincides with the Install phase of the RFP, which is when the target application is updated with new, changed or deleted objects. If the pipeline job is configured to perform the build and deploy at the same time, then set the Attribute Pipeline phase to Deploy. It is recommended to keep build separate from deploy, though, when possible, using conditional stages or multiple job definitions.<br><br>• Acceptance – this phase is executed when the RFP has completed installation in order to perform automated testing and acceptance of those tests against the target environment.<br><br>• Rollback – this phase is executed automatically if the RFP fails at a step after the Deploy phase completed, or at a later date, if the successful RFP is rolled back and executed as a new rollback RFP. |
| Sequence | The sort sequence of the pipeline job when multiple pipelines are defined for the same attribute and phase. |
| Server ID | The ID of the defined Pipeline Server on which the Pipeline job will be executed. |
| Pipeline Job | The full name of the pipeline job to be executed. This name must match the name of an existing Pipeline job on the Jenkins Server. |
| Description | An optional description of the job |
| Run for Modifications | A checked value (Y) indicates this pipeline job should run for new or changed objects. |
| Run for Deletions | A checked value (Y) indicates this pipeline job should run for deleted objects. |
| Ignore Errors | A checked value (Y) indicating if the RFP should continue with warnings if the pipeline job execution fails. |

| Property | Description |
| --- | --- |
| Frequency | • Once per RFP – the pipeline job will run once for an RFP, if at least one request record of the given MDCMS attribute is assigned to the RFP.<br>• Once per Revision – the pipeline job will run once for each distinct set of object requests sharing the same Git Revision hash or SVN Revision number.<br>• Once per Folder – the pipeline job will run once for each distinct set of object requests residing in the same parent folder.<br>• Once per File – the pipeline job will run once for each object request of the given attribute. |
| Timeout in Seconds | The amount of time, in seconds, that MDCMS should wait for the pipeline job to respond as finished before MDCMS times out. A timeout will be treated as a warning or exception, depending on the Ignore Errors checkbox. |
| Run Concurrently | MDCMS can execute several different pipeline jobs concurrently, if those jobs aren't dependent on each other. A checked value (Y) indicates that this job can run concurrently with other jobs that are also marked as allowed to run concurrently. If this job, or an already running job doesn't allow concurrent execution, then this job won't be started until the currently running job has finished. |
| Workflow Acceptance Group | When phase Acceptance is selected, this required field is shown in order to register the defined User Group that contains the user to designate the accept/reject result to. The user group must be assigned to the project(s) impacted by the RFP's object requests as an acceptance role for either the target level or for the *PIPE attribute for the target level. |
| Workflow Acceptance User | When phase Acceptance is selected, this required field is shown in order to register the defined user to designate the accept/reject result to. The user must belong the Workflow Acceptance Group. The user must |

| Property | Description |
| --- | --- |
| | also be defined in MDSEC as either an actual user or as a System Process User. |

Once the field values have been entered, click the OK button

## Attribute Pipeline Parameters

Left-Click on a row in the Attribute Pipelines view to edit the configuration of that Attribute Pipeline.

Besides the fields described in the Add dialog, there is also a listing of any parameters defined for the Attribute Pipeline.
For example:

| Parameter | Description | Parameter Value |
| --- | --- | --- |
| ARTIFACT_DIRECTORY | Folder containing the items to sync to target | *PASSTHRU |
| BACKUP_DIRECTORY | Loc to store prior version of folder contents | /var/backup/php-qa |
| TARGET_DIRECTORY | build target | ++OBJLIB++ |

Attribute Pipeline Parameters are passed as Parameter/Value pairs to the job when it is requested to be executed. These coincide with parameters that are defined for the Pipeline job that are then used within the Pipeline script instead of hardcoded values.

There are 2 String Parameters that must be defined for every pipeline job that will be invoked by MDCMS. These 2 parameters MUST NOT be additionally defined within MDOpen in the list of parameters for the Attribute Pipeline, because they are automatically passed by MDCMS. When configuring the pipeline job within Jenkins, ensure the checkbox “This project is parameterized” is checked and that these 2 String Parameters are defined without default values:

- MD_TRANS
- MD_CALLBACK

Any other parameters that are required by your script should be defined in Jenkins and in MDOpen.

To add a parameter in MDOpen, right-click within the Parameter List in the Attribute Pipeline editor and select option Add Attribute Pipeline Parameter. The following fields will appear:

| Property | Description |
| --- | --- |
| Application | The Application ID |
| Level | The target promotion level in the application |

| Property | Description |
|---|---|
| MDCMS Attribute | A *PIPE attribute to attach a pipeline job to |
| Phase | The phase of the RFP during which the pipeline job should be executed |
| Sequence | The sort sequence of the pipeline job |
| Parameter | The name of a non-MD Parameter that is defined for the Jenkins Pipeline job. The name must match exactly to be used. |
| Description | An optional description of the parameter |
| Parameter Value | The value to pass to the job. This can be specific text, or a collection of text and wildcards. You can prompt for wildcards using content-assist (ctrl-space). The wildcard will be replaced with the runtime value when execution occurs. Many wildcards are packaged with MDCMS and additional, custom wildcards can be defined per application level.<br>***PASSTHRU** – this special value for a parameter indicates that the value should be retrieved from the result of a previously executed pipeline job for the RFP. |

Options to manage Parameters across Attribute Pipelines

Apply attribute pipeline parameter – when this option is selected, every Attribute Pipeline will be listed where the Parameter doesn't exist or the Parameter exists, but with a different description or value. Check the description or value checkbox for each Attribute Pipeline that you wish to apply the values of the parameter to.

Delete attribute pipeline parameter – when this option is selected (multi-select is enabled for this option), the current Attribute Pipeline will be shown at the top of the parameter deletion list with the Delete checkbox checked by default. Below that entry will be every other Attribute Pipeline where the parameter(s) are defined and the option to delete them from those entries as well.

## Configure MDCMS REST API Server

When an MDCMS RFP invokes a Pipeline job, it passes along as parameter MD_CALLBACK the MDCMS Webhook to invoke when the Pipeline job is complete. In order for the Webhook to be accepted by MDCMS, a HTTP server for MDCMS must be configured on the partition where the RFP runs. Only one server can be defined per instance of MDCMS per partition and it only needs to be configured once. If your instance of the MDCMS http server isn't yet configured, perform the instructions in the next 2 sections.

## Set/Change URL

Within a 5250 Emulator, take the following steps to get to the URL configuration screen:

1. Command MDCMS from a command line
2. Option 1=MDCMS Setup Menu
3. Option 10=Interface Settings
4. Option 9=MD REST API Server
5. Option 1=Set/Change URL

MDLRURL COMPANY NAME 7.04.19
SCRN1 Set MD REST API URL Endpoint 13:04:38

Current Endpoint: https://mysystem.mycompany.com

New Endpoint....: https://mysystem.mycompany.com_____

Example API URL.: https://mysystem.mycompany.com/mdcms/applications

WARNING: Remember to update any webhook definitions when modifying a previously active URL Endpoint

Enter=Confirm F12=Cancel

The Endpoint is the URL for the partition where the MDCMS REST server is hosted. If connecting directly to the server, then include the port number in the URL (example: https://mysystem.mycompany.com:2121)

Otherwise, if the using a forward proxy from the default server, the port numbers can be excluded.

The services themselves use the context path of <Endpoint>/<mdcms instance>/<resource> and the instance name and resource are appended automatically, so do not include them in the definition of the Endpoint on this screen.

## Generate MDCMS REST API Server

Within a 5250 Emulator, take the following steps to get to the REST API server generation screen:

1. Command MDCMS from a command line
2. Option 1=MDCMS Setup Menu
3. Option 10=Interface Settings
4. Option 9=MD REST API Server
5. Option 2=Generate Server

Server Name

Only one server is to be generated for an instance of MDCMS. By default, the name of the server is the same as the name of the instance (example = MDCMS or MDCMSTEST).

The http server can't already exist when requesting to generate it.

Regardless of the name of the server, the context path on the URL is always the name of the mdcms instance in lower case and the server configuration is stored in folder /www/<mdcms instance>.

Port Number

The port number is required and should be set to a number not already used by something else.

### Starting/Stopping the MDCMS REST API Server

The server can be started with command STRTCPSVR SERVER(*HTTP) HTTPSVR(MDCMS)
assuming MDCMS is the name of the server.

It's recommended to configure the server to autostart or to add the command to the QSTRUPPGM instructions.

The server can be stopped with command ENDTCPSVR SERVER(*HTTP) HTTPSVR(MDCMS)
assuming MDCMS is the name of the server.

### Testing/Troubleshooting the MDCMS REST API Server

The server can be tested from a browser using the example API URL that is displayed in the Set/Change URL screen described at the beginning of this chapter. If a timeout occurs, verify that the firewall allows the request through.

You can also check the http logs in /www/mdcms/logs.

If the request gets through but it doesn't respond correctly, check log file LXERRLOG in library MDCMS.

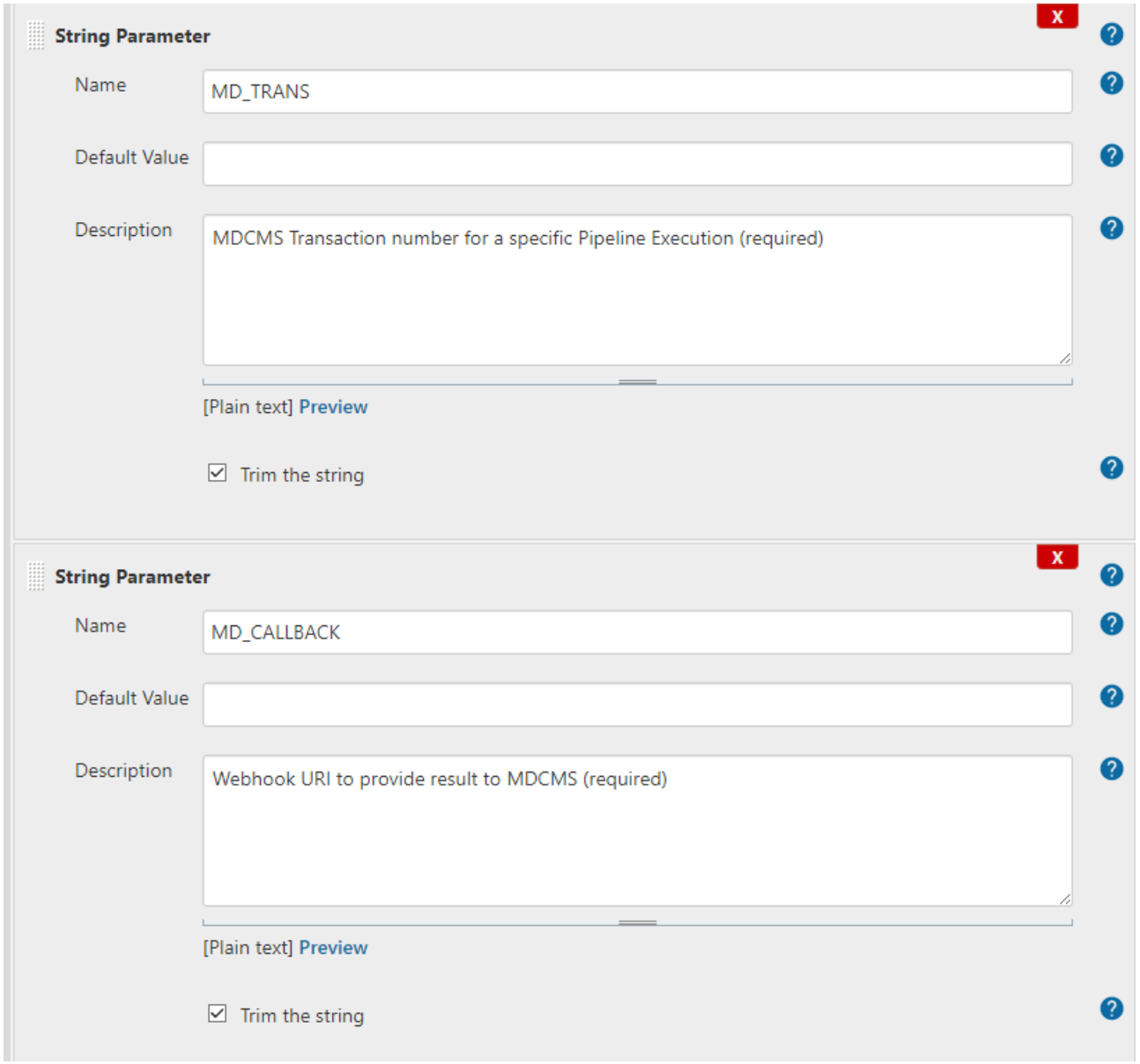
If a Jenkins callback fails, verify the MD_CALLBACK parameter is defined for the Jenkins job and inspect the job's Console Output for detailed information and to verify that the job's script actually tries to invoke the Callback.

## Jenkins Pipeline Job Examples When Jenkins is the Slave

In these examples, MDCMS orchestrates the requests and triggers a Jenkins Pipeline job to carry out build, test and/or deployment tasks.

### Example of Mandatory String Parameters for use with MDCMS




## Example of Jenkins Pipeline Script with MDCMS Callback Including Passthru Parameter

```
// Mandatory – required to build JSON requestBody for MDCMS Callback
import groovy.json.JsonOutput

pipeline {
  agent any
  environment {
    // Optional/Example Variables
    REPODIR = "${env.WORKSPACE}/repo/"
  }
//  Optional/Example Stages. All stages in template are examples.
//  Any stage can be added with any name.
    stages {
    stage('BUILD') {
      steps {
        echo 'Building..'
```

```
        dir("$REPODIR") {
          checkout([$class: 'GitSCM',
            branches: [[name: GITREF]],
            userRemoteConfigs: [[credentialsId: GITCRED, url: GITREPO]]])
        }
        echo 'Successful checkout of commit: ' + GITREF
      }
    }
    stage('TEST') {
      steps {
        echo 'Testing....'
      }
    }
  }
// START - Mandatory post section for MDCMS Interface
post {
   // START - Mandatory always condition for MDCMS Interface do not remove
    always {
      script {
        MD_REQ = JsonOutput.toJson([
            md_trans: MD_TRANS,
            build_num: env.BUILD_NUMBER,
            summary_msg: currentBuild.currentResult,
// Optional - the passthru_name_1 contains the name of a passthru parameter for the MDCMS RFP
to use in subsequent pipeline jobs.
  // The passthru_value_1 contains the value of that parameter that the RFP will store and
reuse.
 // up to 90 passthru parameter name/value pairs (_1 to _90) can be passed back to MDCMS.
// remove the passthru name and value pairs if you do not require passthru variables returned
from this job
            passthru_name_1: "ARTIFACT_DIRECTORY",
            passthru_value_1: REPODIR,
            build_url: env.BUILD_URL
        ])
      }
      echo 'CallBack to MDCMS using : ' + MD_REQ
      timeout(unit: 'SECONDS', time: 60) {
        httpRequest consoleLogResponseBody: true,
        contentType: 'APPLICATION_JSON_UTF8',
        httpMode: 'POST',
        ignoreSslErrors: true,
        quiet: false,
        requestBody: MD_REQ,
        validResponseCodes: '200',
        responseHandle: 'NONE',
        url: MD_CALLBACK,
        wrapAsMultipart: false
      }
      // END - Mandatory always condition for MDCMS Interface do not remove
      // Additional steps can be added to post.always condition
    }
// END - Mandatory post Section for MDCMS Interface
// Addtional post conditions can be added to the post section
  }
}
```

# Jenkins Pipeline Job Examples When MDCMS is the Slave

In this example, Jenkins builds an archive file destined for the IFS on the IBM i and triggers MDCMS to create an object request for that file as well as passing that file to MDCMS for deployment. Through the use of input stages in the Jenkins job, MDCMS can update the status of the deployment in Jenkins.

## Example Jenkins Pipeline script

```
import groovy.json.JsonOutput
pipeline {
      agent any
  environment {
        // Optional/Example Variables
        WARDIR = "/var/www/jenkins-dev"
        WARFILE = "sample.war"
        APPDIR = "${env.WORKSPACE}/app/"
        MDCMS_URI = "https://dev.mdcms.ch/mdcmst84/object-request"
  }
//  Optional/Example Stages. All stages in template are examples.
//  Any stage can be added with any name.
// the BUILD stage can be triggered by a push to a git repo or manually
    stages {
        stage('BUILD') {
          steps {
            echo 'Building..'
        dir("$WARDIR"){
                echo 'test this first'
            sh 'rsync -avr $WARFILE $APPDIR'
        }
        echo 'Successful build of file: ' + WARFILE
      }
    }
// the MDCMS-PUSH stage invokes the MDCMS REST API object-request to create an
// object request for the archive file and attach the file to the request
    stage('MDCMS-PUSH') {
          steps {
            echo 'Pushing WAR file to MDCMS....'
        script {
            MD_REQ = JsonOutput.toJson([
                appl: 'TEST01',
            lvl: 100,
            objt: '*IFS',
            attr: 'WEBSPHERE',
            objn: 'sample.war',
            user: 'MMORGAN',
            folb: '/home/mmorgan',
            proj: 'DEMOUK1',
            task: 5,
            arfp: '*AUTO',
            rfpd: 'deploy war from jenkins',
            pipe: 'JENKINS',
            pipa: 'JENKSLAVE',
            tkey: env.BUILD_NUMBER
        ])
```

```
      }
        dir("$APPDIR"){
                // echo 'CURL will happen here'
            sh "curl -v POST -H \'Expect:\' -H \'Content-Type: multipart/mixed\' -F
'payload=${MD_REQ};type=application/json\' -F file=@sample.war ${MDCMS_URI}"
        }
      }
    }
// the MDCMS-DEV stage goes into a waiting for input state after the push to MDCMS is complete
// MDCMS will post an ok to the job once the RFP is installed into the DEV environment using
MDUPDPIPE
// the id parameter must match the STAGE parameter on MDUPDPIPE and the ok parameter must be
defined and set // to “ok”
    stage('MDCMS-DEV') {
          input {
            message "Installed to DEV "
        id "DEV"
        ok "ok"
        parameters {
                string(name: 'MD_STATUS', defaultValue: 'Success', description: 'MD Status
Value', trim: true)
            string(name: 'MD_MESSAGE', defaultValue: 'RFP ##### Installed Successfully ',
description: 'MD Return
Message')
        }
      }
      steps {
              script {
                  if (MD_STATUS == 'Abort') {
                    echo 'MD Status is: ' + MD_STATUS
                currentBuild.result='ABORTED'
                error('Aborted by MDCMS for: ' + MD_MESSAGE)
              }
              else {
                    echo 'MD Status is: ' + MD_STATUS
                echo 'RFP Status is: ' + MD_MESSAGE
              }
          }
      }
    }
  }
}
```

## Example MDUPDPIPE command to update Jenkins

The following command would be defined as a *RFP command for the target level with command type of Q:

MDUPDPIPE JOB('MD-mdcms-slave-example') STAGE('DEV') RSVR(*RFP) AGP(##APPLIC##)
RFP(##RFPNBR##) VAR((MD_STATUS Success)
(MD_MESSAGE '##OBJNAM## deployed by RFP ##APPLIC##/##RFPNBR## into level ##PROLVL##'))

| Property | Description |
| --- | --- |
| Parameter | Description |
| JOB | The exact name of the Jenkins job |
| STAGE | The id of the stage in the Jenkins job waiting for input. Note that this must match the explicitly defined id parameter in the Jenkins groovy script and isn't necessarily the stage name. |
| RSVR | *RFP indicates that MDCMS will inspect the RFP for any* PIPE requests in order to know the id of the pipeline server and the build number. This information was passed to MDCMS on the POST request in parameters pipe and tkey. Additionally, parameter pipa was passed that indicates the pipeline attribute. When that occurred, MDCMS added the *PIPE object to the same RFP as the war file. |
| AGP | Must be ##APPLIC## to be replaced with the runtime value of the MD application |
| RFP | Must be ##RFPNBR## to be replaced with the runtime value of the RFP number |
| VAR | Any variable name/value pairs defined in the stage to be used for logging purposes |

## Requesting *PIPE Object Requests When Jenkins is the Slave

Now that everything is configured, it's time to start requesting items and having them built and deployed via Jenkins.

The most common way for files and folders to get requested is through Continuous Integration triggered by a push to a Git or SVN repository. This process is defined in MDOpen and full instructions on that configuration is available in the Continuous Integration tutorials.

If the Continuous Integration definition is set to create *PIPE request records, then those records will be created automatically for each new/changed/deleted file if for diffs, or for all folders and files if for contents.

Each record will contain the Repository type (*GIT or *SVN), Repository ID defined in MDOpen and the revision hash for Git or the revision number for SVN.

If manually creating *PIPE Object Requests, you can manually add the repository information to the Object Request Detail record in the Repository fields near the bottom of the Object Request editor.

If the pipeline job requires information about the SCM repository in order to pull artifacts from it, you can pass that information dynamically as wildcards on Attribute Pipeline Parameters.

SCM Wildcards:

| Property | Description | |
|---|---|---|
| ++GITBRN++ | Git Branch | The name of the Git Branch that an object request originated from. |
| ++GITREV++ | Git Revision | The Git Revision hash of the commit that an object request originated from |
| ++GITSVR++ | Git Server ID | The MDCMS Git server ID that an object request originated from |
| ++GITURL++ | Git Server URL | The Git URL defined for the MDCMS Git Server that an object request originated from |
| ++SVNREV++ | SVN Revision | The string representation of the SVN Revision number that an object request originated from |
| ++SVNSVR++ | SVN Server ID | The MDCMS SVN server ID that an object request originated from |
| ++SVNURL++ | SVN Server URL | The SVN URL defined for the MDCMS Git Server that an object request originated from |

MDCMS does NOT physically store and migrate actual files for *PIPE object requests, keeping the MDCMS process very light and fast. Instead, it utilizes best practices to expect the pipeline jobs to retrieve the files from the SCM themselves when they need it.

If using Continuous Integration or Cross-Platform impact analysis, MDCMS will clone the SCM repository onto the IBM i partition to perform tree-walking or diff-walking, but those files don’t proceed further in the promotion process.

Other Object Request-Specific Wildcards that may be useful as parameters:

| Property | Description | |
|---|---|---|
| ++OBJNAM++ | Object Name | The name of the requested object |

| Property | Description | |
|---|---|---|
| ++OBJDSC++ | Object Description | The object description value can be edited on the object request and modified at every level. |