# Managing Application Deployments using Liberty MBeans

IBM Washington Systems Center

# Managing Application Deployments using Liberty MBeans

## Table of Contents




Mitch Johnson (mitchj@us.ibm.com)

## Java Management Extensions and Liberty

Enabling Java Management Extensions (JMX) provides access from client applications to a Liberty server's administrative functions. Functions such as monitoring heap storage and CPU utilization, displaying the details of the Java virtual machine, dynamically activating XML configuration changes and dynamically deploying and installing application archive files. It is the latter two functions which are the focus of this document.

### Managed Bean (MBeans)

The Java object or code that provides access to an administrative function is referred to as a managed bean (MBean). A MBeans is reusable code with a well-defined client interface whose purpose is to perform a particular administrative function.

- For more information about MBeans see URL https://docs.oracle.com/javase/tutorial/jmx/mbeans/index.html.
- And for a list of Liberty provided MBeans, see URL https://www.ibm.com/docs/en/was-liberty/base?topic=liberty-list-provided-mbeans
- For details on Liberty's support for remote JMX clients, see *Connecting to Liberty using JMX* at URL https://www.ibm.com/docs/en/was-liberty/zos?topic=SS7K4U_liberty/com.ibm.websphere.wlp.doc/ae/twlp_admin_jmx.htm.

IBM JMX Management Extensions are accessible using either two protocols, Remote Method Invocation (RMI) or HTTPS. This document focuses on using HTTPS (accessing an MBean using HTTP is not allowed). As you read the information at these sites and other sources it is useful to understand that references to JMX Connector refers to using RMI from Java clients and JMX REST Connector refers to using HTTPS from REST clients.


Mitch Johnson (mitchj@us.ibm.com)

## Enable the Liberty features required to access MBeans

Liberty features *monitor-1.0* and *restConnector-2.0* are required to enable JMX clients like JConsole and REST clients like Postman and cURL to access a Liberty Server. These features are configured in the <*featureManager/*> section of a server's configuration XML file.

```
<!-- Enable features -->
<featureManager>
   <feature>appSecurity-2.0</feature>*
   <feature>zosSecurity-1.0</feature>*
   <feature>transportSecurity-1.0</feature>*
   <feature>apiDiscovery-1.0</feature>*
   <feature>monitor-1.0</feature>
   <feature>restConnector-2.0</feature>
</featureManager>
```

*Invoking MBeans on a Liberty server requires the use of TLS. It is the Washington Systems Center's (WSC) opinion that enabling TLS connection and other required security features is best done using SAF security. The WSC automatically adds these features when enabling SAF and TLS security.

The *monitor-1.0* adds standard monitoring MBeans as well as MXBeans, the latter provide monitoring for specific runtime components. For more information about the *monitor-1.0* feature including the MXBeans added by this feature, see URL https://www.ibm.com/docs/en/was-liberty/zos?topic=environment-monitoring-monitor-10. The restConnector-2.0 feature adds remote REST client access to a set of administrative APIs over HTTP. For more information about the *restConnector-2.0* feature, see URL https://www.ibm.com/docs/en/was-liberty/zos?topic=features-admin-rest-connector-20.

## Define and permit access to required SAF resources

Access to Liberty MBeans and REST administrative APIs is controlled by access to SAF EJBRole resources, see URL https://www.ibm.com/docs/en/was-liberty/zos?topic=liberty-mapping-management-roles-zos for more information.

Below are examples of the commands used to define and permit access to the *Administrator* EJBRoles.

```
RDEFINE EJBROLE BBGZDFLT*.com.ibm.ws.management.security.resource.Administrator OWNER(SYS1)UACC(NONE)
PERMIT BBGZDFLT*.com.ibm.ws.management.security.resource.Administrator CLASS(EJBROLE) ID(ADMNUSRS) ACCESS(READ)

SETR RACLIST(EJBROLE) REFRESH
```

Using the above RACF resources, only members of the group *ADMNUSRS* will have full access to the JMX resources and MBeans (REST GET, POST, PUT and DELETE).

```
    <safCredentials unauthenticatedUser="WSGUEST"
       suppressAuthFailureMessages="false" profilePrefix="BBGZDFLT" />
```

*Note: the value for *BBGZDFLT* used in the commands was derived from the value of the *profilePrefix* attribute used in the <*safCredentials/*> configuration element.


Mitch Johnson (mitchj@us.ibm.com)

# Managing Application Deployments using Liberty MBeans

The setup required to enable JMX and security are covered in detail in the Washington System Center document *Monitoring a z/OS Liberty server using JMX and REST clients* available at URL https://ibm.biz/BdahXK. Authorizing the REST client to RACF EJBRoles and using the HTTPS protocol from REST clients are absolute requirements for accessing MBeans using JMX Connector.

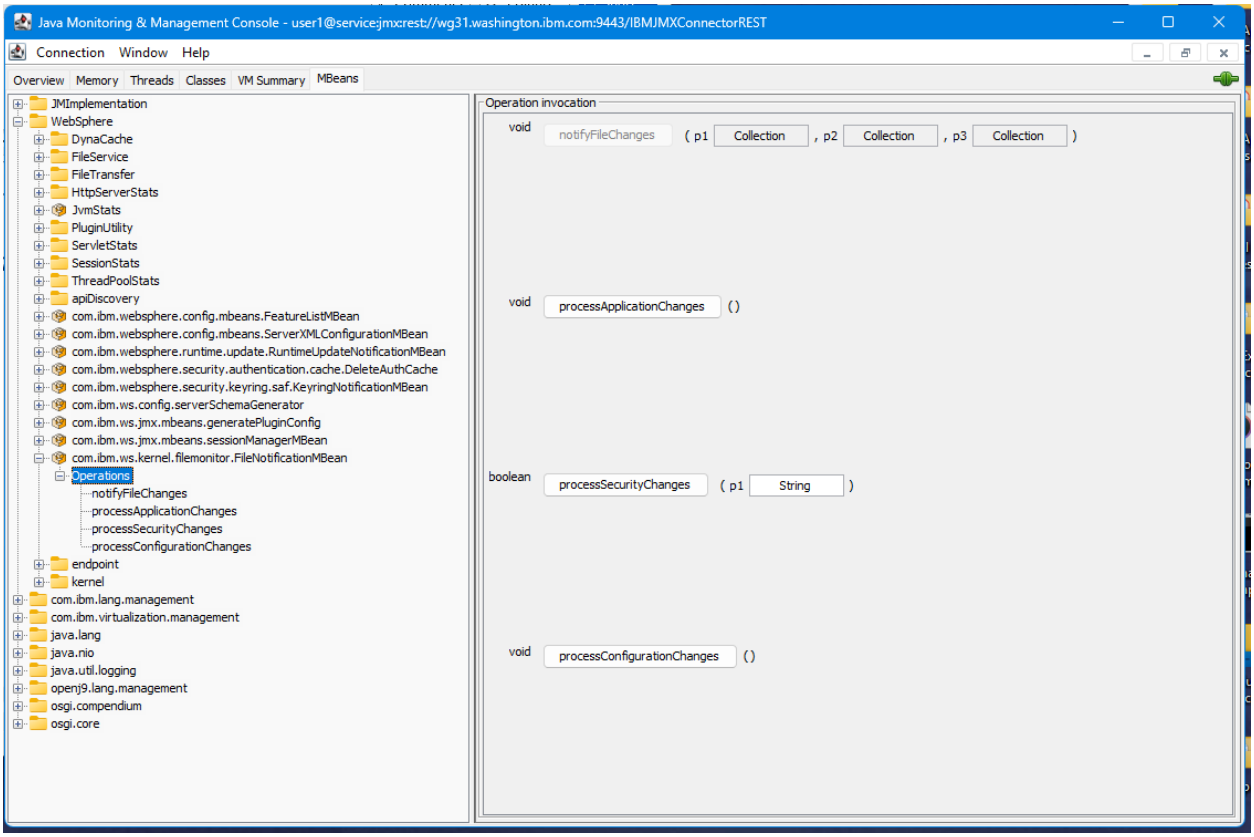
## Exploring the Liberty server's MBeans using JConsole

Most if not all Java Developers Kit (JDK) or Java Runtime Environment (JRE) instances include a Java application known as the Java Console(JConsole). JConsole is a Java GUI client which uses JMX and REST to manage and monitor local and remote JVMs. For details on JConsole, see URL https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/tooldescr009.html

Using the JConsole is useful for understanding functions provided by the MBeans but aunderstand that the information displayed for invoking a MBean is targeted for using the RMI support provided by IBM JMX Connector.

## The FileNotification MBean

The first MBean we will explore is the *FileNotification* MBean. This MBean can be used to notify the server that a file has been created, modified or deleted from the server's configuration directory paths, to inform the server to apply changes in the configuration or to apply application changes.

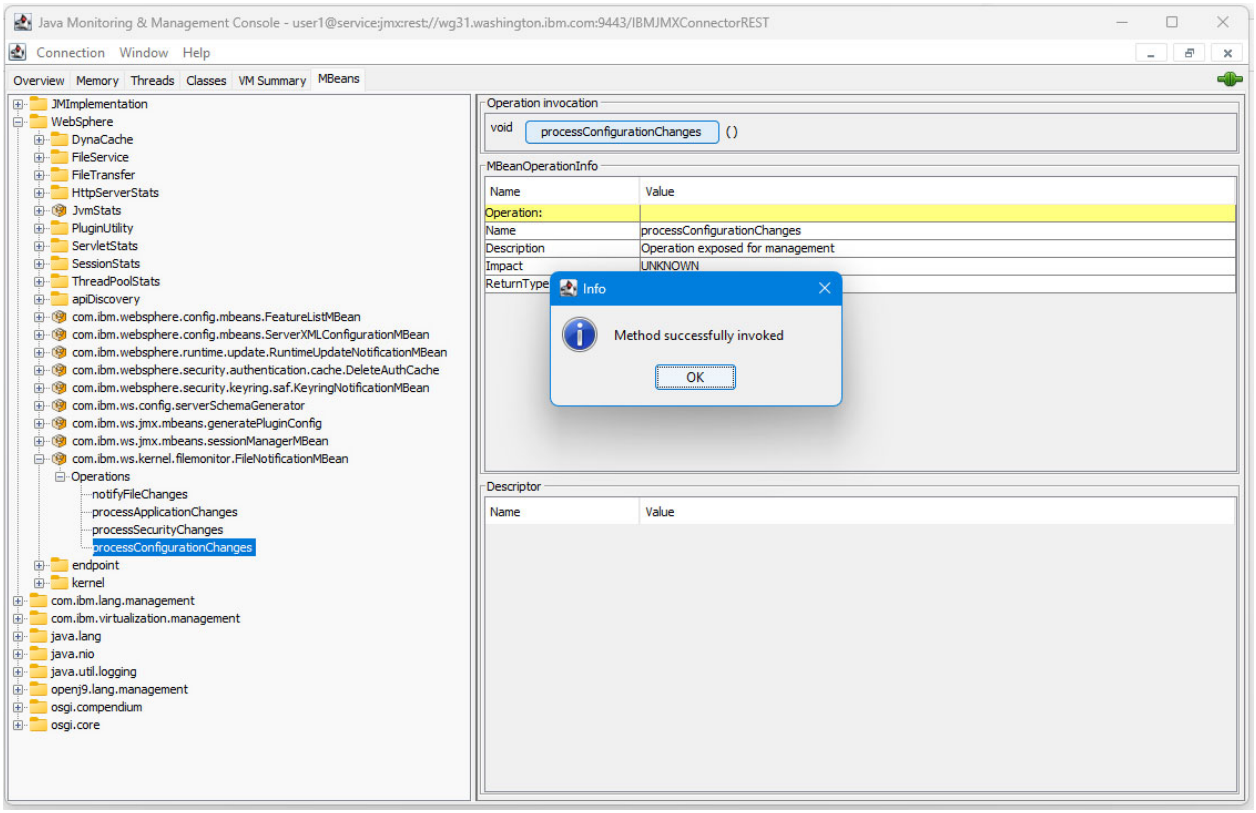




Mitch Johnson (mitchj@us.ibm.com)

## Operation processConfigurationChanges

Expanding a MBean's *Operations* displays a list of functions that this MBean can perform. For example, selecting the *Operations* for MBean *com.ibm.ws.kernal.filemonitor.FileNotificationMBean*, displays the operations shown below.




Selecting operation *processConfigurationChange*s and then clicking on the *processConfigurationChange*s method on the *Operation invocation* section returns the results above.


Mitch Johnson (mitchj@us.ibm.com)

It is important to note that if there are not any actual changes in the contents or to the last modified date of any configuration files then there is no obvious indication that the configuration has been refreshed. To confirm the configuration was being refreshed, the OMVS *touch* command was used to change the *last changed date* of the *server.xml* file and the *processConfigurationChanges* operation was repeated. When this was done these messages appeared in the spool output of the server as well as the *messages.log* file.

```
CWWKG0016I: Starting server configuration update.
CWWKG0093A: Processing configuration drop-ins resource:
/var/zosconnect/servers/myServer/configDropins/defaults/groupAccess.xml
CWWKG0028A: Processing included configuration resource:
/var/zosconnect/shared/config/safSecurity.xml
CWWKG0028A: Processing included configuration resource: /var/zosconnect/shared/config/ipic.xml
CWWKG0028A: Processing included configuration resource:
/var/zosconnect/shared/config/keyring.xml
CWWKG0028A: Processing included configuration resource:
/var/zosconnect/shared/config/features.xml
CWWKG0028A: Processing included configuration resource:
/var/zosconnect/shared/config/apiRequesterHTTPS.xml
CWWKG0028A: Processing included configuration resource:
/var/zosconnect/shared/config/safTrace.xml
CWWKG0028A: Processing included configuration resource: /var/zosconnect/shared/config/db2.xml
CWWKG0028A: Processing included configuration resource:
/var/zosconnect/shared/config/shared.xml
CWWKG0093A: Processing configuration drop-ins resource:
/var/zosconnect/servers/myServer/configDropins/overrides/cscvinc.xml
CWWKG0093A: Processing configuration drop-ins resource:
/var/zosconnect/servers/myServer/configDropins/overrides/db2employee.xml
CWWKG0093A: Processing configuration drop-ins resource:
/var/zosconnect/servers/myServer/configDropins/overrides/insertEmployee.xml
CWWKG0093A: Processing configuration drop-ins resource:
/var/zosconnect/servers/myServer/configDropins/overrides/miniloancics.xml
CWWKG0093A: Processing configuration drop-ins resource:
/var/zosconnect/servers/myServer/configDropins/overrides/phonebook.xml
CWWKG0018I: The server configuration was not updated. No functional changes were detected
```

The *CWWKG0018I* message was displayed because after all the XML has been parsed, there were really no changes in the configuration.

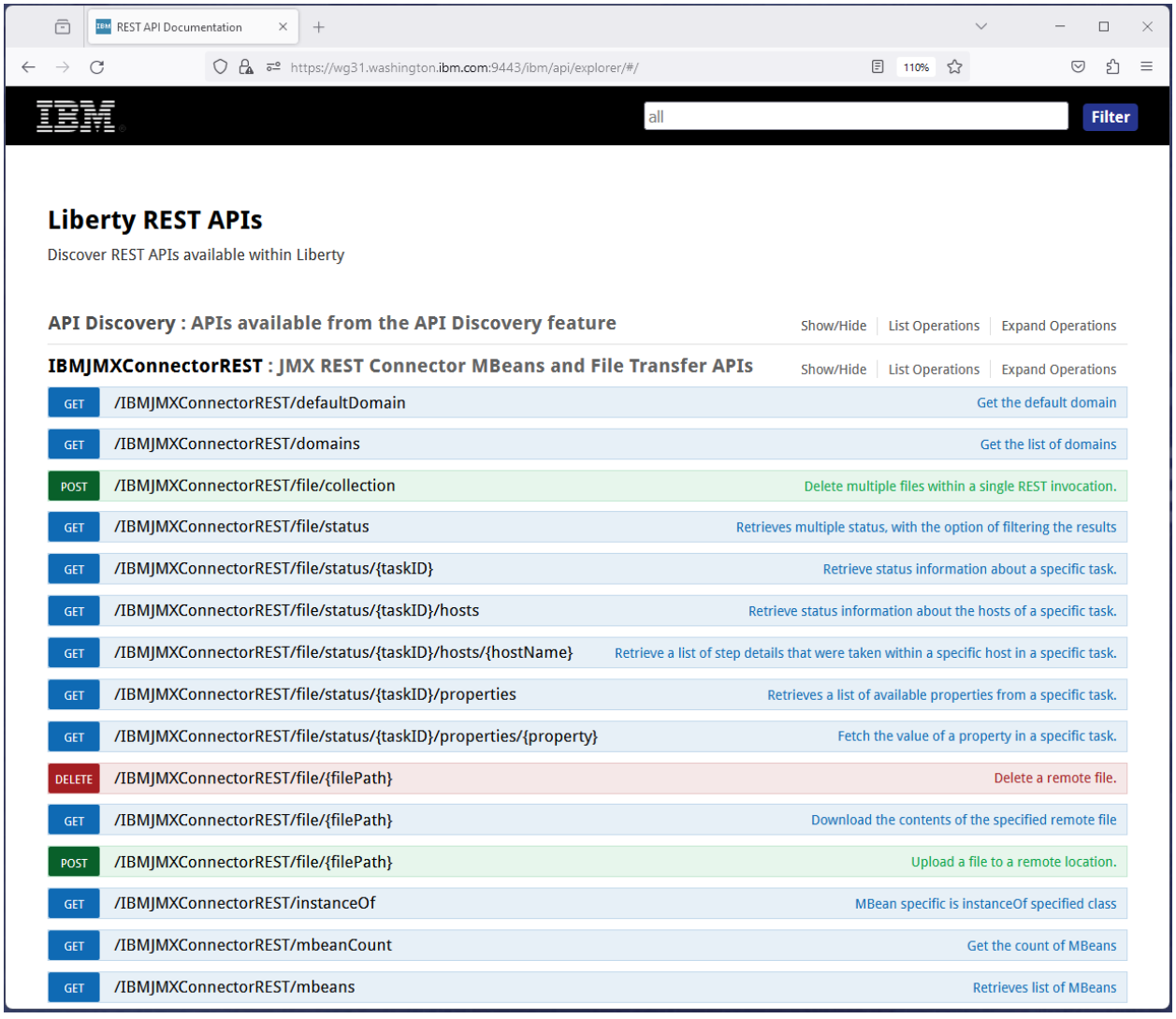
Using JConsole is interesting but not very practical for anything other than exploring how an MBean works. Administrators need to be able to access these MBean in command files, command scripts, etc. Next let's explore using the IBM API Explorer to review invoking a MBean using the JMX REST Connector


Mitch Johnson (mitchj@us.ibm.com)

## Using the IBM API Explorer to access MBeans using REST

Now access the Liberty MBeans from a REST client. Once such client is the *IBM API Explorer* which was added to the Liberty server by adding feature *apiDiscovery-1.0*. This enabled REST access to my server from a web browser.

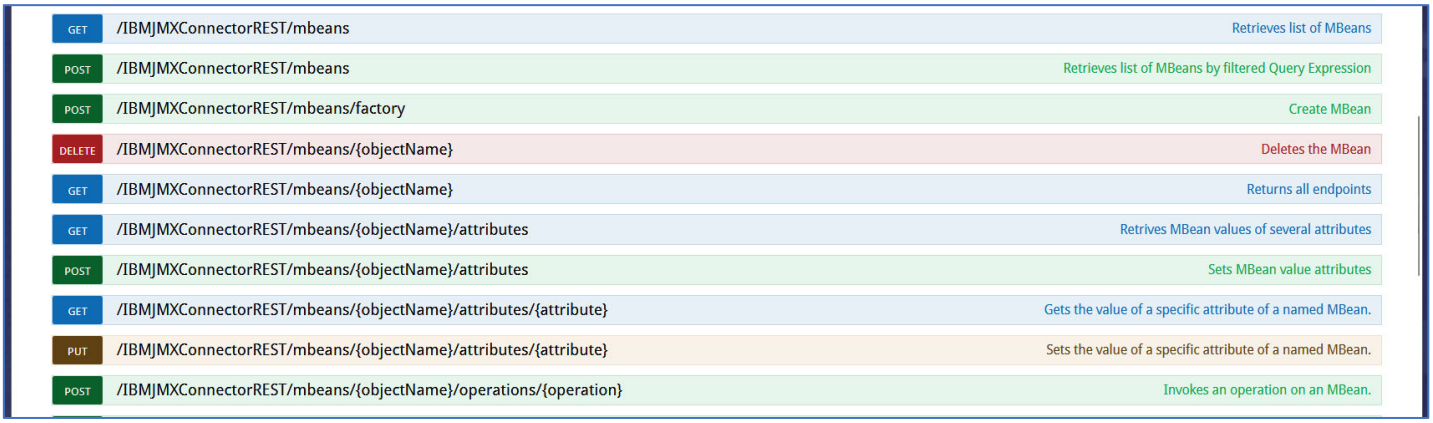
Begin by accessing the server's *IBM API Explorer* web page by using URL, e.g., https://wg31.washington.ibm.com:9443/ibm/api/explorer. Use the *List Operation* option to list the operations in **IBMJMXConnectorREST:JMX REST Connector MBeans and File Transfer APIs**.





Mitch Johnson (mitchj@us.ibm.com)

Focus on these selections when using *the IBM API Explorer*.

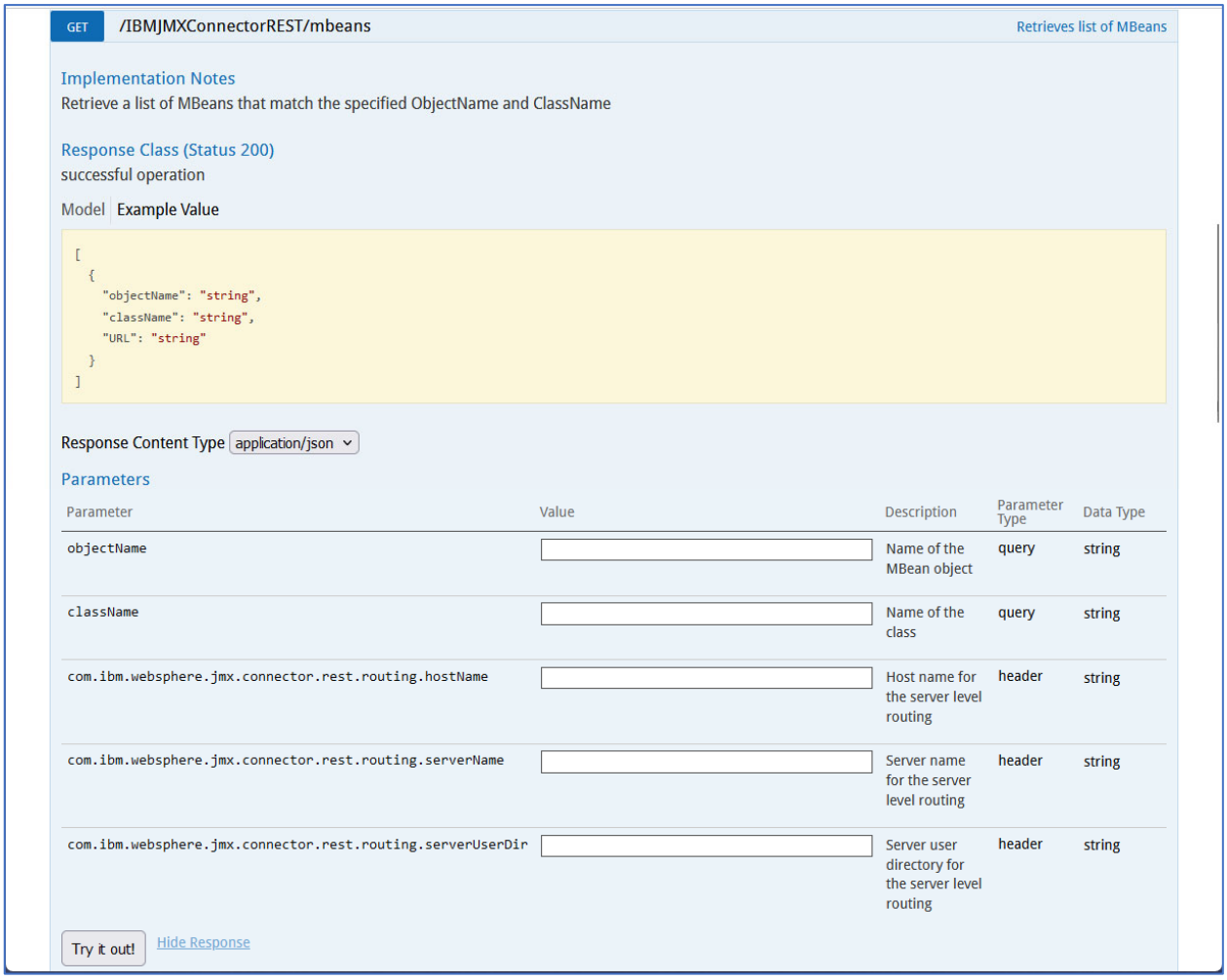



As you see there are REST APIs provided for obtaining a list of MBeans (GET). REST APIs for getting(GET) and setting(POST) an MBean's attributes. And a REST API for invoking (POST) and MBean's operation. The other REST APIs are interesting but beyond scope of the document.


Mitch Johnson (mitchj@us.ibm.com)

# Managing Application Deployments using Liberty MBeans

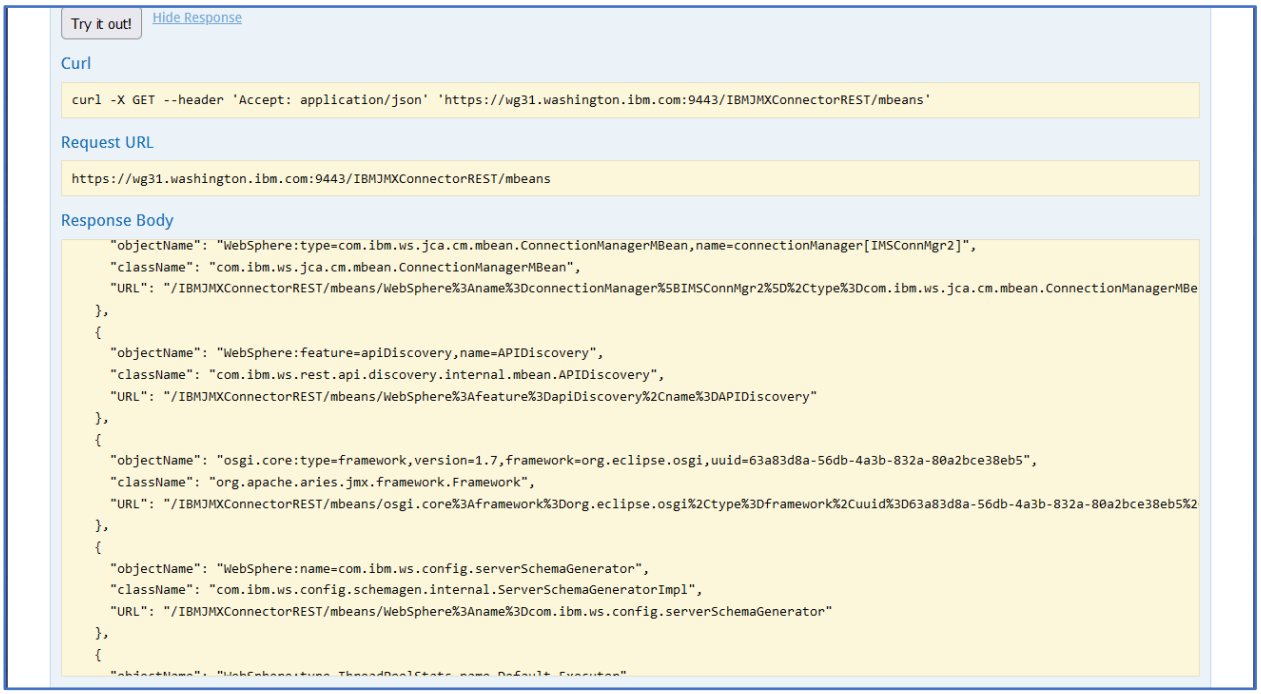
First, display a list of the available MBeans. Select the *GET* method beside */IBMJMXConnectorREST/mbeans* to expose the interface as shown below. Note that you could have display a specific MBean by entering the MBean's object name or class name. But now we want to display them all so leave both fields empty.





Mitch Johnson (mitchj@us.ibm.com)

Pressing the **Try it out!** button will invoke this MBean and display the results in the *Response Body* section.





Mitch Johnson (mitchj@us.ibm.com)

## Accessing the FileNotificationMBean

Now invoke the Mbean that provided performs configuration changes. Using the JConsole interface we learned that the MBean that does this is the *FileNotificationMBean*. So, in the *Response Body* area, search for all occurrences of the string *FileNotificationMBean*. The search found details of a *FileNotificationMBean* , see below. This provides the details(*objectName, className*, and *URL* (*URIPath*)) for this MBean. The MBean's *objectName* is needed to be able to invoke this MBean from the *IBM API Explorer*. The value for the URL(*URIPath*) will be useful when accessing this MBean from a REST client.

Curl

```
curl -X GET --header 'Accept: application/json' 'https://wg31.washington.ibm.com:9443/IBMJMXConnectorREST/mbeans'
```

Request URL

```
https://wg31.washington.ibm.com:9443/IBMJMXConnectorREST/mbeans
```

Response Body


```
    "className": "com.ibm.ws.monitors.helper.JvmStats",
    "URL": "/IBMJMXConnectorREST/mbeans/WebSphere%3Atype%3DJvmStats"
  },
  {
    "objectName": "java.lang:type=Memory",
    "className": "com.ibm.lang.management.internal.ExtendedMemoryMXBeanImpl",
    "URL": "/IBMJMXConnectorREST/mbeans/java.lang%3Atype%3DMemory"
  },
  {
    "objectName": "WebSphere:service=com.ibm.ws.kernel.filemonitor.FileNotificationMBean",
    "className": "com.ibm.ws.kernel.filemonitor.internal.FileNotificationImpl",
    "URL": "/IBMJMXConnectorREST/mbeans/WebSphere%3Aservice%3Dcom.ibm.ws.kernel.filemonitor.FileNotificationMBean"
  },
  {
    "objectName": "java.lang:type=Compilation",
    "className": "com.ibm.java.lang.management.internal.CompilationMXBeanImpl",
    "URL": "/IBMJMXConnectorREST/mbeans/java.lang%3Atype%3DCompilation"
  },
  {
    "objectName": "java.lang:type=GarbageCollector,name=scavenge",
    "className": "com.ibm.lang.management.internal.ExtendedGarbageCollectorMXBeanImpl"
```

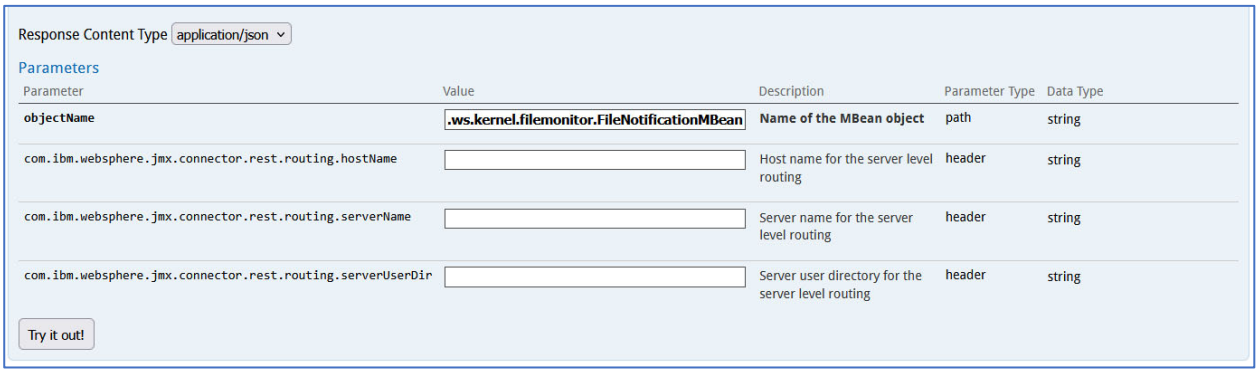

Response Code

```
200
```

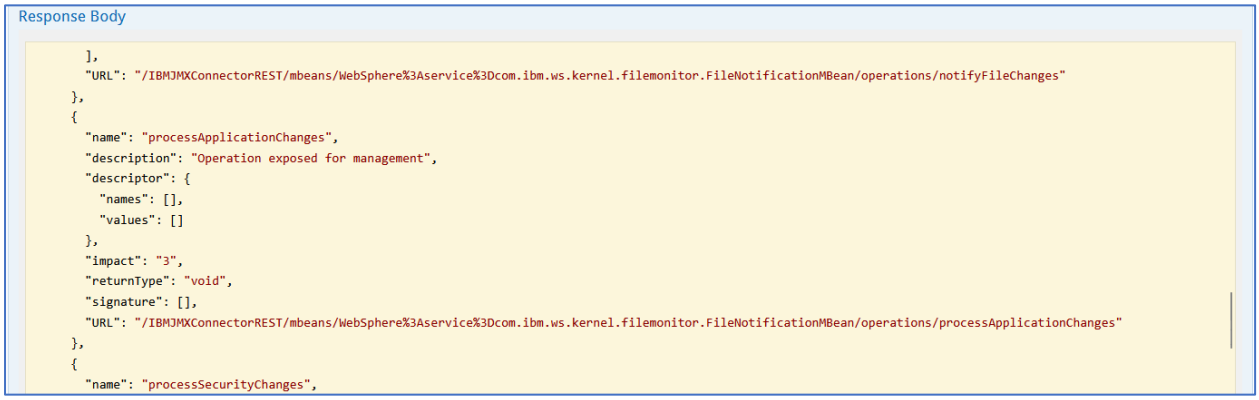

Mitch Johnson (mitchj@us.ibm.com)

# Managing Application Deployments using Liberty MBeans

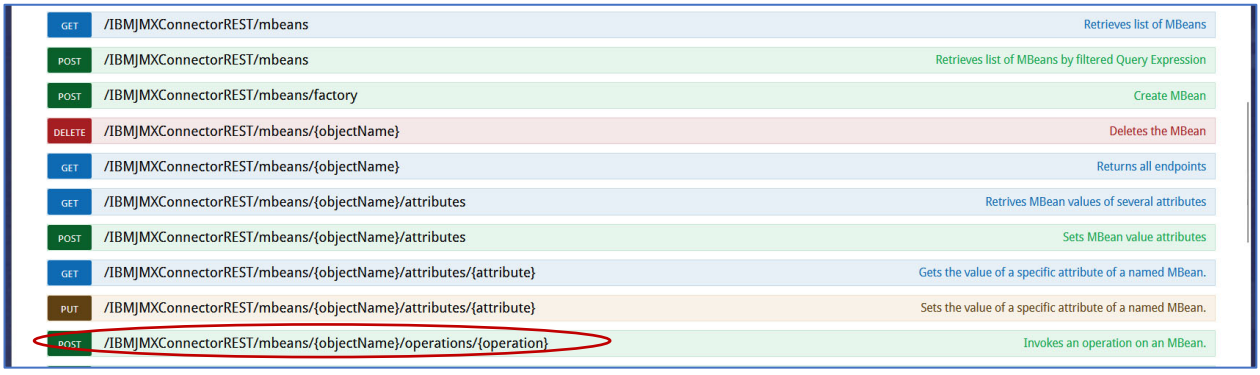
Next to obtain more details about this MBean, e.g., operations, attributes, etc. locate the *GET* method for Mbean */IBMJMXConnectorREST/mbeans/{objectName)* to expose this Mbean's interface. Enter the *objectName* for the FileNotificationMBean, *WebSphere:service=com.ibm.ws.kernel.filemonitor.FileNotificationMBean*




Pressing the **Try it Out** button provided the results below in the *Response Body*. Scrolling down, locate the *processApplicationChanges* operation (that was invoked using JConsole), as well as the URL (URIPath) for use when accessing this operation from a REST client.




To invoke an operation of an MBean requires the use of a POST to URI path for MBean */IBMJMXConnectorREST/mbeans/{objectName}/operations/{operations}*

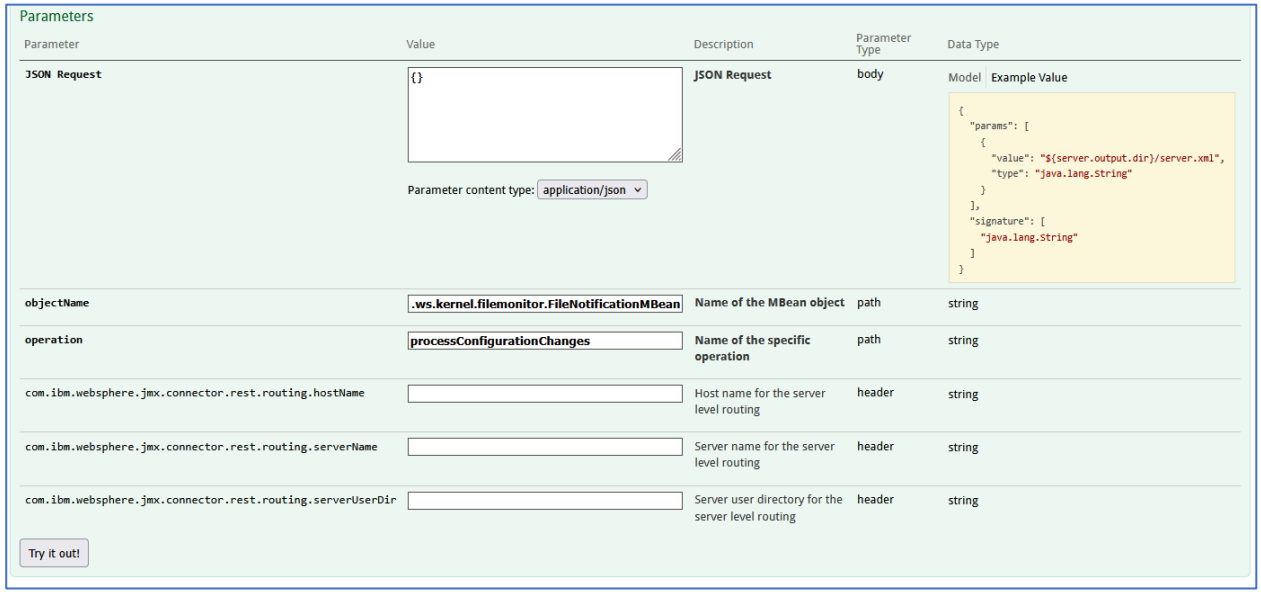




Mitch Johnson (mitchj@us.ibm.com)

## Operation processConfigurationChanges

Now locate the POST method for this Mbean and expose this Mbean's interface. Enter the MBean's object name, *WebSphere:service=com.ibm.ws.kernel.filemonitor.FileNotificationMBean* along with the name of the operations, *processConfigurationChanges,* in the areas beside *objectName* and *operations* respectively. Since this was a POST request with a null request message, a simple JSON message consisting of a beginning and ending braces, *{}*, is sufficient in the *JSON Request* area.





Mitch Johnson (mitchj@us.ibm.com)

Pressing the **Try it Out** button invoked this operation provided the results showing a 200 HTTP response code as well as the expected messages in the server's spool output and messages.log file.


Again, if there has been no change in the configuration files, then no messages are displayed in the spool or messages file.

We can take the *curl* command generated (see above) and use it as a basis for invoking the same MBean operation in a command line environment. Below is an example of invoking the *processConfigurationChanges* using the cURL command line tool.

```
curl -X POST --user USER1:user1 --header "Content-Type: application/json" -d "{}"
https://wg31.washington.ibm.com:9443/IBMJMXConnectorREST/mbeans/WebSphere%3Aservice%3Dcom.ibm.ws.kernel.filemonitor.FileNotificationMBean/operations/processConfigurationChanges --insecure -w " -HTTP CODE: %{http_code}"
```

In the above command basic authentication,***--user USER1:user1***, was used to provide the identity that is to be used to check the authorization to invoke this MBean (TLS mutual authentication security could have just as easily enabled. The **–insecure** flag was used to implicitly trust the server's certificate being provide for the TLS handshake. And finally, included was ***"-w -HTTP CODE: %{http_code}"*** to display the HTTP return code from the request.


Mitch Johnson (mitchj@us.ibm.com)

When the cURL command was invoked, the results below were obtained. **{"value":null,"type":null}** was the response message from the MBean and the HTTP code was 200 (success) along with the expected messages in the server's spool output and *messages.log* file.

```
C:\z\mbean>curl -X POST --user USER1:user1 --header "Content-Type: application/json" -d "{}"
https://wg31.washington.ibm.com:9443/IBMJMXConnectorREST/mbeans/WebSphere%3Aservice%3Dcom.ibm.ws
.kernel.filemonitor.FileNotificationMBean/operations/processConfigurationChanges --insecure -w "
-HTTP CODE: %{http_code}"

{"value":null,"type":null} -HTTP CODE: 200

C:\z\mbean>
```

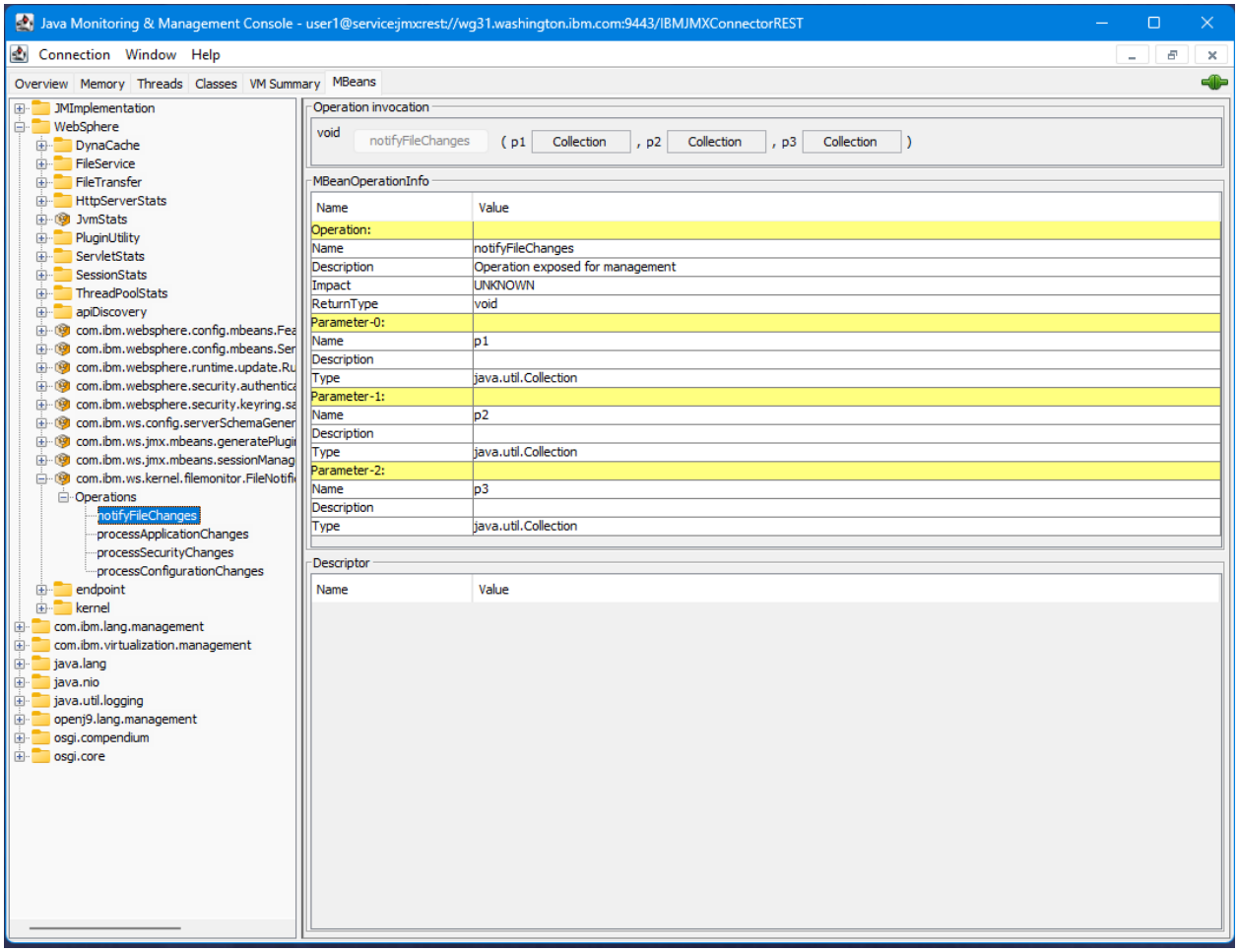
There may be sometimes when basic security is not an accepted option. In these situations, TLS mutual authentication can easily be used. All that is required is access to two local files. In the example cURL command below, file *certauth.pem* is the public certificate provided by the certificate authority used to the sign the server certificate used by the server endpoint. File *johnson.p12* is a file containing the local user's personal certificate. Since this certificate has a private key, a password (*secret*) is required to access this personal certificate. The personal certificate subject's and issuer's distinguished names will need to be mapped to RACF identity using digital certificate mapping for digital certificate filtering.

```
curl -X POST --cacert certauth.pem --cert johnson.p12:secret --cert-type P12
--header "Content-Type: application/json" -d "{}"
https://wg31.washington.ibm.com:9443/IBMJMXConnectorREST/mbeans/WebSphere%3Aservi
ce%3Dcom.ibm.ws.kernel.filemonitor.FileNotificationMBean/operations/processConfig
urationChanges -w " -HTTP CODE: %{http_code}"
```


Mitch Johnson (mitchj@us.ibm.com)

## Operation notifyFileChanges

Using JConsole we learned that this same MBean had an operation named *notifyFileChanges* . Using JConsole to display the results below shows the arguments when invoking the operation using RMI from a Java client.




This operation has three arguments. Each argument is a collection or list of absolute file names where each entry in the collection is a full file path and file path. The first argument is a collection of the files that are newly created. The second argument is a collection of files that have been updated or modified. And the third argument is a collection of files that have been deleted. Each collection must be provided when invoking this operation but it is allowed for one or more of the collections to be null or empty.

Using this information along with the example JSON request message from display this MBean in the IBM API Explorer we can build the required JSON message for invoking this operation from a REST client.



Mitch Johnson (mitchj@us.ibm.com)

Here is an example of providing these arguments in a JSON request message. In this example the collection of created files is null or empty as well as the collection of deleted files. The collection of updated files contains only one entry, */var/zcee/shared/apis/cscvinc.aar.*

```
{
   "params": [
      {
         "value": [""],
         "type": {"className": "java.util.ArrayList",
            "items": ["java.lang.String"]
         }
      },
      {
         "value": ["/var/zcee/shared/apis/cscvinc.aar"],
         "type": {"className": "java.util.ArrayList",
            "items": ["java.lang.String"]
         }
      },
      {
         "value": [""],
         "type": {"className": "java.util.ArrayList",
            "items": ["java.lang.String"]
         }
      }
   ],
   "signature": [
      "java.util.Collection",
      "java.util.Collection",
      "java.util.Collection"]
}
```

Providing this JSON request message in file *zceeRefresh.json* as input simplified the cURL command as shown below.

```
curl -X POST --insecure --user USER1:user1
--header "Content-Type: application/json" -d @c:/z/mbean/zceeRefresh.json
https://wg31.washington.ibm.com:9443/IBMJMXConnectorREST/mbeans/WebSphere%3Aservice%3Dcom.ibm.ws.kernel.filemonitor.FileNotificationMBean/operations/notifyFileChanges
-w " -HTTP CODE: %{http_code}"
```


Mitch Johnson (mitchj@us.ibm.com)

Invoking this command resulted in the messages below appearing in the server's spool output and the *messages.log* file. Invoking this operation is equivalent to the MVS MODIFY command *F BASTRT,ZCON,REFRESH* on a z/OS Connect OpenAPI2 server.

```
BAQR7132I: z/OS Connect EE API cscvinc was unregistered successfully for API Discovery.
BAQR7130I: z/OS Connect EE API cscvinc was registered successfully for API Discovery.
BAQR7034I: z/OS Connect EE API archive file cscvinc updated successfully.
```

Again, if there are not any actual changes in the contents or the last modified date of any application artifact then there is no obvious indication that the application has been refreshed. To confirm the configuration was being refreshed, the OMVS *touch* command was used to change the *last changed date* of the files. When the *notifyFileChanges* operation was invoked again, these messages appeared in the spool output of the server.

Given that these arguments are *collection*, more than one file can be provided. But this requires that the list of *items* for a *collection* provide the Java type of the additional entry, see the bold-faced additions below in the updated JSON request message.

```
{
  "params": [
    {
      "value": [""],
      "type": {"className": "java.util.ArrayList",
        "items": ["java.lang.String"]
      }
    },
    {
      "value": ["/var/zcee/shared/apis/cscvinc.aar","/var/zcee/shared/services/cscvincSelectService.sar"],
      "type": {"className": "java.util.ArrayList",
        "items": ["java.lang.String","java.lang.String"]
      }
    },
    {
      "value": [""],
      "type": {"className": "java.util.ArrayList",
        "items": ["java.lang.String"]
      }
    }
  ],
  "signature": [
    "java.util.Collection",
    "java.util.Collection",
    "java.util.Collection"]
}
```

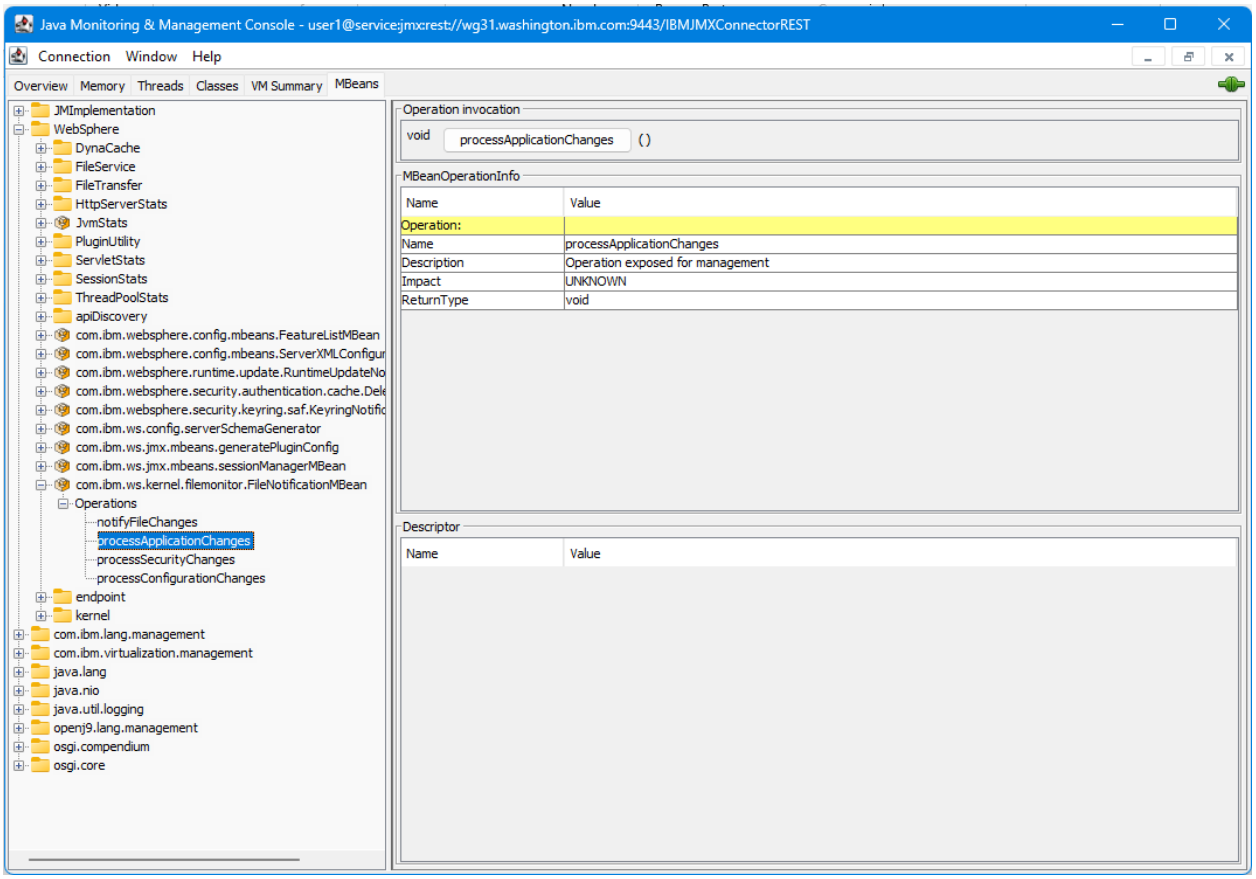

Mitch Johnson (mitchj@us.ibm.com)

Invoking the same cURL command with the updated JSON resulted and after “touching” the files in the OMVS file system resulted in both the service and API being updated.

```
BAQR7044I: z/OS Connect EE service archive cscvincSelectService updated successfully.
BAQR7132I: z/OS Connect EE API cscvinc was unregistered successfully for API Discovery.
BAQR7130I: z/OS Connect EE API cscvinc was registered successfully for API Discovery.
BAQR7034I: z/OS Connect EE API archive file cscvinc updated successfully.
```

## Operation processApplicationChanges

Back in JConsole we learned that by selecting operation *processApplicationChange*s that this operation had a null signature and that a null JSON request messages ({}) would be sufficient.





Mitch Johnson (mitchj@us.ibm.com)

From our earlier experience with operation *processConfigurationChanges* we can easily derive the format of the curl to invoke this operation. See the example below:

```
curl -X POST --user USER1:user1 --header "Content-Type: application/json" -d "{}"
https://wg31.washington.ibm.com:9455/IBMJMXConnectorREST/mbeans/WebSphere%3Aservice%3Dcom.ibm.ws.kernel.filemonitor.FileNotificationMBean/operations/processApplicationChanges --insecure -w " -HTTP CODE: %{http_code}"
```

In the above command basic authentication,***--user USER1:user1***, was used to provide the identity that is to be used to check the authorization to invoke this MBean (TLS mutual authentication security could have just as easily enabled. The **–insecure** flag was used to implicitly trust the server's certificate being provide for the TLS handshake. And finally, included was ***"-w -HTTP CODE: %{http_code}"*** to display the HTTP return code from the request.

When the cURL command was invoked, the results below were obtained.
**{"value":null,"type":null}** was the response message from the MBean and the HTTP code was 200 (success) along with the expected messages in the server's spool output.

```
C:\z\mbean>curl -X POST --user USER1:user1 --header "Content-Type: application/json" -d "{}"
https://wg31.washington.ibm.com:9455/IBMJMXConnectorREST/mbeans/WebSphere%3Aservice%3Dcom.ibm.ws.kernel.filemonitor.FileNotificationMBean/operations/processApplicationChanges --insecure -w " -HTTP CODE: %{http_code}"

{"value":null,"type":null} -HTTP CODE: 200

C:\z\mbean>
```

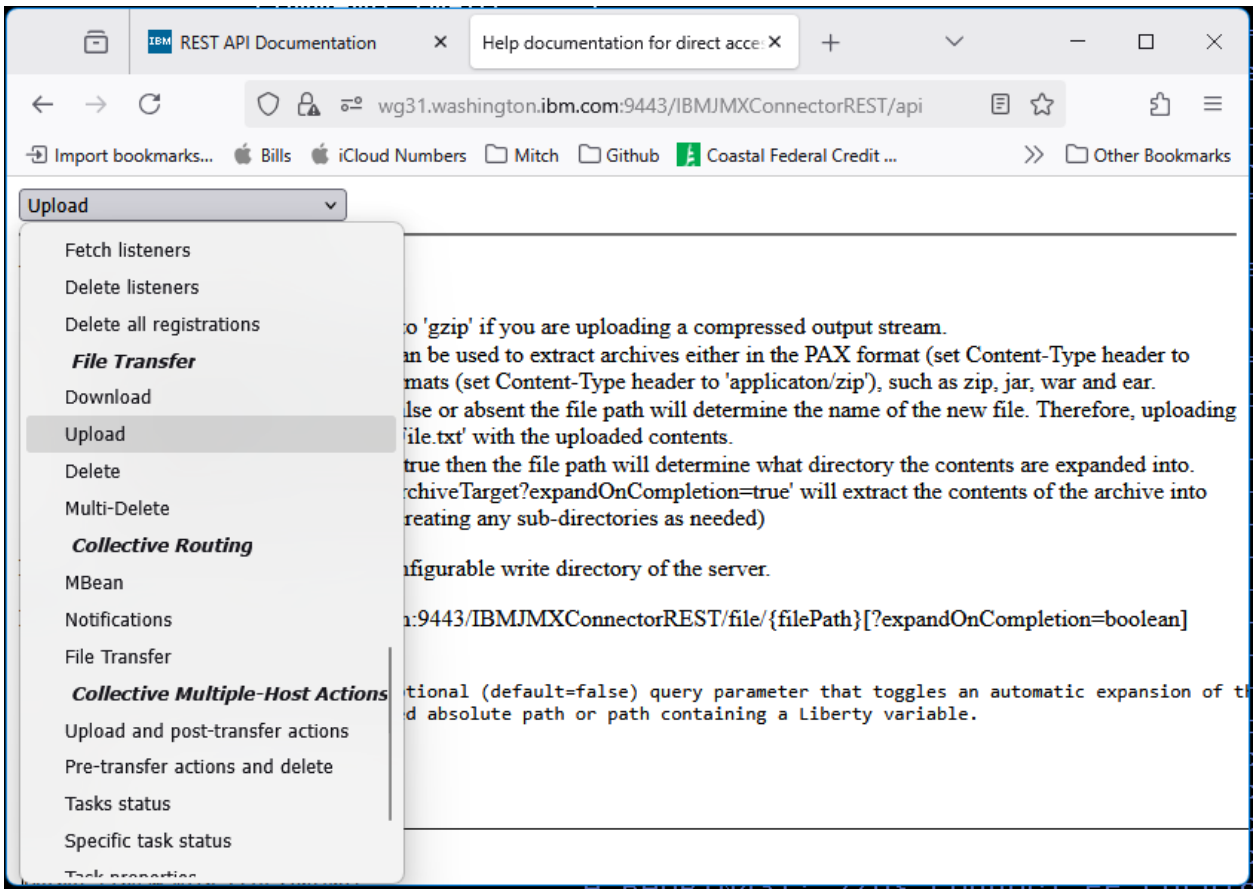
Invoking this operation is equivalent to invoking the MVS MODIFY command *F BAQSTRT,REFRESH,APPS* on a server.


Mitch Johnson (mitchj@us.ibm.com)

## The FileTransfer MBean

The next MBean to be explored is the *FileTransfer* MBean. This MBean can be used to download an existing file, or to delete an existing file or to upload a new file using RESTful request using the IBM JMX REST Connector. RESTful in the respect that a POST uploads a new file to an OMVS file system directory, GET downloads a file from a directory and DELETE removes or deletes a file from a directory.

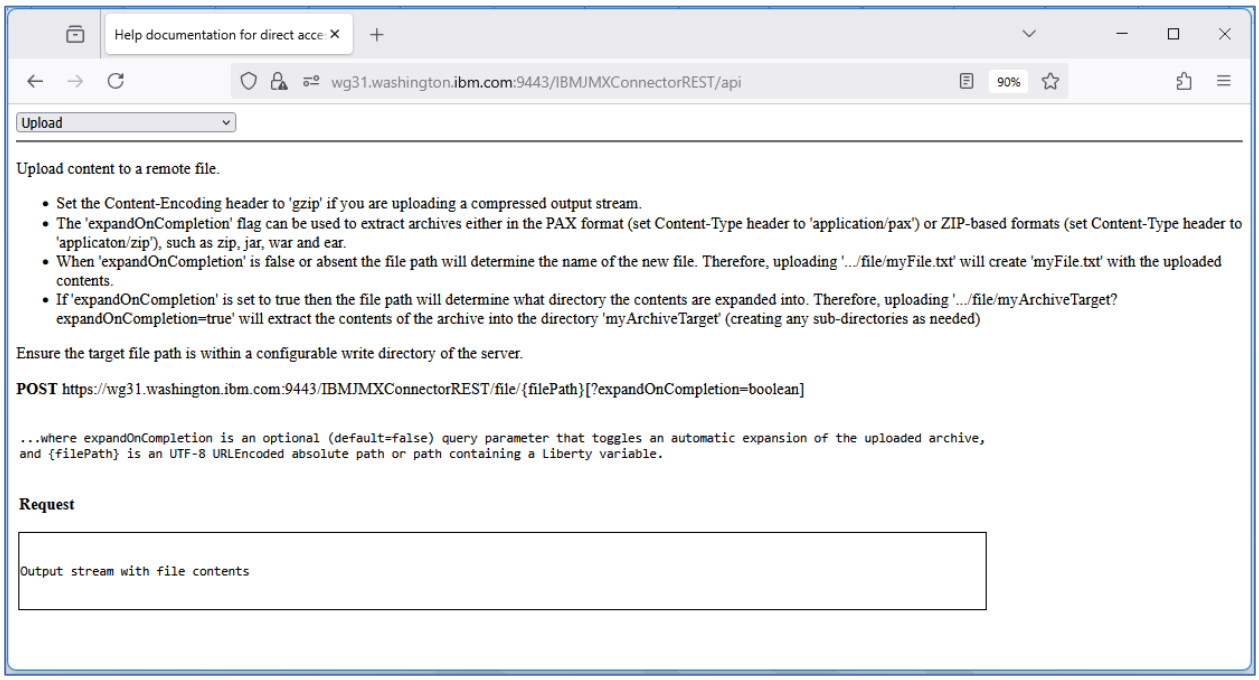
We obtained the details invoking the operations of the MBean by going to URI path */IBMJMXConnectorREST/api* in our server and using the drop down to locate information about the File Transfer operations.





Mitch Johnson (mitchj@us.ibm.com)

Selecting *Upload* displays this information.




N.B. To provide access from the server to the underlying OMVS file system a *<remoteFileAccess/>* configuration element is needed. This configuration element identifies which the directories are to be exposed to a remote client and once exposed whether they are available for reading and/or writing, something like what is shown below.

```
<remoteFileAccess>
   <writeDir>${server.config.dir}</writeDir>
   <writeDir>/var/zcee/ats/config/apis</writeDir>
   <writeDir>/var/zcee/ats/config/services</writeDir>
   <readDir>/var/zosconnect</readDir>
</remoteFileAccess>
```


Mitch Johnson (mitchj@us.ibm.com)

## Uploading a file

Based on the documentation obtained as above, uploading a file to a server requires using a *POST* method with an URI path of */IBMJMXConnectorREST/file/*. There is one required path parameter which is the full or absolute path and name of the target file. The path name must be relative to the root directory so the leading slash (/) is required. This is why double slashes appear in the URI path. One for the end of the context root and another for the beginning of the path name. The file to be upload is provided as is with conversion in the request message using the *–data-binary* parameter. See the example cURL command below.

```
curl -X POST --user user1:user1 --insecure --location
https://wg31.washington.ibm.com:9443/IBMJMXConnectorREST/file/var/zosconnect/se
rvers/myServer/apps/roster.war --header "Content-Type: application/zip"
--data-binary @C:\z\openapi3\wars\roster.war -w " -HTTP CODE: %{http_code}"
```

In the above command the source file is **C:\z\openapi3\wars\roster.war** and the target path and file name are provided by **/var/zosconnect/servers/myServer/apps/roster.war**, be sure to include the trailing slash (/) of the URI path. The **–data-binary** ensures the file is uploaded in binary mode. And finally, the option *expandOnCompletion* was allowed to default to false to disable the automatic expansion of the file.

When the cURL command was invoked, the results below were obtained.
**{"value":null,"type":null}** was the response message from the MBean and the HTTP code was 204 (success with no response message) along with the expected messages in the server's spool output and *messages.log* file.

## Deleting a file

Based on the documentation above, uploading a file to a server requires using a *DELETE* method with an URI path of */IBMJMXConnectorREST/file/*. See the example cURL command below.

```
curl -X DELETE --user user1:user1 --insecure --location
https://wg31.washington.ibm.com:9443/IBMJMXConnectorREST/file/var/zosconnect/se
rvers/myServer/apps/roster.war --header "Content-Type: application/zip"
--data-binary @C:\z\openapi3\wars\roster.war -w " -HTTP CODE: %{http_code}"
```

In the above command the file to be deleted is
**/var/zosconnect/servers/myServer/apps/roster.war**, be sure to include the trailing slash (/) of the URI path.

When the cURL command was invoked, the results below were obtained.
**{"value":null,"type":null}** was the response message from the MBean and the HTTP code was



Mitch Johnson (mitchj@us.ibm.com)

204 (success with no response message) along with the expected messages in the server's spool output and *messages.log* file.

## Downloading a file

Based on the documentation we found, downloading file to a server requires using a *GET* method with an URI path of */IBMJMXConnectorREST/file/* with a Path parameter of the full path and name of the file to be downloaded. See the example cURL command below.

```
curl -GET --user user1:user1 --insecure --location
https://wg31.washington.ibm.com:9443/IBMJMXConnectorREST/file//var/zosconnect/se
rvers/myServer/apps/roster.war --header "Content-Type: application/zip"
--data-binary @C:\z\openapi3\wars\roster.war -w " -HTTP CODE: %{http_code}"
```

In the above command the file to be downloaded is **/var/zosconnect/servers/myServer/apps/roster.war**, be sure to include the trailing slash (/) of the URI path.

When the cURL command was invoked, the results below were obtained.
**{"value":null,"type":null}** was the response message from the MBean and the HTTP code was 204 (success with no response message) along with the expected messages in the server's spool output and *messages.log* file.


Mitch Johnson (mitchj@us.ibm.com)