BigFix Version 10.0.1
REST API

Special notice

Before using this information and the product it supports, read the information in Notices (on page 82).

Contents

Chapter 1. Overview

The BigFix REST API

The REST API is the primary programming interface to the BigFix Server. It allows you to perform the majority of the tasks available in the BigFix Console by using a set of standardized and operating system independent methods. This API is also key if you want to automate activities, implement your custom BigFix user interface, or integrate with other applications.

Overview

REpresentational **S**tate **T**ransfer is a client-server architecture. BigFix provides you with:

- The REST API server part, available on the BigFix server, that manipulates the objects stored in the BigFix database.
- A lightweight command-line tool named IEM Command-Line Interface (CLI), that you can use as a REST client to initiate requests towards the REST API server.

You can choose to use your preferred REST Client, in place of the IEM CLI, to issue methods and interact with the BigFix REST server through HTML calls.

Chapter 2. Prerequisites

The following conditions must be satisfied to use the BigFix REST API:

1. The WebReports service must be running. If you installed multiple Web Reports servers, local or remote, the Web Reports server that is selected for using the REST API is the first entry that appears in the table dbo.AGGREGATEDBY contained in the database **BFEnterprise**.
2. The user logging into the REST API must be defined as a BigFix Console operator with the Can use REST API and the Custom Content permissions set to **YES** in its definition or in one of the assigned roles.
3. If you plan to use the REST API over HTTPS, you must apply the configuration described in Customizing HTTPS on REST API.
4. The IEM CLI does not support the BigFix FIPS-compliant cryptography library. If you plan to use that library in your REST API HTTP environment, use a different REST API Client.
5. Ensure that the font set in the command prompt properties of your workstation is set to a value different from Raster Fonts, otherwise you might get this warning message "*Warning: Current console font may not display locale characters correctly*".

Chapter 3. User Authentication

The login from the REST API Client to the BigFix REST API server uses basic access authentication.

The credentials are the credentials of a valid BigFix Console operator.

By default, the session times out after 5 minutes of idle time. You can modify this timeout from the BigFix Console as follows:

1. Select the BigFix server in the Computers list and click **Edit Settings**.
2. Set the desired value in minutes in the **_BESDataServer_APIAuthenticationTimeoutMinutes** setting.
3. Restart the BigFix Server to activate the change.

The default HTTP Authentication Realm is:

- *BigFix Server* for BigFix Server V9.2.6 and later
- *Endpoint Manager Server* for BigFix Server with version earlier than 9.2.6.

As a best practice try not to depend on a hard-coded definition of HTTP Authentication Realm.

You can use the header field SessionToken supplied by the server as an authentication token by returning it in the same header until it expires on the server.

Chapter 4. Hints about BigFix REST API requests

The BigFix REST API allows you to:

- Operate on a category, that is a set of objects of the same type defined on a specific site.
- Operate on an item belonging to a category.
- Perform specific activities, for example administrative tasks, upload, login and import.

This is accomplished by means of REST API requests issued from the REST API Client user interface against the REST API Server component.

The way you run these REST API requests depends on the HTTP client that you want to use (for example cURL, python, or IEM Command-Line Interface).

Generally speaking, a REST API request includes:

- An ***HTTP Method*** among those supported by the resource represented by the *URL*.
- An ***XML input file*** if you are running PUT and POST methods. The input file contains the description, in XML format, of the object to create or to update. The XML structures to use are documented in schema files. This is a sample XML input file used to create a new action:

```
<?xml version="1.0" encoding="UTF-8"?>
<BES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BES.xsd">
 <SourcedFixletAction>
   <SourceFixlet>
     <Sitename>TestSite</Sitename>
     <FixletID>83</FixletID>
     <Action>Action1</Action
   </SourceFixlet>
   <Target>
```

```
      <ComputerID>13863357</ComputerID>
    </Target>
   <Parameter Name="_BESClient_EMsg_Detail">1000</Parameter>
  </SourcedFixletAction>
</BES>
```

- An ***URL*** that identifies the resource that you want to manage using the method. Depending on the type of resource that you want to work with, you use structures similar to the following:
  - **/api/*category*[/site type | /site type/site name]** if you want to manipulate a category of resources of the same type. For example this URL represents the category of all computers defined in the master actionsite: `/api/computers`
  - **/api/category[/site type | /site type/site name]/*object id*** if you want to manipulate an object with a specific ID belonging to that category. For example, this URL identifies the computer with ID 13863357 defined in the master actionsite: `/api/computers/13863357`
  - **/api/category[/site type | /site type/site name]/object id/*context specific*** if you want to manipulate a property of an object with a specific ID belonging to that category. For example, this URL identifies the status of the action with ID 44: `/api/action/44/status`

  In REST API Resource you see, for each resource, the structure to use for each supported HTTP Method.

  The notation ***/site type*** or ***/site type/site name*** is used to specify the type and the URL-encoded name of the site where that resource resides. It can be:
  - Omitted if the resource does not relate to a site but is specific to your environment. This applies, for example, to operators, roles, computers, and so on.
  - Set to *master* if you logged in as a non-master operator and the category is defined on the master actionsite.
  - Set to /operator/*operator_name* if the category is defined in the operator site of a specific non-master operator (NMO).
  - Set to /external/*site name* if the category is defined on an external site. For example, `/api/tasks/external/BES Support` identifies the category of tasks available on the BES Support site.

    ◦ Set to /custom/*site name* if the category is defined on a custom site of your BigFix environment.

Notes:

1. GET requests are idempotent because they do not change any underlying resource data. You can do measurements and tracking on retrieved data, but the resource data identified by the URI does not change.
2. The time fields returned in the REST API response are expressed in the GMT time zone. If another time zone is required, the REST API client must convert the time zone.
3. At the following link you find useful information to make REST calls in different languages.

Chapter 5. Quick Tutorial

Follow the instructions provided on this page to get started with using the BigFix REST API. In the scenario that you are going to run you'll see how to query resources, run session relevance queries, create an action, and track the action progress.

The scenario requires that you have:

- ◦ Access to the BigFix Server's REST API.
  ◦ Administration rights for at least one computer.
  ◦ Firefox browser with the client poster plugin.

Log in and request help

Open your browser and access the BigFix server REST API's at the URL:

```
https://<bigfix_server>:52311/api/help
```

where `<bigfix_server>` is the BigFix server hostname or IP address. When prompted, enter your credentials and click Enter. The list of available REST API resources is displayed. Drill down for more details about the action resource by entering the URL:

```
https://<bigfix_server>:52311/api/help/action
```

The help shows you the ways you can interact with that resource, in terms of methods that you can invoke and the syntax to use:

```
GET:
    /api/action/{id}
    /api/action/{id}/status
POST:
    /api/action/{id}/stop
DELETE:
    /api/action/{id}
```

Query a resource

One of the simplest resouces to query is a computer. From your browser, access this URL to see the list of computers that you administer:

```
https://<bigfix_server>:52311/api/computers
```

This list shows, for each computer, the last time that the computer reported information and its ID, for example:

```
<?xml version="1.0" encoding="UTF-8"?>
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="BESAPI.xsd">
    <Computer Resource="api/computer/2785212">
        <LastReportTime>Fri, 07 Nov 2015 15:18:14 +0000</LastReportTime>
        <ID>2785212</ID>
    </Computer>
...
</BESAPI>
```

The XML structure used to list the administered computers is defined in the BESAPI.xsd file, as you can see in the /api/computers reference.

Query a resource using GET

You can get the same result by invoking a query in Session Relevance Language throught the REST API. This is the form to use:

```
https://<bigfix_server>:52311/api/query?relevance={relevance statement}
```

If you do not have a REST API Client available in your environment, install a browser plugin such as Poster for Firefox or RestClient for Firefox. Assuming that you installed the Poster plugin, enter the following URL in Poster and click **GET**.

```
https://<bigfix_server>:52311/api/query?relevance=(ids of it, last report
 time of it)
of bes computers
```

The result that you get is similar to the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
    <Query Resource="(ids of it, last report time of it) of bes computers">
        <Result>
            <Tuple>
                <Answer type="integer">2785212</Answer>
                <Answer type="time">Fri, 07 Nov 2014 15:18:14 +0000</
Answer>
            </Tuple>
...

       <Evaluation>
            <Time>0.114ms</Time>
            <Plurality>Plural</Plurality>
        </Evaluation>
    </Query>
</BESAPI>
```

Create a resource using POST

Now you see how you can create and trigger an action using a POST method. This exercise uses the action definition specified in Fixlet ID 168, which is available on the BES Support site. Fixlet ID 168 adjusts the CPU usage on the BigFix client. Revert the CPU usage to your normal setting after completing this exercise. Enter the following URL in Poster, making sure that you substitute your computer ID in the XML content, and click **POST**.

URL:

```
https://<bigfix_server>:52311/api/actions
```

Content to send:

```
<?xml version="1.0" encoding="UTF-8"?>
<BES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BES.xsd">
  <SourcedFixletAction>
    <SourceFixlet>
      <Sitename>BES Support</Sitename>
        <!-- CPU Usage -->  <FixletID>168</FixletID>
        <!--Medium 5% -->  <Action>Action3</Action>
    </SourceFixlet>
    <Target>
        <!-- set computer id-->    <ComputerID>{id}</ComputerID>
    </Target>
</SourcedFixletAction>
</BES>
```

In return you get the ID of the action that was created and triggered, action ID 42 in this example:

```
<?xml version="1.0" encoding="UTF-8"?>
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
    <Action Resource="http://{server}:52311/api/action/42"
     LastModified="Fri, 21 Nov 2014 16:49:04 +0000">
        <Name>BES Client Setting: CPU Usage</Name>
        <ID>42</ID>
    </Action>
</BESAPI>
```

Take note of this ID to run the next steps in this topic.

Monitor a resource using GET

You can now track the progress of the action.

Enter the following URL in Poster, making sure that you substitute your action ID in the XML content, and click **GET**.

URL:

```
https://server:52311/api/action/{id}/status
```

Content to send:

```
<?xml version="1.0" encoding="UTF-8"?>
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
    <ActionResults Resource="http://{server}:52311/api/action/42/status">
        <ActionID>42</ActionID>
        <Status>Open</Status>
        <DateIssued>Fri, 21 Nov 2014 16:49:04 +0000</DateIssued>
    </ActionResults>
</BESAPI>
```

As an alternative, you can get the same result by using a Session Relevance query as follows:

```
https://{server}:52311/api/query?relevance=(id of it, state of it, time
 issued of it)
of bes action whose (id of it = {id})
```

In return, you get the ID of the action, its state, and the time it was issued, as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
    <Query Resource="(id of it, state of it, time issued of it) of bes
 action whose (id of it = 42)">
        <Result>
            <Tuple>
                <Answer type="integer">42</Answer>
                <Answer type="string">Open</Answer>
                <Answer type="time">Fri, 21 Nov 2014 16:49:04 +0000</
Answer>
            </Tuple>
        </Result>
        <Evaluation>
            <Time>0.127ms</Time>
            <Plurality>Singular</Plurality>
        </Evaluation>
    </Query>
</BESAPI>
```

Chapter 6. REST API Resources

Begin learning about the available REST API resources and how to use them.

Action

GET /api/actions

Returns a list of actions.

POST /api/actions

Creates a new action.

GET /api/action/{action id}

Fetches the BES XML representation of the specified action.

DELETE /api/action/{action id}

Stops and deletes the specified action.

GET /api/action/{action id}/status

Gets the status of an action against it's targets.

POST /api/action/{action id}/stop

Stops the specified action.

Filtering Response Fields

You can use the `?fields=` parameter to limit the fields returned for a given resource when using the API resources `/api/actions` and `/api/action/{action id}/status`. The value following the `?fields=` parameter is the filter. Because the XML is case sensitive, ensure that you specify the correct case to avoid errors.

Use these characters to define the filter:

- `,` to separate elements, children, and attribute pairs
- `(...)` to denote children within a field
- `&`as pairing marker for attributes

- `<...>` to denote attributes
- = to mark LHS and RHS of attributes

Note: These are reserved characters. By default, they are not allowed in the name of the filter.

These are some example of filtering results using `?fields=`:

```
/api/action/<action id>/status?fields=Status,Computer
/api/action/<action id>/status?fields=Computer(Status,State,StartTime)
/api/action/<action id>/status?
fields=Computer<ID=11111>,Computer(Status,StartTime)
```

Admin

GET /api/admin/fields

Returns the list of admin fields.

POST /api/admin/fields

Sets one or more admin fields.

GET /api/admin/field/{field}

Fetches specific admin field value.

PUT /api/admin/field/{field}

Sets a specific admin field.

POST /api/admin/field/{field}

Sets a specific admin field.

DELETE /api/admin/field/{field}

Deletes the specific admin field.

GET /api/admin/masthead

Exports the current masthead.

POST /api/admin/masthead

Updates the masthead in the database.

GET /api/admin/masthead/parameters

Returns the list of the masthead configuration parameters and values.

GET /api/admin/icon

Exports the current client icon.

PUT /api/admin/icon

Updates/sets client icon.

POST /api/admin/icon

Updates/sets client icon.

DELETE /api/admin/icon

Deletes current client icon.

POST /api/admin/propagate/operator/{operator}

Propagates the specified operator site.

POST /api/admin/propagate/actionsite

Propagates the master action site.

POST /api/admin/propagate/operators

Propagates all operator sites.

POST /api/admin/propagate/resetepoch

Resets the database epoch forcing Console's to refresh their cache.

GET /api/admin/options

Displays the current admin options for the server.

POST /api/admin/options

Updates the admin options from the provided BESAPI XML.

GET /api/admin/reports

Returns the client report settings.

GET /api/admin/reports/cert

Returns the server certificate.

GET /api/admin/reports/key

Returns the server certificate.

Analysis

Note: For information about how to specify a site, see Hints about BigFix REST API requests.

GET analyses/{site type}/{site name}

Fetches a list of analyses in the specified site.

POST analyses/{site type}/{site name}

Create an analysis in the specified site.

GET analysis/{site type}/{site name}/{analysis id}

Get the XML for a specific analysis.

PUT analysis/{site type}/{site name}/{analysis id}

Update an analysis.

POST analysis/{site type}/{site name}

Create an analysis with the supplied XML in the specified site.

DELETE analysis/{site type}/{site name}/{analysis id}

Delete and deactivate an analysis.

GET analysis/{site type}/{site name}/{analysis id}/activations

List of analysis activations.

POST analysis/{site type}/{site name}/{analysis id}/activations

Create a new activation for the current operator.

DELETE analysis/{site type}/{site name}/{analysis id}/activations

Deletes all activations for the specified analysis.

GET analysis/{site type}/{site name}/{analysis id}/activation/{activation id}

Gets analysis activation details.

DELETE analysis/{site type}/{site name}/{analysis id}/activation/{activation id}

Deactivates a specific analysis activation.

Baseline

Available starting from BigFix version 9.5.5.

Note: For information about how to specify a site, see Hints about BigFix REST API requests.

GET /api/baselines/{site type}/{site name}

Lists all baselines in a site.

POST /api/baselines/{site type}/{site name}

Creates a baseline.

GET /api/baseline/{site type}/{site name}/{id}

Shows the baseline with the specified {id}.

PUT /api/baseline/{site type}/{site name}/{id}

Modifies the baseline with the specified {id}.

DELETE /api/baseline/{site type}/{site name}/{id}

Deletes the baseline with the specified {id}.

GET /api/baseline/{site type}/{site name}/{id}/computers

Lists all relevant computers of the baseline with the specified {id}.

GET /api/baseline/{site type}/{site name}/{id}/sync

Shows a synchronized version of the baseline with the specified {id}.

Baseline Component Synchronization Status

A baseline component is created from a Fixlet or a task and one of its actions. The Fixlet or task and the selected action are recorded as the component's source. If the source is modified, the component is not automatically updated. For example, if a source Fixlet is modified, or a source action is deleted, the corresponding component is not automatically updated. This is why components become out-of-sync.

You can determine if a component is out-of-sync by examining its element in the baseline XML. Each component's element has a `SyncStatus` attribute that indicates the component's synchronization status. The possible values of this attribute are defined below:

- `SyncStatus="synchronized"` - The component is synchronized.
- `SyncStatus="source fixlet differs"` - The component is not synchronized because it differs from the source.
- `SyncStatus="source fixlet differs (source action has been deleted)"` - The component is not synchronized because the source's action used to create the component no longer exists.
- `SyncStatus="source unavailable"` - The component is not synchronized because the source no longer exists.

Manipulating Baseline Components

Baseline components, or component groups, can be manipulated in several ways using the REST API. Because a baseline's components are specified in the baseline's XML, you can manipulate the components by manipulating the XML.

Run these steps to manipulate a baseline's components:

1. Get the baseline XML using `GET /api/baseline/{site type}/{site name}/{id}` request.
2. Manipulate the XML.
3. Update the baseline using `PUT /api/baseline/{site type}/{site name}/{id}` request.

Following the steps listed above you can:

- Create components, or component groups, by creating their elements in the baseline XML.

- Reorder components, or components groups, by reordering their elements in the baseline XML.
- Delete components, or components groups, by deleting their elements in the baseline XML.
- Selectively synchronize components: See the section below for how to do this.

Selectively Synchronizing Baseline Components

It is possible for you to synchronize all of a baseline's components using the two step process described for the request `GET /api/baseline/{site type}/{site name}/{id}/sync`.

However, it is also possible to selectively synchronize only some components. You can do this by modifying their elements to have the same format as when creating a component from source. This format is described in the Creating Baseline Components Directly From Source section for the request `POST /api/baseline/{site type}/{site name}`.

Specifically, to get this format, you will need to manipulate the XML by manipulating each BaselineComponent XML element that corresponds to a component that you want synchronize. The manipulation is described as follows:

- Delete the Name XML attribute.
- Delete all the children XML elements, making sure that no ActionScript, Relevance, SuccessCriteria, or Delay child elements exist.

Example

Let's say that you want to synchronize only the first component of the below baseline:

```
<?xml version="1.0" encoding="UTF-8"?>
<BES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BES.xsd">
    <Baseline>
        <Title>Custom Baseline</Title>
        <Description />
        <Relevance>true</Relevance>
        <BaselineComponentCollection>
            <BaselineComponentGroup>
                <BaselineComponent Name="CustomFixlet1"
 IncludeInRelevance="true" SourceSiteURL=
                 "http://{root-server}:52311/cgi-bin/bfgather.exe/
actionsite" SourceID="40"
                  ActionName="Action1" SyncStatus="source fixlet differs">
                  <ActionScript MIMEType="application/x-Fixlet-Windows-
Shell">//
                   Unsynchronized action script</ActionScript>
                    <SuccessCriteria
 Option="CustomRelevance">"unsynchronized custom relevance"</
SuccessCriteria>
                    <Relevance>"unsynchronized relevance"</Relevance>
                </BaselineComponent>
                <BaselineComponent Name="CustomFixlet2"
 IncludeInRelevance="true" SourceSiteURL=
                 "http://{root-server}:52311/cgi-bin/bfgather.exe/
actionsite" SourceID="41"
                  ActionName="Action1" SyncStatus="source fixlet differs">
                    <ActionScript MIMEType="application/x-Fixlet-Windows-
Shell">//
                    Unsynchronized action script</ActionScript>
                    <SuccessCriteria Option="OriginalRelevance" />
                    <Relevance>true</Relevance>
                </BaselineComponent>
            </BaselineComponentGroup>
        </BaselineComponentCollection>
    </Baseline>
</BES>
```

You would accomplish the synchronization by first manipulating the above XML to produce the below XML, and then updating the baseline by performing the PUT request using the below XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<BES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BES.xsd">
    <Baseline>
        <Title>Custom Baseline</Title>
        <Description />
        <Relevance>true</Relevance>
        <BaselineComponentCollection>
            <BaselineComponentGroup>
                <BaselineComponent IncludeInRelevance="true"
 SourceSiteURL="http://{root-server}:
                 52311/cgi-bin/bfgather.exe/actionsite"
                 SourceID="40" ActionName="Action1" SyncStatus="source
 fixlet differs" />
                <BaselineComponent Name="CustomFixlet2"
 IncludeInRelevance="true"
                SourceSiteURL="http://{root-server}:52311
                /cgi-bin/bfgather.exe/actionsite" SourceID="41"
 ActionName="Action1"
                SyncStatus="source fixlet differs">
                <ActionScript MIMEType="application/x-Fixlet-Windows-
Shell">//
                 Unsynchronized action script</ActionScript>
                    <SuccessCriteria Option="OriginalRelevance" />
                    <Relevance>true</Relevance>
                </BaselineComponent>
            </BaselineComponentGroup>
        </BaselineComponentCollection>
    </Baseline>
```

```
</BES>
```

After the baseline is synchronized, showing it using GET request should return the below XML, which has the first component synchronized:

```
<?xml version="1.0" encoding="UTF-8"?>
<BES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BES.xsd">
    <Baseline>
        <Title>Custom Baseline</Title>
        <Description />
        <Relevance>true</Relevance>
        <BaselineComponentCollection>
            <BaselineComponentGroup>
                <BaselineComponent Name="CustomFixlet1"
 IncludeInRelevance="true" SourceSiteURL=
                "http://{root-server}:52311/cgi-bin/bfgather.exe/
actionsite" SourceID="40" ActionName="Action1"
                 SyncStatus="synchronized">
                    <ActionScript MIMEType="application/x-Fixlet-Windows-
Shell">// Synchronized action script</ActionScript>
                    <SuccessCriteria Option="CustomRelevance">"synchronized
 custom relevance"</SuccessCriteria>
                    <Relevance>"synchronized relevance"</Relevance>
                </BaselineComponent>
                <BaselineComponent Name="CustomFixlet2"
 IncludeInRelevance="true" SourceSiteURL="http://{root-server}:52311/
cgi-bin/bfgather.exe/actionsite" SourceID="41" ActionName="Action1"
 SyncStatus="source fixlet differs">
                    <ActionScript MIMEType="application/x-Fixlet-Windows-
Shell">// Unsynchronized
                  action script</ActionScript>
                    <SuccessCriteria Option="OriginalRelevance" />
                    <Relevance>true</Relevance>
                </BaselineComponent>
            </BaselineComponentGroup>
        </BaselineComponentCollection>
    </Baseline>
</BES>
```

BigFix Query

POST /api/clientquery

Submits a new BigFix Query request.

GET /api/clientquery/{id}

Retrieves the submitted query that was assigned the identifier *{id}*.

GET /api/clientqueryresults/{id}?par1=value1&par2=value2&..

Retrieves the result of a BigFix Query, optionally with paging options.

Note:

1. If you are using the REST API, be aware that only the operator issuing the query can see its results.
2. BigFix now supports queries with inspector context when you set the `UseClientContext` element to *true* in the REST API requests.

Targeting by group criteria

If you target your query by group, ensure to specify the *SiteName*. This is an example for a group named manual created by a master operator:

```
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation=
   "BESAPI.xsd">
  <ClientQuery>
  <ApplicabilityRelevance>true</ApplicabilityRelevance>
  <QueryText>names of processes</QueryText>
  <Target>
      <ComputerGroup>
        <Name>manual</Name>
        <SiteName>ActionSite</SiteName>
      </ComputerGroup>
  </Target>
  </ClientQuery>
</BESAPI>
```

By default, the highest number of targets for each group is 100.

You can change this value by editing the **_Enterprise Server_BigFix Query_MaxTargetsForGroups** computer setting.

This is how the **_Enterprise Server_BigFix Query_MaxTargetsForGroups** settings works:

- If the number of targets in a group exceeds the value of **_Enterprise Server_BigFix Query_MaxTargetsForGroups**, the BigFix Query request is sent to all the clients and each client determines whether or not it is a member of the targeted group.
- If the number of targets does not exceed the value of **_Enterprise Server_BigFix Query_MaxTargetsForGroups**, the BigFix server determines which clients are members of the targeted group and the BigFix Query request is sent only to those clients.
- If the list of specified groups contains at least one automatic computer group or one client-evaluated static computer group, this settings is ignored and the BigFix Query request is sent to all the clients. Each client determines whether or not it is a member of the targeted group.

Computer

GET api/computers

Fetches a list of computers.

GET api/computer/{computer id}

Gets a computer's core properties.

DELETE api/computer/{computer id}

Marks a computer as deleted in the database.

GET api/computer/{computer id}/settings

Gets a computer's settings.

POST api/computer/{computer id}/settings

Updates or creates the value of a setting.

DELETE api/computer/{computer id}/settings

Deletes all activations for the specified analysis.

POST api/computers/settings

Updates or creates settings for multiple computers.

GET api/computer/{computer id}/setting/{setting name}

Gets the setting value.

PUT api/computer/{computer id}/setting/{setting name}

Updates the setting value.

POST api/computer/{computer id}/setting/{setting name}

Updates or creates the setting value.

DELETE api/computer/{computer id}/setting/{setting name}

Delete computer setting.

GET api/computer/{computer id}/fixlets

Gets the list of relevant Fixlets.

GET api/computer/{computer id}/tasks

Gets the list of relevant tasks.

GET api/computer/{computer id}/fixletsandtasks

Gets the list of all the relevant fixlets and tasks.

GET api/computer/{computer id}/analyses

Gets the list of relevant analyses and their ID.

GET api/computer/{computer id}/baselines

Gets the list of relevant baselines and their ID.

GET api/computer/{computer id}/mailbox

Gets contents of the computer's mailbox.

Filtering Response Fields

You can use the `?fields=` parameter to limit the fields returned for a given resource when using the API resources `/api/computers` and `/api/computer/{computer id}`. The value following the `?fields=` parameter is the filter. Because the XML is case sensitive, ensure that you specify the correct case to avoid errors.

Use these characters to define the filter:

- `,` to separate elements, children, and attribute pairs
- `,` within the parenthesis to denote multiple children
- `&`as pairing marker for attributes
- `<...>` to denote attributes
- = to mark LHS and RHS of attributes

Note: These are reserved characters. By default, they are not allowed in the name of the filter.

These are some example of filtering results using `?fields=`:

```
/api/computer/1234?fields=Property<Name=Computer%20Name,OS,Last%20Report
%20Time>
/api/computer/1234?
fields=Property<Analysis&Name=Analysis1&Computer%20Name,&OS,Analysis2&Last
%20Report%20Time>
```

Computer Group

Note: For information about how to specify a site, see Hints about BigFix REST API requests.

GET computergroups/{site type}/{site name}

Fetches the list of computer groups in the specified site.

POST computergroups/{site type}/{site name}

Creates a computer group in the specified site as described in the posted XML document.

GET computergroup/{site type}/{site name}/{id}

Fetches the BES XSD representation of the specified computer group.

PUT computergroup/{site type}/{site name}/{id}

Updates the specified computer group.

POST computergroup/{site type}/{site name}

Create a computer group in the specified site.

DELETE computergroup/{site type}/{site name}/{id}

Deletes the indicated computer group.

GET computergroup/{site type}/{site name}/{id}/computers

Fetches a list of computer relevant to this computer group. This essentially gives you the computers that are members of this group.

Dashboard Variables

GET dashboardvariables

Fetch list of dashboard IDs that have stored information.

GET dashboardvariables/{dashboard ID}

Lists variables for the specified dashboard.

POST dashboardvariables/{dashboard ID}

Creates new dashboard variable.

DELETE dashboardvariables/{dashboard ID}

Deletes all stored variables for a dashboard.

GET dashboardvariable/{dashboard ID}/{variable name}

Fetches value of a specific dashboard variable.

PUT dashboardvariable/{dashboard ID}/{variable name}

Updates a specific dashboard variable.

POST dashboardvariable/{dashboard ID}/{variable name}

Updates a specific dashboard variable.

DELETE dashboardvariable/{dashboard ID}/{variable name}

Deletes the specified dashboard variable.

Fixlet

Note: For information about how to specify a site, see Hints about BigFix REST API requests.

GET fixlets/{site type}/{site name}

Fetches a list of Fixlets of a particular site.

POST fixlets/{site type}/{site name}

Creates a fixlet in the specified site.

GET fixlet/{site type}/{site name}/{fixlet id}

Returns the details of the specified Fixlet.

PUT fixlet/{site type}/{site name}/{fixlet id}

Updates a Fixlet.

POST fixlet/{site type}/{site name}

Creates a Fixlet in the specified site.

DELETE fixlet/{site type}/{site name}/{fixlet id}

Deletes a Fixlet.

GET fixlet/{site type}/{site name}/{fixlet id}/computers

Lists the computers that are relevant for this Fixlet.

Help

GET help

Returns a list of top-level available resources.

GET help/{resource}

Returns a list of all available commands for the specified resource for each supported HTTP method.

Import

Note: For information about how to specify a site, see Hints about BigFix REST API requests.

POST import/{site type}/{site name}

Import all provided objects in XML document.

LDAP Directory

GET ldapdirectories

Fetches all LDAP Directories.

POST ldapdirectories

Creates LDAP Directory.

GET ldapdirectory/{id}

Fetches detailed information for a directory.

PUT ldapdirectory/{id}

Updates LDAP directory.

POST ldapdirectory/{id}

Updates LDAP directory.

DELETE ldapdirectory/{id}

Deletes LDAP Directory.

Login

GET login/

Performs login.

Mailbox

Manage computer mailboxes and the files they contain.

GET mailbox/{computer id}

Fetches a list of files in a given computer's mailbox.

POST mailbox/{computer id}

Creates a new mailbox file.

GET mailbox/{computer id}/{file id}

Fetches the details of the specified file in the specified computer's mailbox.

PUT mailbox/{computer id}/{file id}

Updates the specified mailbox file.

POST mailbox/{computer id}/{file id}

Updates the specified new mailbox file.

DELETE mailbox/{computer id}/{file id}

Deletes the specified files from the specified computer's mailbox.

POST mailbox/

Creates a new mailbox file.

Operator

The following information applies also to LDAP operators.

GET operators

Fetches all operators.

POST operators

Creates an operator.

Note: The following REST APIs use operator name. If the deployment option, "enableRESTAPIOperatorID" is enabled, all the REST APIs that use the operator name are disabled and replaced with the equivalent REST APIs that use the operator id, which means enabling any REST API that uses operator ID renders all the REST APIs that use operator name unusable.

GET operator/{operator name}

Fetches detailed information about an operator.

Note: The operator name to use for retrieving operator information (especially when the name contains special characters) must be the same as that returned by the get/operators API. The field `PostActionBehaviorPrivilege` in the XML response is always referred at the "Explicit Permissions" value.

PUT operator/{operator name}

Updates an operator.

POST operator/{operator name}

Updates an operator.

DELETE operator/{operator name}

Deletes an operator.

GET operator/{operator name}/roles

Shows a list of roles the operator is a member of.

> **Note:** The following APIs use operator ID. Before using any of these APIs, you must enable the deployment option, "enableRESTAPIOperatorID". The enableRESTAPIOperatorID option disables all the REST APIs that use the operator name and replaces them with the equivalent REST APIs that use the operator id, which means enabling any REST APIs that use operator ID renders the REST APIs that use operator name unusable.

GET operator/{operator ID}

Fetches detailed information about a specific operator.

PUT operator/{operator ID}

Updates an operator.

POST operator/{operator ID}

Updates an operator.

DELETE operator/{operator ID}

Deletes an operator.

Filtering Response Fields

You can use the `?fields=` parameter to limit the fields returned for a given resource when using the API resources `/api/operators` and `/api/operator/{operator name}`. The value following the `?fields=` parameter is the filter. Because the XML is case sensitive, ensure that you specify the correct case to avoid errors.

Use these characters to define the filter:

- `,` to separate elements, children, and attribute pairs
- `(...)` to denote children within a field
- `&`as pairing marker for attributes
- `<...>` to denote attributes
- = to mark LHS and RHS of attributes

Note: These are reserved characters. By default, they are not allowed in the name of the filter.

These are some example of filtering results using `?fields=`:

```
/api/operators?
fields=Name,MasterOperator,InterfaceLogins(WebUI),InterfaceLogins(Applications)
/api/operator/mo1?
fields=LastLoginTime,CanSubmitQueries,InterfaceLogins(API,Console)
/api/operator/nmo1?
fields=ShowOtherActions,StopOtherActions,CanCreateActions,LoginPermission,Name
```

Property

GET properties*

Fetches a list of all properties.

POST properties

Creates a new property.

GET property/{id}

Fetches detailed information about a specific property.

PUT property/{id}

Updates the specified property.

POST property

Updates the specified property.

DELETE property/{id}

Deletes the specified property.

Query

POST /api/query

Evaluate a relevance expression and get the result.

GET /api/query

Evaluate a relevance expression and get the result.

Replication

GET /api/replication/servers

Fetches a list of all replication servers and their links.

POST /api/replication/servers

Posts settings for a single or all servers at once.

GET replication/server/{id}

Fetches the XML for a specific replication server and it's link.

PUT replication/server/{id}

Updates the replication server **ReplicationIntervalSeconds** and associated links if supplied.

POST replication/server/{id}

Updates the replication server **ReplicationIntervalSeconds** and associated links if supplied.

GET replication/server/{source id}/link/{destination id}

Fetches the XML for a specific replication server link.

PUT replication/server/{source id}/link/{destination id}

Updates replication link weight.

POST replication/server/{source id}/link/{destination id}

Update replication link weight.

DELETE replication/server/{source id}/link/{destination id}

Deletes the specified link.

Role

GET /api/roles

Fetches a list of all Roles.

POST /api/roles

Creates a new role.

GET role/{role id}

Fetches the XML for the specified role.

PUT role/{role id}

Updates a role.

POST role/{role id}

Replace existing role with supplied XML.

DELETE role/{role id}

Deletes a role.

In a BigFix Distributed Server Architecture (DSA) environment, where multiple servers are installed, you can perform actions related to roles only on primary servers. When on a secondary server, you cannot create, modify, or delete roles. Any attempt to perform these actions is prevented and you get the following error message:

"This operation requires a role change that can be performed only on the main server."

Site

Note: For information about how to specify a site, see Hints about BigFix REST API requests.

GET sites

Fetches a list of sites and their types.

POST sites

Creates a custom site.

GET site/{site type}/{site name}

Fetches a specific site.

PUT site/{site type}/{site name}

Updates the specified site.

DELETE site/{site type}/{site name}

Deletes the specified site and its content.

GET site/{site type}/{site name}/permissions

Fetches the list of operators/roles/LDAP groups and their permissions on the specified site.

PUT site/{site type}/{site name}/permissions

Sets the permissions for a site.

DELETE site/{site type}/{site name}/permission

Removes the permissions for a site.

GET site/{site type}/{site name}/permission/{operator/role}/{name/id}

Returns specific site permission for operator or role.

PUT site/{site type}/{site name}/permission/{operator/role}/{name/id}

Updates site permission for specified user or role.

POST site/{site type}/{site name}/permission/{operator/role}/{name/id}

Sets site permission for specified user or role.

DELETE site/{site type}/{site name}/permission/{operator/role}/{name/id}

Removes permission from site.

GET site/{site type}/{site name}/content

Fetches a list all content within a site.

GET site/{site type}/{site name}/files

Fetches a list all site files.

POST /api/site/{site}/files

Adds files to a site.

GET site/{site type}/{site name}/file/{file id}

Gets a site file.

PUT site/{site type}/{site name}/file/{file id}

Updates a site file.

POST site/{site type}/{site name}/file/{file id}

Updates a site file.

DELETE site/{site type}/{site name}/file/{file id}

Deletes the specified file.

POST site/{site type}/{site name}/file/{file name}

Creates a site file with the specified name.

Task

Note: For information about how to specify a site, see Hints about BigFix REST API requests.

GET /api/tasks/{site type}/{site name}

Fetches a list of tasks of a particular site.

POST /api/tasks/{site type}/{site name}

Creates a task in the specified site.

GET /api/task/{site type}/{site name}/{task id}

Returns the details of the specified task.

PUT /api/task/{site type}/{site name}/{task id}

Updates a task.

POST /api/task/{site type}/{site name}

Creates a task in the specified site.

DELETE /api/task/{site type}/{site name}/{task id}

Deletes a task.

GET /api/task/{site type}/{site name}/{fixlet id}/computers

Retrieves the list of computers that are relevant for this task.

Upload

POST upload

Uploads one or more files.

GET uploads

Returns a list of all FileUpload elements.

GET upload/{filelocation}

Returns a FileUpload at {filelocation}. {filelocation} is a hash/name form.

GET upload/{filelocation}/references

Returns a list of FileUploadReference for a FileUpload at {filelocation}.

POST upload/{filelocation}/references

Creates a FileUploadReference for a FileUpload at {filelocation}. This also returns a FileUploadReference.

GET upload/{filelocation}/reference/{id}

Returns a FileUploadReference with {id} for a FileUpload at {filelocation}.

DELETE

Deletes a FileUploadReference with {id} for a FileUpload at {filelocation}.

Web Reports

GET /api/webreports

Fetches a list of Web Reports servers.

GET /api/webreports/details

Fetches details of the Web Reports servers in terms of ID, priority and server URL.

PUT /api/webreports/{id}

Updates the priority of a Web Reports server based on its unique ID.

The preferred Web Reports server is the one with the highest value set for `Priority`.

Chapter 7. Schema files BES.xsd and BESAPI.xsd

During the installation of the BigFix server component, the BESAPI.xsd and BES.xsd schema files are stored in the installation path. They contain the information needed to correctly represent the BigFix object instances in XML format. To learn how to download an XSD file, click its name.

These files are automatically updated, during the product upgrade, whenever fields are added or modified.

To ensure consistency, use the version of the schema files available on your BigFix server under the path:

- On Windows systems: `C:\Program Files (x86)\BigFix Enterprise\BigFix server \Reference`
- On RedHat Linux Enterprise version 5.0 or later: `/opt/BESServer/reference`

The schema files are also available via HTTP/HTTPS from the BigFix Root Server to facilitate automated XML validation:

```
https://<bigfix_server>:<port>/xmlschema/BES.xsd
https://<bigfix_server>:<port>/xmlschema/BESAPI.xsd
```

(where port is 52311 by default).

BESAPI.xsd schema file

The BESAPI.xsd schema file is available for download (under the "Utilities" section) at BigFix Enterprise Suite Download Center on the corresponding release pages. For releases older than 9.5.13, use the file available (under the "Utilities" section) at the Patch 13 release page.

Important: To ensure consistency, use the version of the schema files available on your BigFix server under the path:

- On Windows systems: `C:\Program Files (x86)\BigFix Enterprise\BigFix server \Reference`
- On RedHat Linux Enterprise version 5.0 or later: `/opt/BESServer/reference`

The file is also available via HTTP/HTTPS from the BigFix Root Server:

```
https://<bigfix_server>:<port>/xmlschema/BESAPI.xsd
```

BES.xsd schema file

The BES.xsd schema file is available for download (under the "Utilities" section) at BigFix Enterprise Suite Download Center on the corresponding release pages. For releases older than 9.5.13, use the file available (under the "Utilities" section) at the Patch 13 release page.

Important: To ensure consistency, use the version of the schema files available on your BigFix server under the path:

- On Windows systems: `C:\Program Files (x86)\BigFix Enterprise\BigFix server \Reference`
- On RedHat Linux Enterprise version 5.0 or later: `/opt/BESServer/reference`

The file is also available via HTTP/HTTPS from the BigFix Root Server:

```
https://<bigfix_server>:<port>/xmlschema/BES.xsd
```

Chapter 8. Examples using cURL and python

Learn how to implement common use cases with the REST API.

The examples have been implemented for the following tools:

- cURL
- python

For these tools to establish a secure connection to the server, you must install a custom HTTPS certificate. Alternatively, you can provide a disable HTTPS connection option to the tool you are using. This is described below for every tool.
For the following examples to work successfully:

- `username` must be your user name.
- `password` must be your password.
- `server` must be the hos tname of your root server.
- `port` must be the port of your root server. By default this is **52311**.

cURL Usage

Security

Use the `--cacert` option to specify the custom HTTPS certificate. To disable SSL verification, use the `--insecure` option.

python Usage

Examples have been written using the python Request Package, which can be installed with pip using the following command:

```
$ pip install requests
```

Security

Requests verifies SSL certificates for HTTPS requests. This is enabled by default, and it will throw an SSL error if it is unable to verify the certificate. You can pass `verify` the path to a CA certificate or a custom one. To ignore verifying SSL, set `verify` to False.

```
>>> requests.get('https://bigfix.server:52311/api/help', verify=False)
<Response [200]>
```

Create Action

Create and execute an action contained in `action.xml`.

cURL

```
curl -X POST --data-binary @action.xml --user {username}:{password}
https://{server}:{port}/api/actions
```

python

```
import requests
with open('action.xml', 'rb') as xml:
    r = requests.post('https://{server}:{port}/api/actions',
 auth=('{username}',
    '{password}'), data=xml)
    print(r.text)
```

Input

action.xml

```
<?xml version="1.0" encoding="utf-8"?>
<BES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd=
 "http://www.w3.org/2001/XMLSchema" SkipUI="true">
    <SingleAction>
        <Title>test action</Title>
        <Relevance>true</Relevance>
        <ActionScript>setting "test action"="{now as string}" on "{now}"
 for client</ActionScript>
        <SuccessCriteria />
        <Settings />
        <SettingsLocks />
        <Target>
            <AllComputers>true</AllComputers>
        </Target>
    </SingleAction>
</BES>
```

Create Analysis

Create a new analysis for a site.

cURL

```
curl -X POST --data-binary @analysis.xml --user {username}:
{password} https://{server}:{port}/api/analyses/{site}
```

python

```
import requests
with open('analysis.xml', 'rb') as xml:
  r = requests.post('https://{server}:{port}/api/analyses/{site}',
 auth=('{username}',
  '{password}'), data=xml)
  print(r.text)
```

Input

analysis.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<BES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BES.xsd">
    <Analysis>
    <Title>Custom Analytics I</Title>
    <Description><![CDATA[my description]]></Description>
    <Relevance>true</Relevance>
    <Source>Internal</Source>
    <SourceReleaseDate>2016-05-25</SourceReleaseDate>
    <MIMEField>
      <Name>x-fixlet-modification-time</Name>
      <Value>Wed, 25 May 2016 20:59:59 +0000</Value>
    </MIMEField>
    <Domain>BESC</Domain>
    <Property Name="New Property" ID="1">operating system</Property>
  </Analysis>
</BES>
```

Output

```
<?xml version="1.0" encoding="UTF-8"?>
<BES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BES.xsd">
    <Analysis Resource="https://{server}:{port}/api/analysis/{site}/1"
 LastModified="Wed, 25 May
     2016 20:59:59 +0000">
    <Name>Custom Analysis I</Name>
    <ID>1</ID>
  </Analysis>
</BES>
```

Create Dashboard Variable

Creates a new dashboard variable with the contents of `dashvar.xml`. Adds the variable to the specified dashboard ID if the ID already exists, otherwise creates a dashboard ID with the specified dashboard variable.

- `dashboard ID` is the ID of the dashboard to which this variable belongs.
- `variable name` is the name of the variable to be created.

cURL

```
curl -X POST --data-binary @dashvar.xml --user {username}:{password}
https://{server}:{port}/api/dashboardvariables/{dashboard ID}
```

python

```
import requests
with open('site.xml', 'rb') as xml:
    r = requests.post('https://{server}:{port}/api/dashboardvariables/
{dashboard ID}',
    auth=('{username}', '{password}'), data=xml)
    print(r.text)
```

Input

dashvar.xml

```
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="BESAPI.xsd">
        <DashboardData Resource="https://localhost:52311/api/
dashboardvariable/
         ShowNonRelevantContent/Show">
                <Dashboard>ShowNonRelevantContent</Dashboard>
                <Name>Show</Name>
                <IsPrivate>false</IsPrivate>
                <User>__op_1</User>
                <Value>true</Value>
        </DashboardData>
</BESAPI>
```

Create File in a Site

This command will genereate a POST request to create one or more files specified by `-F "file=@/{file_pathname}` in the specified site. For information about how to specify a site, see Hints about BigFix REST API requests. Depending on the permission to access the site, this command can be run by master operators and non master operators.

cURL

To add a file to a custom site as master operator, run the following command:

```
curl -k --user master:{password} -X POST -F "file=@/tmp/file.my"
"https://{server}:{port}/api/site/custom/test/file/file.my"
```

To add a file to the master action site as master operator, run the following command:

```
curl -k --user master:{password} -X POST -F "file=@/tmp/file.my"
"https://{server}:{port}/api/site/master/file/file.my?force=true"
```

Where the option `force=true` allows to override the file it if already exists in the site.

To add a file to the operator site as non master operator, run the following command:

```
curl -k --user nmo:{password} -X POST -F "file=@/tmp/file.my"
"https://{server}:{port}/api/site/operator/nmo/file/file.my?isClient=true"
```

Where the option `isClient=true` allows to make the file available for download by Clients.

The options `force` and `isClient` apply when running post and put requests against files, both as master and as non master operator. For these two options values different from false are managed as true.

Use the syntax `?key1=value1;key2=value2` to specify multiple options, for example:

```
curl -k --user master:{password} -X POST -F "file=@/tmp/file.my" "https://
{server}:
{port}/api/site/custom/test/file/file.my?force=true;isClientFile=true"
```

Note: Be aware that no validity check on input options and parameters is automatically performed.

Create Fixlet

This command will genereate a POST request to create a fixlet specified by `new.xml` in the site `site type/site name`.

- `site name` is the the name of the site under which the fixlet will be created.
- `site type` is the type of the site.

`site type` and `site name` can be inpsected via GET `/api/sites`

cURL

```
curl -X POST --data-binary @new.xml --user {username}:{password} https://
{server}:
{port}/api/fixlets/{site type}/{site name}
```

python

```
import requests
with open('new.xml', 'rb') as xml:
    r = requests.post('https://{server}:{port}/api/fixlets/{site type}/
{site name}',
    auth=('{username}', '{password}'), data=xml)
print(r.text)
```

Input

```
<?xml version="1.0" encoding="UTF-8"?>
<BES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BES.xsd">
    <Fixlet>
        <Title>New Fixlet</Title>
        <Description> This is the example new fixlet to be created </
Description>
        <Relevance>exists folder "C:\Programs Files"</Relevance>
        <Category></Category>
        <Source>Internal</Source>
        <SourceID></SourceID>
        <SourceReleaseDate>2016-05-21</SourceReleaseDate>
    </Fixlet>
```

```
</BES>
```

Create Operator

Creates a new operator with the contents of `operator.xml`. This operation requires master operator privileges.

cURL

```
curl -X POST --data-binary @operator.xml --user {username}:{password}
https://{server}:{port}/api/operators
```

python

```
import requests
with open('operator.xml', 'rb') as xml:
    r = requests.post('https://{server}:{port}/api/operators',
 auth=('{username}', '{password}'), data=xml)
    print(r.text)
```

Input

operator.xml

```
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
        <Operator Resource="https://localhost:52311/api/operator/sample-
operator">
                <Name>sample-operator</Name>
                <Password>sample-password</Password>
                <MasterOperator>false</MasterOperator>
                <CustomContent>true</CustomContent>
                <ShowOtherActions>true</ShowOtherActions>
                <CanCreateActions>true</CanCreateActions>
                <PostActionBehaviorPrivilege>AllowRestartAndShutdown</
PostActionBehaviorPrivilege>
```

```
                <ActionScriptCommandsPrivilege>AllowRestartAndShutdown</
ActionScriptCommandsPrivilege>
                <CanLock>true</CanLock>
                <CanSendMultipleRefresh>true</CanSendMultipleRefresh>
                <LoginPermission>Unrestricted</LoginPermission>
                <UnmanagedAssetPrivilege>ShowAll</UnmanagedAssetPrivilege>
                <InterfaceLogins>
                        <Console>true</Console>
                        <WebUI>true</WebUI>
                        <API>true</API>
                </InterfaceLogins>
                <ComputerAssignments Match="Any"></ComputerAssignments>
        </Operator>
</BESAPI>
```

Create Role

This command will POST the file `role.xml` to the server to create a new role.

python

```
import requests
with open('role.xml', 'rb') as xml:
    r = requests.post('https://{server}:{port}/api/roles',
 auth=('{username}', '{password}'), data=xml)
    print(r.text)
```

Input

role.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
        <Role Resource="https://localhost:52311/api/role/id">
```

```
                <Name>name</Name>
                <ID>id</ID>
                <MasterOperator>0</MasterOperator>
                <CustomContent>1</CustomContent>
                <ShowOtherActions>1</ShowOtherActions>
                <CanCreateActions>1</CanCreateActions>
                <PostActionBehaviorPrivilege>AllowRestartAndShutdown</
PostActionBehaviorPrivilege>
                <ActionScriptCommandsPrivilege>AllowRestartAndShutdown</
ActionScriptCommandsPrivilege>
                <CanSendMultipleRefresh>1</CanSendMultipleRefresh>
                <CanLock>1</CanLock>
                <UnmanagedAssetPrivilege>ShowNone</UnmanagedAssetPrivilege>
                <InterfaceLogins>
                        <Console>true</Console>
                        <WebUI>true</WebUI>
                        <API>true</API>
                </InterfaceLogins>
        </Role>
</BESAPI>
```

Create Site

Creates the specified site on the server with the contents of `site.xml`. In this example, we create a custom site with a Relevance conditional that returns true if the name of the operating system contains "Win".

cURL

```
curl -X POST --data-binary @site.xml --user {username}:{password}
https://{server}:{port}/api/sites/
```

python

```
import requests
```

```
with open('site.xml', 'rb') as xml:
    xmldata = ''.join(xml.readlines())
    r = requests.post('https://{server}:{port}/api/sites',
 auth=('{username}',
    '{password}'), data=xmldata)
    print(r.text)
```

Input

site.xml

```
<BES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BES.xsd">
        <CustomSite>
                <Name>Test</Name>
                <GatherURL>http://bigfix.test:52311/cgi-bin/bfgather.exe/
CustomSite_Test</GatherURL>
                <Description>This is an example Custom Site</Description>
                <Domain>BESC</Domain>
                <GlobalReadPermission>false</GlobalReadPermission>
                <Subscription>
                        <Mode>Custom</Mode>
                        <CustomGroup JoinByIntersection="false">
                                <SearchComponentRelevance
 Comparison="IsTrue">
                                        <Relevance>name of operating system
 contains "Win"</Relevance>
                                </SearchComponentRelevance>
                        </CustomGroup>
                </Subscription>
        </CustomSite>
</BES>
```

Get Analyses

Get a site's available analyses.

cURL

```
curl --user {username}:{password} https://{server}:{port}/api/analyses/
{site}
```

python

```
import requests
r = requests.get('https://{server}:{port}/api/analyses/{site}',
 auth=('{username}', '{password}'))
print(r.text)
```

Output

```
<?xml version="1.0" encoding="UTF-8"?>
<BES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BES.xsd">
    <Analysis Resource="https://{server}:{port}/api/analysis/{site}/1"
     LastModified="Wed, 25 May 2016 20:59:59 +0000">
    <Name>Custom Analysis I</Name>
    <ID>1</ID>
  </Analysis>
    <Analysis Resource="https://{server}:{port}/api/analysis/{site}/2"
     LastModified="Wed, 25 May 2016 21:00:00 +0000">
    <Name>Custom Analysis II</Name>
    <ID>2</ID>
  </Analysis>
</BES>
```

Get Computer

Gets the properties of a specific computer registered in the console.

cURL

```
curl -X GET --user {username}:{password} https://{server}:{port}/api/
computer/{id}
```

python

```
import requests
r = requests.get('https://{server}:{port}/api/computer/{id}',
 auth=('{username}', '{password}'))
print(r.text)
```

Output

```
<?xml version="1.0" encoding="UTF-8"?>
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
  <Computer Resource="https://localhost:52311/api/computer/{id}">
    <Property Name="Client Settings">_BESClient_EMsg_File=/var/opt/
BESClient/besclientdebug.log</Property>
    <Property Name="Client Settings">_BESClient_EMsg_Detail=0</Property>
    <Property Name="Client Settings">__RelayServer1=</Property>
    <Property Name="Client Settings">__RelayServer2=</Property>
    <Property Name="Client Settings">__Relay_Control_Server1=</Property>
    <Property Name="Client Settings">__Relay_Control_Server2=</Property>
    <Property Name="Client Settings">__LockState=false</Property>
    <Property Name="Client
 Settings">_BESClient_UploadManager_BufferDirectory=/var/opt/BESClient/
     __BESData/__Global/Upload</Property>
    <Property Name="Client
 Settings">_BESClient_Resource_StartupNormalSpeed=0</Property>
    <Property Name="Client Settings">_BESRelay_HTTPServer_ServerRootPath=/
var/opt/BESServer/
     wwwrootbes</Property>
    <Property Name="Client
 Settings">_BESRelay_HTTPServer_PortNumber=52311</Property>
    <Property Name="Client Settings">_BESRelay_HTTPServer_LogFilePath=/var/
log/BESRelay.log</Property>
    <Property Name="Client
 Settings">_BESGather_Download_CheckInternetFlag=1</Property>
    <Property Name="Client Settings">_BESGather_Comm_UseUrlMoniker=0</
Property>
    <Property Name="Client Settings">_Enterprise
    Server_ClientRegister_UDPMessagePort=52311</Property>
    <Property Name="Client Settings">_BESRelay_UploadManager_ParentURL=</
Property>
    <Property Name="Client Settings">_BESServer_Database_DSN=BFENT</
Property>
    <Property Name="Client
 Settings">_BESServer_Database_DatabaseAddress=localhost</Property>
    <Property Name="Client Settings">_BESServer_Database_User=db2inst1</
Property>
    <Property Name="Client
 Settings">_BESServer_Database_Password={obf}YmlnZml4</Property>
    <Property Name="Client Settings">_BESServer_Database_Port=50000</
Property>
    <Property Name="Client
 Settings">_BESClient_DeploymentEncoding_IANAName=windows-1252</Property>
    <Property Name="Client Settings">_BESRelay_Log_Verbose=0</Property>
    <Property Name="Client
 Settings">_WebReports_HTTPServer_HostName=bigfix.test</Property>
    <Property Name="Client Settings">_WebReports_HTTPServer_PortNumber=80</
Property>
    <Property Name="Client
 Settings">_WebReports_HTTPServer_ServerRootPath=/var/opt/
BESWebReportsServer/
```

```
     wwwroot</Property>
    <Property Name="Client
 Settings">_WebReports_HTTPServer_SSLCertificateFilePath=</Property>
    <Property Name="Client Settings">_WebReports_HTTPServer_UseSSLFlag=0</
Property>
    <Property Name="Client
 Settings">_BESClient_DeploymentEncoding_IANAName=windows-1252</Property>
    <Property Name="Computer Name">bigfix.test</Property>
    <Property Name="OS">Linux Red Hat Enterprise Server 7.0
 (3.10.0-123.el7.x86_64)</Property>
    <Property Name="CPU">2600 MHz Core i7-3720QM</Property>
    <Property Name="Last Report Time">Wed, 25 May 2016 22:28:02 +0000</
Property>
    <Property Name="Locked">No</Property>
    <Property Name="BES Relay Selection Method">Manual</Property>
    <Property Name="Relay">BES Root Server</Property>
    <Property Name="Distance to BES Relay">0</Property>
    <Property Name="BES Relay Service Installed">BES Root Server</Property>
    <Property Name="Relay Name of Client">bigfix.test</Property>
    <Property Name="DNS Name">bigfix.test</Property>
    <Property Name="Active Directory Path">&lt;none&gt;</Property>
    <Property Name="IP Address">10.0.2.15</Property>
    <Property Name="IPv6 Address">fe80:0:0:0:a00:27ff:fe15:a420</Property>
    <Property Name="Subscribed Sites">http://sync.bigfix.com/cgi-bin/
bfgather/bessupport</Property>
    <Property Name="BES Root Server">bigfix.test (0)</Property>
    <Property Name="License Type">Non-Windows Server</Property>
    <Property Name="Agent Type">Native</Property>
    <Property Name="Device Type">Server</Property>
    <Property Name="Agent Version">9.2.7.53</Property>
    <Property Name="ID">{id}</Property>
    <Property Name="Computer Type">Physical</Property>
    <Property Name="User Name">&lt;none&gt;</Property>
    <Property Name="RAM">1856 MB</Property>
    <Property Name="Free Space on System Drive">33228 MB</Property>
    <Property Name="Total Size of System Drive">38373 MB</Property>
    <Property Name="BIOS">&lt;n/a&gt;</Property>
    <Property Name="Subnet Address">10.0.2.0</Property>
  </Computer>
</BESAPI>
```

Get Computers

Retrieves the list of computers registered in the console.

cURL

```
curl -X GET --user {username}:{password} https://{server}:{port}/api/
computers
```

python

```
import requests
r = requests.get('https://{server}:{port}/api/computers',
 auth=('{username}', '{password}'))
print(r.text)
```

Output

```
<?xml version="1.0" encoding="UTF-8"?>
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
    <Computer Resource="https://{server}:{port}/api/computer/11112538">
        <LastReportTime>Wed, 25 May 2016 22:28:03 +0000</LastReportTime>
        <ID>11112538</ID>
    </Computer>
    <Computer Resource="https://{server}:{port}/api/computer/11112539">
        <LastReportTime>Wed, 25 May 2016 22:28:03 +0000</LastReportTime>
        <ID>11112539</ID>
    </Computer>
    <Computer Resource="https://{server}:{port}/api/computer/11112540">
        <LastReportTime>Wed, 25 May 2016 22:28:03 +0000</LastReportTime>
        <ID>11112540</ID>
    </Computer>
</BESAPI>
```

Get Sites

Gets the current list of sites on the server.

cURL

```
curl -X GET --user {username}:{password} https://{server}:{port}/api/sites
```

python

```
import requests
r = requests.get('https://{server}:{port}/api/sites', auth=('{username}',
 '{password}'))
print(r.text)
```

Update Analysis

Update a previously created analysis.

cURL

```
curl -X PUT --data-binary @analysis.xml
--user {username}:{password} https://{server}:{port}/api/analysis/{site}/
{analysis id}
```

python

```
import requests
with open('analysis.xml', 'rb') as xml:
  r = requests.put('https://{server}:{port}/api/analysis/{site}/{analysis
 id}',
  auth=('{username}', '{password}'), data=xml)
  print(r.text)
```

Input

analysis.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<BES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BES.xsd">
    <Analysis>
    <Title>Better Custom Analytics I</Title>
    <Description><![CDATA[my new description]]></Description>
    <Relevance>true</Relevance>
    <Source>Internal</Source>
    <SourceReleaseDate>2016-05-25</SourceReleaseDate>
    <MIMEField>
      <Name>x-fixlet-modification-time</Name>
      <Value>Wed, 25 May 2016 20:59:59 +0000</Value>
    </MIMEField>
    <Domain>BESC</Domain>
    <Property Name="Newer Property" ID="1">now</Property>
  </Analysis>
</BES>
```

Output

```
{analysis id}
```

Update Fixlet

This command will generate a PUT request to update the fixlet found with the id of `fixlet id` under `site type/site name`, replacing it with an updated fixlet specified in `update.xml`.

- `site type` is the type of the site containing the fixlet to be updated.
- `site name` is the the name of the site containing the fixlet to be updated.

`site type` and `site name` can be inspected via GET /api/sites.

cURL

```
curl -X PUT --data-binary @update.xml --user {username}:{password} https://
{server}:
{port}/api/fixlet/{site type}/{site name}/{fixlet id}
```

python

```
import requests
with open('update.xml', 'rb') as xml:
    r = requests.put('https://{server}:{port}/api/fixlet/{site type}/{site
 name}/{fixlet id}',
auth=('{username}', '{password}'), data=xml)
print(r.text)
```

Input

```
<?xml version="1.0" encoding="UTF-8"?>
<BES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BES.xsd">
    <Fixlet>
        <Title>Updated Fixlet</Title>
        <Description><![CDATA[&lt;enter a description of the problem and
 the corrective action here&gt;
         ]]></Description>
        <Relevance>exists folder "C:\"</Relevance>
        <Category></Category>
        <Source>Internal</Source>
        <SourceID></SourceID>
        <SourceReleaseDate>2016-05-24</SourceReleaseDate>
        <SourceSeverity></SourceSeverity>
        <CVENames></CVENames>
        <SANSID></SANSID>
        <Domain>BESC</Domain>
        <DefaultAction ID="Action1">
            <Description>
                <PreLink>Click </PreLink>
                <Link>here</Link>
                <PostLink> to deploy this action.</PostLink>
            </Description>
            <ActionScript MIMEType="application/x-Fixlet-Windows-Shell">//
 Updated Action here
            </ActionScript>
         </DefaultAction>
    </Fixlet>
</BES>
```

Update Role

This command will POST the file `role.xml` (can be a partial XML) to the server to update an existing role.

python

```
import requests
with open('role.xml', 'rb') as xml:
    r = requests.put('https://{server}:{port}/api/role/{role id}',
 auth=('{username}',
'{password}'), data=xml)
    print(r.text)
```

Input

role.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
        <Role Resource="https://localhost:52311/api/role/id">
                <Name>name</Name>
                <ID>id</ID>
                <MasterOperator>0</MasterOperator>
                <CustomContent>1</CustomContent>
                <ShowOtherActions>1</ShowOtherActions>
                <CanCreateActions>1</CanCreateActions>
                <PostActionBehaviorPrivilege>AllowRestartAndShutdown</
PostActionBehaviorPrivilege>
                <ActionScriptCommandsPrivilege>AllowRestartAndShutdown</
ActionScriptCommandsPrivilege>
                <CanSendMultipleRefresh>1</CanSendMultipleRefresh>
                <CanLock>1</CanLock>
                <UnmanagedAssetPrivilege>ShowNone</UnmanagedAssetPrivilege>
                <InterfaceLogins>
                        <Console>true</Console>
                        <WebUI>true</WebUI>
                        <API>true</API>
                </InterfaceLogins>
        </Role>
</BESAPI>
```

Update Site

Updates an existing site on the server with the contents of `site.xml`.

A list of sites to be updated can be inspected by via GET /api/sites.

- `site type` is type of the site (Master Action, External, Custom, Operator).
- `site name` is the name of the site to be updated.

cURL

```
curl -X PUT --data-binary @site.xml --user {username}:{password}
https://{server}:{port}/api/site/{site type}/{site name}
```

python

```
import requests
with open('site.xml', 'rb') as xml:
    r = requests.put('https://{server}:{port}/api/site/{site type}/{site
 name}', auth=('{username}',
'{password}'), data=xml)
    print(r.text)
```

Input

site.xml

```
<?xml version="1.0" encoding="utf-8"?>
<BES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BES.xsd">
        <CustomSite>
                <Name>TestSite</Name>
                <GatherURL>http://bigfix.test:52311/cgi-bin/bfgather.exe/
CustomSite_TestSite</GatherURL>
                <Description></Description>
                <Domain>BESC</Domain>
                <GlobalReadPermission>true</GlobalReadPermission>
                <Subscription>
                        <Mode>None</Mode>
                </Subscription>
        </CustomSite>
</BES>
```

Upload Files

Upload one or more files to the server.

cURL

Single File

```
curl -X POST -F file=@file.xml --user {username}:{password} https://
{server}:{port}/api/upload
```

Multiple Files

```
curl -X POST -F file=@file1.xml -F file=@file2.xml --user {username}:
{password} https://{server}:
{port}/api/upload
```

python

Single File

```
import requests
with open('file.xml', 'rb') as xml:
    r = requests.post('https://{server}:{port}/api/upload',
 auth=('{username}', '{password}'),
file=xml)
    print(r.text)
```

Multiple Files

```
import requests
filelist = [('file', open('file1.xml', 'rb')), ('file', open('file2.xml',
 'rb'))]
r = requests.post('https://{server}:{port}/api/upload', auth=('{username}',
 '{password}'),
files=filelist)
print(r.text)
```

Chapter 9. IEM Command-Line Interface

This command-line interface (CLI) is a lightweight wrapper for user authentication, session management, HTTP request and response generation, and parsing. A request initiated by the IEM CLI is sent to the REST API for processing.

Note: The IEM CLI does not support the BigFix FIPS-compliant cryptography library. If you must use this capability in your REST API HTTP environment, use a different FIPS-capable REST Client application.

Executable Files and Directories

The IEM CLI installed with the BigFix server installer as:

- %PROGRAM FILES%\BigFix Enterprise\BES Server\IEM CLI\iem.exe on Windows systems
- /opt/BESServer/bin/iem on Linux Red Hat Enterprise V.5.0 (or later) systems

To run the IEM CLI executable on a system different from the BigFix server, copy the IEM CLI executable and the libBEScrypto_1_0_0_4 library into the same directory on the target machine.

IEM CLI Data Directory

The IEM CLI uses a local directory, named .iem for local caching. This directory contains:

- A configuration file.
- A directory tree of cached server certificates.
- The BigFix CLI session credentials.

By default the .iem directory is created in the user profile directory:

- *%LOCALAPPDATA%\BigFix* on Windows systems

- */usr/{user}* on Linux systems. This means that, by default, a single OS user can use only a single BigFix account.

If you want to allow a single OS user to connect using different BigFix accounts, set the **IEM_DATADIR** environment variable to save the session credentials in a different location.

If a console is installed on the local Windows machine, any server certificate that was trusted by the console is implicitly trusted by the CLI as well.

User Authentication and Session Management

To start the command line interface, first log in to the server with the following command from a command prompt:

```
iem LOGIN --server <bigfix_server> --user <operator_name> --password
 <operator_password>
[--masthead <path_to_masthead>]
```

If the server uses a self-signed certificate for HTTPS interactions, at the first login you are prompted to accept or decline the certificate. If you choose to trust it, the certificate is cached in the local data directory and used to validate all future interactions with the server. You can use the --masthead argument to prevent displaying the certificate trust prompt. In this case, the masthead file is copied into the local cache directory and used for all future interactions with the server.

Upon a successful login, the server provides the IEM CLI utility with a session token that lasts, by default, 5 minutes. After 5 minutes of inactivity, the session token expires and you must log in again to the IEM CLI. You can customize the duration of the session token by configuring the **_BESDataServer_APIAuthenticationTimeoutMinutes** setting on the server and then restarting the server.

Updating the root server certificate

If the root server certificate has changed, for example because the server signing certificate was rotated, as described in Generating a new encryption key, the authentication might fail with the following error message:

```
The server's SSL certificate is not trusted.
```

In this case, to update the certificate, delete the IEM CLI data directory folder and then authenticate again.

The IEM CLI data directory folder is:

- `/root/.iem` on Linux systems
- `%LOCALAPPDATA%\BigFix` on Windows systems

Warning: If the certificate was not rotated, this error might indicate that a man-in-the-middle attack is trying to compromise your password.

Running Requests from the IEM CLI

This is the basic syntax for constructing requests using the IEM CLI:

```
iem <METHOD> <RESOURCE> [-q] [--param <value>]
```

where:

METHOD

Is one of the HTTP methods supported by that specific REST API resource.

Depending on the method, the form of the request can be any of the following:

```
iem <GET|DELETE> <RESOURCE> [-q] [--outFile FILE] [--param
 value]
iem <POST|PUT> [inputFile] <RESOURCE> [-q] [--outFile FILE] [--
param value]
iem login [-q] [--server=SERVER[:PORT]] [--
windowsAuthentication]
[--masthead=PATH_TO_TRUSTED_MASTHEAD]
iem login [-q] [--server=SERVER[:PORT]] [--user=USER] [--
password=PASS]
[--masthead=PATH_TO_TRUSTED_MASTHEAD]
```

where inputFile and the file referenced by --outFile are XML files containing resources descriptions as described in the schema files. POST and PUT requests require a body in their HTTP Requests. You can specify the body either as an input file on the command line, such as:

```
iem POST inputfile.xml operator/bigfix
```

or, you can enter it manually when prompted, for example:

```
iem POST query
```

```
Input: relevance=now
```

Because the IEM CLI does not do any pre-parsing or sanity checking on the requests issued, you must ensure that the input uses the format specified in the RESTAPI resource description.

RESOURCE

Is one of the REST API Resources.

-q

Means quiet mode. Set it to disable the input prompts.

--param

Is used to specify optional parameters that might be requested by the specific *RESOURCE* and *METHOD*. For example, if you want to use the RESTAPI resource **/api/query**, which requires the parameter **relevance=**, you can write:

```
iem GET query --relevance "names of bes computers"
```

IEM CLI Samples

Here you find usage samples showing how to run requests on REST API resources from the IEM CLI.

Actions

To submit the Fixlet ID 42 in the Master Action Site, on the computer my_computer.my_domain.com, create an XML file as follows:

```
<BES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BES.xsd">
      <SourcedFixletAction>
        <SourceFixlet>
            <Sitename>ActionSite</Sitename>
            <FixletID>42</FixletID>
            <Action>Action1</Action>
        </SourceFixlet>
        <Target>
            <ComputerName>my_computer.my_domain.com</ComputerName>
        </Target>
      </SourcedFixletAction>
</BES>
```

Use the following command to post the action of submitting the Fixlet on a specific computer:

```
./iem post /BigFix/take_action_site.xml actions
```

Advanced Options

To get the list of advanced options, run the following command:

```
./iem post /BigFix/take_action_site.xml actions
```

The command returns the list of fields in XML format as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
            <AdminField Resource="https://my_computer.my_domain.com:52311/
api/admin/field/
             usePre70ClientCompatibleMIME">
```

```
                 <Name>usePre70ClientCompatibleMIME</Name>
                 <Value>false</Value>
           </AdminField>
```

To set the admin key disableNmoSiteManagementDialog, create an XML file (besadmin.xml) as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
           <AdminField Resource="https:/my_computer.my_domain.com:52311/
api/admin/
            field/disableNmoSiteManagementDialog">
                 <Name>disableNmoSiteManagementDialog</Name>
                 <Value>1</Value>
           </AdminField>
</BESAPI>
```

Use the following command to set the appropriate attribute:

```
./iem post /BigFix/besadmin.xml admin/fields
```

Export Masthead

Use the following command to export the masthead to standard output:

```
./iem get admin/masthead
```

Use the following command to retrieve masthead parameters:

```
./iem get admin/masthead/parameters
```

The command returns the list of parameters in XML format as follows:

```
<BESAPI xmlns:xsi="http://www.w3.org/2001
          /XMLSchema-instance" xsi:noNamespaceSchemaLocation="BESAPI.xsd">
            <MastheadParameters Resource="https://
my_computer.my_domain.com:52311
```

```
             /api/admin/masthead/parameters">
                <PortNumber>52311</PortNumber>
                <GatherInterval>Day</GatherInterval>
                <Controller>nobody</Controller>
                <InitialLockState>on</InitialLockState>
                <RequireFIPSCompliantCrypto>false</
RequireFIPSCompliantCrypto>
           </MastheadParameters>
</BESAPI>
```

Fixlet

To get the list of Fixlets in the custom site myfixes, use the following command:

```
./iem get fixlets/custom/myfixes
```

The command returns the list of Fixlets in XML format as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
      <Fixlet Resource="https://my_computer.my_domain.com:52311/api/fixlet/
custom/myfixes/34?"
       LastModified="Mon, 10 Dec 2012 14:33:36 +0000">
              <Name>myfixes Custom Fixlet</Name>
              <ID>34</ID>
      </Fixlet>
      <Fixlet Resource="https://my_computer.my_domain.com:52311/api/fixlet/
custom/myfixes/40?"
       LastModified="Mon, 10 Dec 2012 16:05:30 +0000">
              <Name>MyFixlet</Name>
              <ID>40</ID>
      </Fixlet>
</BESAPI>
```

LDAP

To get the list of defined LDAPs, use the following command:

```
./iem get ldapdirectories
```

The command returns the list of LDAP in XML format as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
    <LDAPDirectory Resource=" https://my_computer.my_domain.com:52311/
ldapdirectory/34">
          <ID>34</ID>
          <Name>AD</Name>
          <IsActiveDirectory>true</IsActiveDirectory>
          <IsGlobalCatalog>true</IsGlobalCatalog>
          <UseSSL>false</UseSSL>
          <BaseDN>DC=tem,DC=test,DC=com</BaseDN>
          <UIDAttribute>userPrincipalName</UIDAttribute>
          <UserFilter>(objectCategory=user)</UserFilter>
          <GroupFilter><![CDATA[(&(objectCategory=group)
 (groupType:1.2.840.113556.1.4.803:=2147483648))]]>
          </GroupFilter>
          <User>BigFix\Administrator</User>
          <Servers>
            <Server>
                <Host>10.43.5.20</Host>
                <Port>3268</Port>
                <Priority>0</Priority>
            </Server>
          </Servers>
    </LDAPDirectory>
```

To create a new LDAP, use the same XML syntax as ./iem get ldapdirectories and add the following row after the User row in the XML file:

```
<Password>MyLDAP-Password</Password>
```

Then create the new LDAP with the following command:

```
./iem post MyLDAP.xml ldapdirectories
```

To get the configuration data of a specific LDAP having its ID (in the current example ID=34) run the following command:

```
./iem get ldapdirectory/34
```

The command returns the LDAP configuration in XML format as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
      <LDAPDirectory Resource="https://my_computer.my_domain.com:52311/
ldapdirectory/34">
             <ID>34</ID>
             <Name>AD</Name>
             <IsActiveDirectory>true</IsActiveDirectory>
             <IsGlobalCatalog>true</IsGlobalCatalog>
             <UseSSL>false</UseSSL>
             <BaseDN>DC=tem,DC=test,DC=com</BaseDN>
             <UIDAttribute>userPrincipalName</UIDAttribute>
             <UserFilter>(objectCategory=user)</UserFilter>
             <GroupFilter><![CDATA[(&(objectCategory=group)
 (groupType:1.2.840.113556.1.4.803:=2147483648))]]>
             </GroupFilter>
             <User>BigFix\Administrator</User>
             <Servers>
                     <Server>
                            <Host>10.43.5.20</Host>
```

```
                             <Port>3268</Port>
                             <Priority>0</Priority>
                     </Server>
              </Servers>
       </LDAPDirectory>
```

To remove a specific LDAP having its ID (in the current example ID=34) run the following command:

```
./iem delete ldapdirectory/34
```

To convert a local operator into an LDAP operator, run the following command:

```
BESAdmin.exe /convertToLDAPOperators [/mappingFile:<file>]
```

where `<file>` is the mapping file containing the match between Windows local operators and LDAP operators. Each line of the file must contain the name of the user to convert, followed by a tab and the name of the user in LDAP or Active Directory. The LDAP name must have the same format used to log into the console, such as domain\user, user@domain, or user. If the file is not available, BESAdmin converts all local users assuming their name in LDAP or Active Directory is the same as their local user name.

Login

To log in, run the command:

```
./iem login --server=ServerName:ServerPort --user=master --
password=Mypassword
```

To perform an https login:

```
./iem login --server=https://my_computer.my_domain.com:52311 --user=master
--password=Mypassword
```

Operators

To display a list of operators (local and LDAP), run the following command:

```
./iem get operators
```

To get roles associated to an operator, run the following command:

```
./iem get operator/OperatorName/roles
```

To add an operator, use the XML syntax example from ./iem get operators, remove the row . For a local operator, add the row , and then run the following command:

```
./iem post MyOperator.xml operators
```

To modify an operator, use the XML syntax example from ./iem get operators, and then run the following command:

```
./iem post /tmp/Operator.xml operator/MyOperatorName
```

To remove an operator (local and LDAP), run the following command:

```
./iem delete operator/OperatorName
```

DSA Replication

You can change the replication interval and the master server of your replication servers by using the command line.

Replication interval changes

To change the replication interval, complete the following steps:

1. Start the command line:

   On Windows systems:

   ```
   iem login --server=servername:serverport --user=username --
   password=password
   ```

   On Linux systems:

   ```
   ./iem login --server=servername:serverport --user=username --
   password=password
   ```

2. Retrieve the replication server settings by running the following command:

   On Windows systems:

```
iem get replication/server/0 > c:\temp\replicationServer0.xml
```

On Linux systems:

```
./iem get replication/server/0 > /appo/replicationServer0.xml
```

3. Edit the following keyword of the replicationServer0.xml file:

```
<ReplicationIntervalSeconds>300</ReplicationIntervalSeconds>
```

to change the value in seconds of the replication interval. Using longer replication intervals means that the servers replicate data less often, but have more data to transfer each time.

This is an example of the replicationServer0.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
 <BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
    <ReplicationServer Resource="http://
my_computer.my_domain.com:52311/api/replication/server/0">
            <ServerID>0</ServerID>
            <URL>http://my_computer.my_domain.com:52311</URL>
            <DNS>my_computer.my_domain.com</DNS>
       <ReplicationIntervalSeconds>300</ReplicationIntervalSeconds>
       <ReplicationLink Resource="http://
my_computer.my_domain.com:52311/api/replication/server/0/link/3">
               <SourceServerID>0</SourceServerID>
               <DestinationServerID>3</DestinationServerID>
               <Weight>1</Weight>
               <IsConnected>0</IsConnected>
               <LastReplication>Fri, 01 Mar 2013 11:17:12 +0000
               </LastReplication>
               <LastError>19NoMatchingRecipient - Fri, 01 Mar 2013
 11:17:12 +0000
               </LastError>
```

```
        </ReplicationLink>
        <ReplicationLink Resource="http://
my_computer.my_domain.com:52311/api/replication/server/3/link/0">
                    <SourceServerID>3</SourceServerID>
                    <DestinationServerID>0</DestinationServerID>
                    <Weight>1</Weight>
                    <IsConnected>1</IsConnected>
                    <LastReplication>Fri, 01 Mar 2013 11:17:18 +0000
                    </LastReplication>
        </ReplicationLink>
     </ReplicationServer>
 </BESAPI>
```

4. Upload the modified file by running the following command:

   On Windows systems:

   ```
   iem post c:\temp\replicationServer0.xml  replication/server/0
   ```

   On Linux systems:

   ```
    ./iem post /appo/replicationServer0.xml  replication/server/0
   ```

Master Server Switch

By default, server 0 (zero) is the master server. To switch the master to another server, set the deployment option masterDatabaseServerID to the other server ID as follows:

1. Start the command line:

   On Windows systems:

   ```
   iem login --server=servername:serverport --user=username --
   password=password
   ```

   On Linux systems:

```
 ./iem login --server=servername:serverport --user=username --
password=password
```

2. Retrieve the settings to switch the master server:

   On Windows systems:

   ```
    iem get admin/fields > c:\temp\switchmaster.xml
   ```

   On Linux systems:

   ```
     ./iem get admin/fields > /appo/switchmaster.xml
   ```

   In the switchmaster.xml file, add or edit the following keyword and its value:

   ```
       <Name>masterDatabaseServerID<Name>
       <Value>0</Value>
   ```

   to switch the master server to another master server:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
    <BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
     noNamespaceSchemaLocation="BESAPI.xsd">
        <AdminField Resource="http://my_computer.my_domain.com:52311/
        api/admin/field/masterDatabaseServerID">
           <Name>masterDatabaseServerID</Name>
           <Value>3</Value>
        </AdminField>
    </BESAPI>
   ```

3. Upload the modified file by running the following command:

   On Windows systems:

   ```
   iem post c:\temp\switchmaster.xml admin/fields
   ```

   On Linux systems:

   ```
   ./iem post /appo/switchmaster.xml admin/fields
   ```

After the value has successfully replicated to the new server, it become the master server. If a server has a failure while it is the master, another server must be made the master server by direct manipulation of the ADMINFIELDS table in the database.

Role

To get the role configuration, run the following command:

```
./iem get roles
```

The command returns the role configuration in XML format.

To create a new role, run the following command:

```
./iem post Example.xml roles
```

Where Example.xml contains role configuration data in XML format.

Site

To add a file to a custom site as master operator, run the following command:

```
./iem post /tmp/file.my /api/site/custom/test/file/file.my
```

To add a file to the master action site as master operator, run the following command:

```
iem post /tmp/file.my /api/site/master/file/file.my --force=yes
```

Where the option `--force=yes` allows to override the file it if already exists in the site. To add a file to the operator site as non master operator, run the following command:

```
iem post /tmp/file.my /api/site/operator/nmo/file/file.my
```

Where the option `--isClient=yes` allows to make the file available for download by Clients.

The options `--force` and `--isClient` apply when running post and put requests against files, both as master and as non master operator.

System Options

To display MinimumRefreshSeconds (seconds), and DefaultFixletVisibility (Visible, Hidden) run the following command:

```
./iem get admin/options
```

The command returns the list of options in XML format as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
        <SystemOptions Resource="https://my_computer.my_domain.com:52311/
api/admin/options">
             <MinimumRefreshSeconds>15</MinimumRefreshSeconds>
             <DefaultFixletVisibility>Visible</DefaultFixletVisibility>
       </SystemOptions>
</BESAPI>
```

To set the system option MinimumRefreshSeconds, create an XML file (SystemOptions.xml) as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<BESAPI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="BESAPI.xsd">
        <SystemOptions Resource="https://my_computer.my_domain.com:52311/
api/admin/options">
             <MinimumRefreshSeconds>20</MinimumRefreshSeconds>
             <DefaultFixletVisibility>Hidden</DefaultFixletVisibility>
        </SystemOptions>
</BESAPI>
```

Use the following command to set the appropriate attribute:

```
./iem post /BigFix/SystemOptions.xml admin/options
```

Chapter 10. Support

For more information about this product, see the following resources:

- BigFix Support Portal
- BigFix Developer
- BigFix playlist on YouTube
- BigFix Tech Advisors channel on YouTube
- BigFix Forum

Notices

Trademarks

Terms and conditions for product documentation

Applicability

Personal use

Commercial use