

**Web Service API Guide**

**PowerEditor 5.10**

Revised: February 27, 2014

Re-Released: September 7, 2018

Notice: This document includes confidential and proprietary data that is within the scope of non-disclosure agreements between MindBox and its alliances. The document shall not be duplicated, used or disclosed - in whole or in part - for any purpose other than that specified within the scope of such an agreement. Acceptance of this document presumes adherence to the above conditions.

The PowerEditor documentation suite includes the following materials:

PowerEditor Business Analysts Guide

PowerEditor Custom Reports Guide

PowerEditor Installation and Customization Guide

PowerEditor Reference Manual

PowerEditor Release Notes

PowerEditor Rule Writers Toolkit

PowerEditor Web Service API Guide

*Release Notes* for earlier versions are also provided in the release package. For technical assistance with upgrading or any other PowerEditor-related issue please contact CoreLogic Technical Support at 1-855.369.2410, or [ADESupport@corelogic.com](mailto:ADESupport@corelogic.com)

**Table of Contents**

1. PowerEditor Web Services API 4

1.1 Test with Sample Client 4

1.1.1 Runnig Sample Client 5

1.1.2 Call *Ping* with Sample Client 5

1.1.3 Call Other Functionswith Sample Client 5

1.1.4 Client Filter Properties File 6

1.2 Build Test Client 6

1.2.1 Building Java Client 7

1.2.2 Building C# Client 7

1.2.3 Building a Different Client 7

1.3 Build Real Client 8

1.4 Client API Security 8

1.4.1 HTTP Basic Authentication 8

1.4.2 Credentials As Method Arguments 8

1.4.3 HTTPS 8

1.5 Client API Technical Specifications 8

1.5.1 Deploy API 9

1.5.2 Importing Entities API 11

1.5.3 Export API 12

1.5.4 Ping 15

# PowerEditor Web Services API

The PowerEditor server is architected to allow custom-built client applications to programmatically access certain PowerEditor functions. For example, your site might want to initiate automatic nightly rule deploys from a custom built application rather than from the PowerEditor's manual user interface. Similarly, you might want to automatically synchronize your externally defined product database with PowerEditor product entities.

This chapter documents the application programming interface (API) provided by the PowerEditor for use by external client applications. Note that this section describes the API for data *updates and functions* only. Data *queries* can be performed by the Reporting API, as described in the document entitled *PowerEditor Custom Reports Guide*.

The PowerEditor currently provides external interfaces to the following functionality:

* *Entity Import* – Entities can be imported into the PowerEditor database, allowing you to synchronize externally stored data with the PowerEditor database.
* *Rule Deploy* – A rule deploy can be initiated from your external client application
* *Data Export* – Exporting data of any type can be initiated from your external application. For example, your application can initiate an entity export or a guideline export.
* *Ping* – Verify PowerEditor is running

Any external client that wishes to update PowerEditor data will need to do so by accessing the PowerEditor **Web Service** interface. Web Services are independent of language, so the PowerEditor Web Service can be accessed from different types of clients. The PowerEditor Web Service has been specifically tested with Java and C# clients, and sample code for each of these clients is provided with the PowerEditor software distribution.

To proceed the following approach is recommended for getting started using the Web Services API

1. *Test with Sample Client -* Use the provided sample client application to test a round-trip client/server request
2. *Build Sample Client with Ping -* Build or enhance your own sample client application to call the PowerEditor server with a simple *ping* request
3. *Build Real Client -* Enhance your own client application to call the PowerEditor server with desired API calls (e.g. *deploy, export, importEntities)*
4. *Add Security -* Change the server and client security as required for your installation.

Each step listed above is described in its own section below.

## Test with Sample Client

As mentioned above, the recommended first step in trying out the Web Service API is to use the provided sample client application to test a round-trip client/server request. The distributed client application is a Java client. You'll need to run this client application on the same machine that runs the PowerEditor server.

The PowerEditor distribution zip file contains a folder called *tool*. For the purpose of this discussion, let's assume you unzipped this into a folder called /mindbox/PowerEditor/tool.

### Runnig Sample Client

To run the web service client, open command prompt, go to the directory where PEWSClient.jar is located (in the Tool directory in the distribution zip file), and execute:

java –jar PEWSClient.jar <URL> <command> [arguments]

The following table summarizes the available commands and their:

|  |  |  |
| --- | --- | --- |
| **Command** | **Arguments** | **Description** |
| DEPLOY | [user password] | Performs deployment. |
| EXPORT | <filename> [user password] | Exports PE KB into the specified file. |
| IMPORT | <filename> <merge> [user password] | Imports the specified file into PE. The <merge> flag indicates if the imported data should be merged with existing data or not. |
| PING | [user password] | Pings the PE server. |

The first argument to the client, <URL>, is PowerEditor web service URL. It’s in the format of http://<server>:<port>/powereditor.

The client takes in username and passsword for authentication. Note that if user or password is not provided, demo/demo will be used.

The user must have WebService Access privilege. The password must be encrypted using the password tool. To obtain an encrypted password,

1. Run PE password tool by executing: java –jar /mindbox/PowerEditor/tool/PEPasswordTool.jar
2. Enter the clear text password in the Password field.
3. Make sure MD5 One Way Hash is selected in the Encryption Type field.
4. Click the Encrypt button. The encrypted password will be displayed in the Encrypted field.

### Call *Ping* with Sample Client

To call the PowerEditor server *ping* function from the sample Java client, perform these steps:

* Make sure the PowerEditor server is running
* On the PowerEditor server machine, open a command window (e.g. a DOS window or shell window)
* Change to the tool directory
* Run the test client by typing

run-wsclient <URL> ping

* Verify that the client returns a success message. For example,

[RESULT] Ping command received and processed.

### Call Other Functionswith Sample Client

After you've successfully called *ping*, you might want to try calling some of the other API functions using the sample client. Although this step is optional, it could prove a useful debugging technique as you build and debug your own client application. Here are examples of other functions you can invoke using the sample API from the command line:

* run-wsclient <URL> ping
* run-wsclient <URL> import *<filename> <merge> [username] [password]*
* run-wsclient <URL> export <filename> *[username] [password]*
* run-wsclient <URL> deploy *[username] [password]*

If you type in run-wsclient all by itself (with no arguments) documentation for this tool will be displayed.

### Client Filter Properties File

Note that the API for import, export, and deploy functions includes many optional parameters, as described in *Section 1.5* Client API Technical Specifications. For example, you could specify what usage-types to deploy, or what activation dates to export.

These parameters can be passed into the run-wsclient application by editing the following file: pewsclient.properties. The following table summarizes supported properties in the file:

|  |  |
| --- | --- |
| **Property** | **Description** |
| includeEntities | Set to true to include entities for DEPLOY and EXPORT. |
| includeSecurity | Set to true to include security date for DEPLOY and EXPORT. |
| includeGuidelines | Set to true to include guidelines for DEPLOY and EXPORT. |
| includeParameters | Set to true to include parameters for DEPLOY and EXPORT. |
| includeTemplates | Set to true to include templates for DEPLOY and EXPORT. |
| includeGuidelineActions | Set to true to include guideline actions for DEPLOY and EXPORT. |
| includeTestConditions | Set to true to include test conditions for DEPLOY and EXPORT. |
| includeDateSynonyms | Set to true to include date synonyms for DEPLOY and EXPORT. |
| includeEmptyContexts | If set to true, guidelines with empty context will be included for DEPLOY and EXPORT. |
| includeParentCategories | If set to true, parent categories will be included when filter for the specified context elements for DEPLOY and EXPORT. |
| includeChildrenCategories | If set to true, child categories will be included when filter for the specified context elements for DEPLOY and EXPORT. |
| contextElements | Context elements to filter data. |
| includePolicies | Set to true to include policy data for DEPLOY and EXPORT. |
| includeProcessData | Set to true to include process data for DEPLOY and EXPORT. |
| includeCBR | Set to true to include CBR data for DEPLOY and EXPORT. |
| useDaysAgo | If set to true, filter data by the specified daysAgo for DEPLOY and EXPORT. |
| daysAgo | If useDaysAgo is set to true, only the data that’s newer than the specified days will be include for DEPLOY and EXPORT. |
| activeOnDate | If specified, only the data that is effective on the specified date will be included for DEPLOY and EXPORT. |
| status | Status to filter data. |
| guidelineTemplateIDs | Guideline template ids to filter data. |
| paramTemplateIDs | Parameter template ids to filter data. |
| usageTypes | Usage types to filter data. |
| credentialsAsArgs | If set to true, credentials will be passed as arguments when invoking for DEPLOY, EXPORT, and IMPORT. |

## Build Test Client

After you've successfully invoked *ping* with the distributed sample client application, you'll want try invoking *ping* from your own client application. This document assumes that you know how to build a client application in the language you desire, and this section is intended to demonstrate how the *ping* function could be invoked from your client.

### Building Java Client

To build a Java client, you'll need the following:

* Java JDK 1.6.0 or higher

Source code for a sample Java application is provided with the PowerEditor distribution. This file is included in the PowerEditor distribution zip file at *wsclient/java/com/mindbox/test/pe/webservices/client/PEWSClient.java*. You'll want to examine this file at this time. The interesting lines of code to look at in this file are:

**import** com.mindbox.pe.test.wsdl.api.PowerEditorAPIInterface;

**import** com.mindbox.pe.test.wsdl.api.PowerEditorAPIInterfaceService;

**import** com.mindbox.pe.test.wsdl.api.PowerEditorInterfaceReturnStructure;

PowerEditorAPIInterface peAPIInterface =

new PowerEditorAPIInterfaceService().getPowerEditorAPIInterfacePort();

**return** peAPIInterface.ping(**new** PingRequest()).getStatus();

When building your application, you will need to generate client code from a WSDL file. The PowerEditor Web Service WSDL file is included in the distribution zip file at *wsdl/PowerEditorAPIInterfaceService.wsdl*. The sample client code uses client code generated from the WSDL file using wsimport java utility. See JDK documentation for details on wsimport utility.

### Building C# Client

Source code for a sample C# application is provided with the PowerEditor distribution. This is in the PowerEditor.zip file under *wsclient/csharp/PEWSClient.cs*. You'll want to examine this file and (other files in this directory) at this time. The interesting lines of code to look at in this file are:

localhost.PowerEditorAPIInterfaceService CallWebService =

new localhost.PowerEditorAPIInterfaceService();

PEWSClient.localhost.powerEditorInterfaceReturnStructure pingResult = CallWebService.ping();

Console.WriteLine("Result content: {0}", pingResult.content);

When building your application, you will need to make sure that the WSDL location is set properly. This is the location of the **W**eb **S**ervices **D**efinition **L**anguage definition that can be accessed when PowerEditor is installed and is running. The following is an example of setting the WSDL location to a PowerEditor server running on localhost on port 8080:

http://localhost:8080/powereditor/PowerEditorWebService?wsdl

The general format of the WSDL location is

http://*host***:***port*/powereditor/PowerEditorWebService?wsdl

For example, if you create a new “Console Application”, in the right pane (Solution Explorer) click the right button and select "Add Web Reference". Enter the WSDL location as described above.

### Building a Different Client

Read the sections above to get a general feel for things, then use the functions specified *Section 1.5 Client API Technical Specifications*. Note that the web service WSDL file is included in the distribution zip file at *wsdl/PowerEditorAPIInterfaceService.wsdl*.

## Build Real Client

After you've built a client that calls *ping* you are ready to call the real API function you want to use. Specifically you can write your client to call one of the following functions:

* deploy
* importEntities
* exportData

As mentioned above, these functions can be called with filtering options, such as usage-types, dates, or status values. These are all described in *Section 1.5 Client API Technical Specifications*.

Note that the functions listed above require username and encrypted passwords, which the level of encryption you'll want as you get started. After you get these calls to work as you'd like, then you'll then consider what security features best suit your application. Security is described in the following section.

## Client API Security

As discussed above, the PowerEditor WebService API requires username and encrypted password. The API supports the following security options:

* HTTP Basic Authentication
* Credentials As Method Arguments
* HTTPS Encryption

### HTTP Basic Authentication

The PowerEditor Web Service supports HTTP basic authentication, in which credentials are provided in the HTTP header when making web service calls. With HTTP basic authentication, your client should call the “credentials-free” API methods. For example, you should call the function deploy rather than deployWithCredentials. Calling the “credentials-free” methods without HTTP basic authentication will result in an error.

### Credentials As Method Arguments

If use of HTTP basic authentication is not desired, you may use methods that accept credentials as method arguments. For example, you should call deployWithCredentialsrather than deploy.

### HTTPS

For added security, you may require adding a “certificate” for use with the application server and setting it up to listen for secure socket requests for HTTPS. This level of security is rarely required, since the PowerEditor API will rarely be required over a public DNS. Although the PowerEditor Web Service API can be run with HTTPS, it requires an application container change, and is not documented here.

## Client API Technical Specifications

If you're using HTTP basic authentication, then the following methods are available to you:

* deploy
* importEntities
* exportData
* ping

If you're not using HTTP basic authentication, then the following methods are available to you:

* deployWithCredentials
* importEntitiesWithCredentials
* exportDataWithCredentials
* ping

The remainder of this section lists the function headers for these methods.

### Deploy API

If you'd like to initiate the deployment of rules and/or parameters from your client, then use either the deploy or deployWithCredentials method. Invoking this method is equivalent to hitting the *Deploy* button from the PowerEditor user interface – output files will be generated as specified by your PowerEditor configuration file.

Note that sometimes the deploy method can take a long time. This is a blocking, synchronous method, so the method won't return until the deploy process is complete.

<xs:complexType name=*"deployRequest"*>

<xs:sequence>

<xs:element name=*"status"* type=*"xs:string"* minOccurs=*"0"* />

<xs:element name=*"usageTypes"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"unbounded"* />

<xs:element name=*"deployGuidelines"* type=*"xs:boolean"* />

<xs:element name=*"guidelineTemplateIDs"* type=*"xs:int"* minOccurs=*"0"* maxOccurs=*"unbounded"* />

<xs:element name=*"deployParameters"* type=*"xs:boolean"* />

<xs:element name=*"paramTemplateIDs"* type=*"xs:int"* minOccurs=*"0"* maxOccurs=*"unbounded"* />

<xs:element name=*"useDaysAgo"* type=*"xs:boolean"* />

<xs:element name=*"daysAgo"* type=*"xs:int"* />

<xs:element name=*"activeOnDate"* type=*"xs:string"* minOccurs=*"0"* />

<xs:element name=*"includeEmptyContexts"* type=*"xs:boolean"* />

<xs:element name=*"includeParentCategories"* type=*"xs:boolean"* />

<xs:element name=*"includeChildrenCategories"* type=*"xs:boolean"* />

<xs:element name=*"includeProcessData"* type=*"xs:boolean"* />

<xs:element name=*"includeCBR"* type=*"xs:boolean"* />

<xs:element name=*"includeEntities"* type=*"xs:boolean"* />

<xs:element name=*"contextElements"* type=*"xs:string"* minOccurs=*"0"* />

<xs:element name=*"exportPolicies"* type=*"xs:boolean"* />

</xs:sequence>

</xs:complexType>

<xs:complexType name=*"deployWithCredentialsRequest"*>

<xs:sequence>

<xs:element name=*"status"* type=*"xs:string"* minOccurs=*"0"* />

<xs:element name=*"usageTypes"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"unbounded"* />

<xs:element name=*"deployGuidelines"* type=*"xs:boolean"* />

<xs:element name=*"guidelineTemplateIDs"* type=*"xs:int"* minOccurs=*"0"* maxOccurs=*"unbounded"* />

<xs:element name=*"deployParameters"* type=*"xs:boolean"* />

<xs:element name=*"paramTemplateIDs"* type=*"xs:int"* minOccurs=*"0"* maxOccurs=*"unbounded"* />

<xs:element name=*"useDaysAgo"* type=*"xs:boolean"* />

<xs:element name=*"daysAgo"* type=*"xs:int"* />

<xs:element name=*"activeOnDate"* type=*"xs:string"* minOccurs=*"0"* />

<xs:element name=*"includeEmptyContexts"* type=*"xs:boolean"* />

<xs:element name=*"includeParentCategories"* type=*"xs:boolean"* />

<xs:element name=*"includeChildrenCategories"* type=*"xs:boolean"* />

<xs:element name=*"includeProcessData"* type=*"xs:boolean"* />

<xs:element name=*"includeCBR"* type=*"xs:boolean"* />

<xs:element name=*"includeEntities"* type=*"xs:boolean"* />

<xs:element name=*"contextElements"* type=*"xs:string"* minOccurs=*"0"* />

<xs:element name=*"exportPolicies"* type=*"xs:boolean"* />

<xs:element name=*"userID"* type=*"xs:string"* minOccurs=*"0"* />

<xs:element name=*"password"* type=*"xs:string"* minOccurs=*"0"* />

</xs:sequence>

</xs:complexType>

The arguments to the deploy methods closely match the deploy options presented in the PowerEditor user interface. These arguments are as follows:

status This string must be the internal name of a valid PowerEditor status. The PowerEditor will deploy guidelines and parameters of all statuses at this level and above.

usageTypes List of usage types. Empty value indicates that all guidelines for all templates shall be deployed, unless deployGuidelines is set to false.

deployGuidelines true/false – The value true indicates that guidelines shall be deployed.

guidelineTemplateIDs List of guideline template IDs. A null value indicates that all guidelines for all templates shall be deployed, unless deployGuidelines set to false.

deployParameters true/false – The value true indicates that parameters shall be deployed.

paramTemplateIDs List of guideline template IDs. A null value indicates that all guidelines for all templates shall be deployed, unless deployGuidelines=false.

useDaysAgo true/false – whether or not the daysAgo argument will be used

daysAgo number of days before the current day that expired rules and parameters will be deployed.

activeOnDate A date string of the following format: yyyy-MM-dd hh:mm:ss. Only guidelines and parameters that are active on the given date will be deployed.

includeEmptyContexts This true/false argument is only relevant if the contextElements argument is used. A value of true specifies that guidelines with blank contexts should be included in the results.

includeParentCategories This true/false argument is only relevant if the contextElements argument is used. A value of true specifies that the resulting guidelines should include ones whose context contain parent categories of the provided context elements.

includeChildrenCategories This true/false argument is only relevant if the contextElements argument is used. A value of true specifies that the resulting guidelines should include ones whose context contain children categories/ entities of the provided context elements.

includeProcessData This true/false argument specifies whether data from the Manage Process Data tab will be deployed. Process data includes phase and request objects. Note that many PowerEditor installations have this tab hidden, in which case the value of this argument doesn’t matter.

includeCBRData This true/false argument specifies whether data from the Manage CBR Data tab will be deployed. CBR is Case-Based Reasoning. Note that many PowerEditor installations have this tab hidden, in which case the value of this argument doesn’t matter.

includeEntities This true/false argument specifies whether the entities.xml file will be generated as part of this deployment.

contextElements This string of comma separated entity IDs narrows the list of guidelines returned to those whose context includes the specified entities/categories. If categories are specified the guidelines returned will be those that appear in *all* the categories specified (i.e. AND logic is used).

Each comma-separated value must be one of the following formats:

*<entity-type>*:**entity**:*<entity-name>*

*<entity-type>*:**category**:*<category-name>*

Example:

product:entity:FNMA Conforming Fixed 15

product:category:Fixed

product:category:Fixed,product:category:Conforming

If the category names in your application are not unique they can be fully qualified with the “->” separator. For example:

product:category:Root->AmTerm->Fixed

exportPolicies This true/false argument specifies whether policies export file is generated or not.

userID A user ID for a user that has deploy privileges. (This argument is only valid for the *deploy* method – not the *deployWithCredentials* method).

password The user's password, unencrypted. (This argument is only valid for the *deploy* method – not the *deployWithCredentials* method).

### Importing Entities API

If you would like to import entity definitions from your client into the PowerEditor database, use one of importEntities or importEntitiesWithCredentials. The entities to be imported are specified in the *content* argument to this method. The content is a string that is contains entity XML. The XSD specification for this XML string is specified in the following file: *xsd/PowerEditorData.xsd*. You should only need to use the <Entities> section of the XSD.

<xs:complexType name=*"importEntitiesRequest"*>

<xs:sequence>

<xs:element name=*"content"* type=*"xs:string"* minOccurs=*"0"* />

<xs:element name=*"merge"* type=*"xs:boolean"* />

</xs:sequence>

</xs:complexType>

<xs:complexType name=*"importEntitiesWithCredentialsRequest"*>

<xs:sequence>

<xs:element name=*"content"* type=*"xs:string"* minOccurs=*"0"* />

<xs:element name=*"merge"* type=*"xs:boolean"* />

<xs:element name=*"userID"* type=*"xs:string"* minOccurs=*"0"* />

<xs:element name=*"password"* type=*"xs:string"* minOccurs=*"0"* />

</xs:sequence>

</xs:complexType>

The arguments to the deploy methods closely match the deploy options presented in the PowerEditor user interface. These arguments are as follows:

content The content is a string that is contains entity XML. The XSD specification for this XML string is specified in the following file: *xsd/PowerEditorEntityData.xsd*.

merge true/false – The merge option provides a means for merging two completely independent KBs.

* The value **true** indicates that the imported values are treated independently from pre-existing values, which means that new identifiers will be created for all imported entities, ensuring that no overlapping IDs will occur.
* The value **false** indicates that you are not merging independent KBs, therefore entity IDs will be preserved. If you are importing an entity that uses the same ID as an entity that is already in the PowerEditor, the existing PowerEditor entity will be replacedwith the entity that is being imported.

userID A user ID for a user that has deploy privileges. (This argument is only valid for the *importEntities* method – not the *importEntitiesWithCredentials* method).

password The user's password, unencrypted. (This argument is only valid for the *importEntities* method – not the *importEntitiesWithCredentials* method).

### Export API

To have the PowerEditor generate an export XML file, use the exportData or exportDataWithCredentials methods. Most of the arguments are documented in the Deploy API section above.

Note that sometimes the export method can take a long time. This is a blocking, synchronous method, so the method won't return until the export process is complete.

<xs:complexType name=*"exportDataRequest"*>

<xs:sequence>

<xs:element name=*"exportEntities"* type=*"xs:boolean"* />

<xs:element name=*"exportSecurity"* type=*"xs:boolean"* />

<xs:element name=*"exportGuidelines"* type=*"xs:boolean"* />

<xs:element name=*"exportParameters"* type=*"xs:boolean"* />

<xs:element name=*"exportTemplates"* type=*"xs:boolean"* />

<xs:element name=*"exportGuidelineActions"* type=*"xs:boolean"* />

<xs:element name=*"exportTestConditions"* type=*"xs:boolean"* />

<xs:element name=*"exportDateSynonyms"* type=*"xs:boolean"* />

<xs:element name=*"includeEmptyContexts"* type=*"xs:boolean"* />

<xs:element name=*"includeParentCategories"* type=*"xs:boolean"* />

<xs:element name=*"includeChildrenCategories"* type=*"xs:boolean"* />

<xs:element name=*"status"* type=*"xs:string"* minOccurs=*"0"* />

<xs:element name=*"usageTypes"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"unbounded"* />

<xs:element name=*"guidelineTemplateIDs"* type=*"xs:int"* minOccurs=*"0"* maxOccurs=*"unbounded"* />

<xs:element name=*"paramTemplateIDs"* type=*"xs:int"* minOccurs=*"0"* maxOccurs=*"unbounded"* />

<xs:element name=*"useDaysAgo"* type=*"xs:boolean"* />

<xs:element name=*"daysAgo"* type=*"xs:int"* />

<xs:element name=*"activeOnDate"* type=*"xs:string"* minOccurs=*"0"* />

<xs:element name=*"contextElements"* type=*"xs:string"* minOccurs=*"0"* />

</xs:sequence>

</xs:complexType>

<xs:complexType name=*"exportDataWithCredentialsRequest"*>

<xs:sequence>

<xs:element name=*"exportEntities"* type=*"xs:boolean"* />

<xs:element name=*"exportSecurity"* type=*"xs:boolean"* />

<xs:element name=*"exportGuidelines"* type=*"xs:boolean"* />

<xs:element name=*"exportParameters"* type=*"xs:boolean"* />

<xs:element name=*"exportTemplates"* type=*"xs:boolean"* />

<xs:element name=*"exportGuidelineActions"* type=*"xs:boolean"* />

<xs:element name=*"exportTestConditions"* type=*"xs:boolean"* />

<xs:element name=*"exportDateSynonyms"* type=*"xs:boolean"* />

<xs:element name=*"includeEmptyContexts"* type=*"xs:boolean"* />

<xs:element name=*"includeParentCategories"* type=*"xs:boolean"* />

<xs:element name=*"includeChildrenCategories"* type=*"xs:boolean"* />

<xs:element name=*"status"* type=*"xs:string"* minOccurs=*"0"* />

<xs:element name=*"usageTypes"* type=*"xs:string"* minOccurs=*"0"* maxOccurs=*"unbounded"* />

<xs:element name=*"guidelineTemplateIDs"* type=*"xs:int"* minOccurs=*"0"* maxOccurs=*"unbounded"* />

<xs:element name=*"paramTemplateIDs"* type=*"xs:int"* minOccurs=*"0"* maxOccurs=*"unbounded"* />

<xs:element name=*"useDaysAgo"* type=*"xs:boolean"* />

<xs:element name=*"daysAgo"* type=*"xs:int"* />

<xs:element name=*"activeOnDate"* type=*"xs:string"* minOccurs=*"0"* />

<xs:element name=*"contextElements"* type=*"xs:string"* minOccurs=*"0"* />

<xs:element name=*"userID"* type=*"xs:string"* minOccurs=*"0"* />

<xs:element name=*"password"* type=*"xs:string"* minOccurs=*"0"* />

</xs:sequence>

</xs:complexType>

The arguments to the export methods closely match the deploy options presented in the PowerEditor user interface. These arguments are as follows:

exportEntities true/false – The value true indicates that entities shall be exported.

exportSecurity true/false – The value true indicates that security data shall be exported.

exportGuidelines true/false – The value true indicates that guidelines shall be exported.

exportParameters true/false – The value true indicates that parameters shall be exported.

exportTemplates true/false – The value true indicates that guideline templates shall be exported.

exportGuidelineActions true/false – The value true indicates that guideline actions shall be exported.

exportTestConditions true/false – The value true indicates that test conditions shall be exported.

exportDateSynonyms true/false – The value true indicates that date synonyms shall be exported.

includeEmptyContexts This true/false argument is only relevant if the contextElements argument is used. A value of true specifies that guidelines with blank contexts should be included in the results.

includeParentCategories This true/false argument is only relevant if the contextElements argument is used. A value of true specifies that the resulting guidelines should include ones whose context contain parent categories of the provided context elements.

includeChildrenCategories This true/false argument is only relevant if the contextElements argument is used. A value of true specifies that the resulting guidelines should include ones whose context contain children categories/ entities of the provided context elements.

status This string must be the internal name of a valid PowerEditor status. The PowerEditor will deploy guidelines and parameters of all statuses at this level and above.

usageTypes List of usage types. Empty value indicates that all guidelines for all templates shall be deployed, unless deployGuidelines is set to false.

guidelineTemplateIDs List of guideline template IDs. A null value indicates that all guidelines for all templates shall be deployed, unless deployGuidelines set to false.

deployParameters true/false – The value true indicates that parameters shall be deployed.

paramTemplateIDs List of guideline template IDs. A null value indicates that all guidelines for all templates shall be deployed, unless deployGuidelines=false.

useDaysAgo true/false – whether or not the daysAgo argument will be used

daysAgo number of days before the current day that expired rules and parameters will be deployed.

activeOnDate A date string of the following format: yyyy-MM-dd hh:mm:ss. Only guidelines and parameters that are active on the given date will be deployed.

contextElements This string of comma separated entity IDs narrows the list of guidelines returned to those whose context includes the specified entities/categories. If categories are specified the guidelines returned will be those that appear in *all* the categories specified (i.e. AND logic is used).

Each comma-separated value must be one of the following formats:

*<entity-type>*:**entity**:*<entity-name>*

*<entity-type>*:**category**:*<category-name>*

Example:

product:entity:FNMA Conforming Fixed 15

product:category:Fixed

product:category:Fixed,product:category:Conforming

If the category names in your application are not unique they can be fully qualified with the “->” separator. For example:

product:category:Root->AmTerm->Fixed

exportPolicies This true/false argument specifies whether policies export file is generated or not.

userID A user ID for a user that has deploy privileges. (This argument is only valid for the *export* method – not the *exportWithCredentials* method).

password The user's password, unencrypted. (This argument is only valid for the *export* method – not the *exportWithCredentials* method).

### Ping

To verify that PowerEditor is running use the Ping method. The Ping method takes no input.