

REST Gateway

**User Guide**

Updated: 2019-09-30

# Contents

[Contents 2](#_bookmark0)

[Introduction 4](#_bookmark1)

[Installing and configuring the REST Gateway 5](#_bookmark2)

[Product compatibility 5](#_bookmark3)

[Deploying the REST Gateway application package 5](#_bookmark4)

[Configuring the REST Gateway 6](#_bookmark5)

[Deploying the REST Gateway TomEE configuration 6](#_bookmark6)

[Modifying the BCP classpath 6](#_bookmark7)

[Creating a WS-AppServer service group 6](#_bookmark8)

[Assigning roles 7](#_bookmark9)

[Developing a REST example 8](#_bookmark10)

[Creating a sample SOAP service for map generation 8](#_bookmark11)

[Creating the FTP Connector service container 8](#_bookmark12)

[Creating an FTP configuration 8](#_bookmark13)

[Testing the FTP Service 8](#_bookmark14)

[Generating REST services 9](#_bookmark15)

[Testing Generated REST Services 9](#_bookmark16)

[Packaging and deploying a REST solution 10](#_bookmark17)

[Packaging the REST solution 10](#_bookmark18)

[Deploying the REST Solution 10](#_bookmark19)

[Additional information 11](#_bookmark21)

[Monitoring JMX Settings 11](#_bookmark22)

[Limitations 11](#_bookmark23)

[Troubleshooting 12](#_bookmark20)

[REST Gateway Developer Guide 12](#_bookmark24)

[Generating the REST services interface 12](#_bookmark25)

[Selecting the Web Service 13](#_bookmark26)

[Selecting the Web service operation 13](#_bookmark27)

[Updating the SOAP to REST parameter mappings 14](#_bookmark28)

[Using XSLT to transform SOAP to REST 16](#_bookmark29)

[Test REST Services interface 17](#_bookmark30)

[Known Issues and troubleshooting 19](#_bookmark31)

[JSON generation Issue with REST Gateway 2.0.0 (Java 7-8 version) 19](#_bookmark32)

[Classpath issues with REST Gateway 3.0.1 (Java 11 version) 19](#_bookmark33)

[Test REST services interface’s download button does not work 19](#_bookmark34)

[Request fails with http status code 405 Method Not Allowed 20](#_bookmark35)

[Contact Information 21](#_bookmark36)

# Introduction

REST (Representational State Transfer) is a modern architecture style that consumes and exposes resources over the web. This is an introductory guide to using the REST Gateway component on an OpenText AppWorks Platform system. The guide covers component configuration, packaging, and deployment of a REST solution.

# Installing and configuring the REST Gateway

This section describes the steps needed to deploy and configure the REST Gateway.

## Product compatibility

The REST Gateway is available in two versions to support Java 7-8 versions of Cordys BOP/AppWorks Platform and Java 11 version of AppWorks Platform. Additionally, there is a configuration file needed to support URL rewrite..

|  |  |
| --- | --- |
| **Download** | **Compatibility** |
| REST Gateway 2.0.0.cap | Cordys BOP 4.2 – OpenText AppWorks 16.5 (Java 7-8) |
| REST Gateway 3.0.1.cap | OpenText AppWorks Platform 16.6+ (Java 11) |
| REST Gateway TomEE conifg.tar (URL rewrite support files) | All |

## Deploying the REST Gateway application package

The REST Gateway component is a community connector. As a result, the application package is not signed and cannot be deployed using default security settings. AppWorks Platform (<instance name>) must be configured to accept the installation of unsigned application packages. OpenText highly recommends changing your settings to Prompt (rather than Allow).

### To allow unsigned applications:

1. Open the *Security Administration* tool.
2. Switch to the *Code Signing* tab.
3. To allow for the deployment of unsigned artifacts, change the option for *Application Status* to

**Prompt** for *Unsigned applications*.

1. **Save** and close the Security Administration tool.

### To deploy the REST Gateway application package:

1. Download the appropriate version of the REST Gateway component (see [*Product compatibility*](#_bookmark3), above, to select the version that is supported by your AppWorks Platform instance).
2. Deploy the REST Gateway application package:
   1. Open the Application Deployer.
   2. Browse to your REST Gateway application package download location using Browse in the Deployment Status panel.
   3. Click **Upload and Deploy**.
   4. In the Application Deployer – Step 1 of 5, verify that the REST Gateway application is listed and click **Next**.
   5. In the Application Deployer – Step 2 of 5, review the Package Impact report that indicates that the package will be deployed. Click **Next**.
   6. In the Application Deployer – Step 3 of 7 (additional steps due to an unsigned application), place a checkmark in the *REST Gateway* application row to accept the unsigned application. Click **Next**.
   7. In the Application Deployer – Step 4 of 7, review the license agreement. Accept the license agreement and click **Next**.
   8. In the Application Deployer – Step 5 of 7, review the Impact Analysis and click **Next**.
   9. In the Application Deployer – Step 6 of 7, review the summary and click **Deploy**.
   10. When the application deployment is complete, click **Finish**.

## Configuring the REST Gateway

The following sections describe the steps required to configure the REST Gateway and set up a WS- AppServer service container to provide access to the REST services.

**Deploying the REST Gateway TomEE configuration**

1. Download and extract the *REST Gateway TomEE config.tar* archive.
2. Copy the contents of that file to your platform instance under *<defaultinst>webroot\WEB- INF\lib* directory (if the lib directory doesn’t exist, you can create it).

**Modifying the BCP classpath**

1. Open the file *bcp.classpath* under *<defaultinst>\config\bcp.classpath* in your preferred text editor.
2. To the end of the file, append entries for *RESTGateway.jar*, *json-lib.jar*, *commons-lang-2.6.jar*, and *commons-logging-1.2.jar* with their full system paths. Using the default Windows paths, the additional entry string will look like this (note that the paths are not all for the same folder):

C:\Program Files\OpenText\ProcessPlatform\defaultInst\com\cordys\web\rest\RESTGateway.jar;C:\Program Files\OpenText\ProcessPlatform\defaultInst\com\cordys\web\rest\ext\json-lib.jar;C:\Program Files\OpenText\ProcessPlatform\defaultInst\ext\commons-lang-2.6.jar;C:\Program Files\OpenText\ProcessPlatform\defaultInst\ext\commons-logging-1.1.1.jar

1. Save and close the file.

**Creating a WS-AppServer service group**

1. Open the System Resource Manager.
2. Click **New** to open the New Service Group Wizard.



1. (Step 1 of 5) Scroll to the bottom of the page and select WS-AppServer (or use the search feature) and click **Next**.
2. (Step 2 of 5) Provide a name and scroll to the bottom of the Web Service Interfaces table.
3. (Step 2 of 5) select the following Web service interfaces:
   * RESTMappingWebServiceInterface
   * WebServiceInterfaceGenerateServiceMappings
4. (Step 2 of 5) Click **Next**.
5. (Step 3 of 5) Provide a name for the service container.
6. (Step 3 of 5) Switch to the JRE Configuration tab. Add the locations of the same four files from Step 2 of the Modify the BCP Classpath section to the Classpath field (each file separated by ;).
7. Click **Next**.
8. (Step 4 of 5) Clear **Configure Database** and click **Next**.
9. (Step 5 of 5) Click **Finish**.
10. When returned to the System Resource manager, start the WS-AppServer service container.

**Assigning roles**

1. Open the User Manager.
2. In the Users panel, select the user who will generate REST services.
3. In the Roles panel, right-click the REST Developer role and select **Assign to selected User(s)**.
4. Close the User Manager.

# Developing a REST example

Following is a basic example of developing a REST service. For more details, see *REST Provider - Developer Guide*.

This example uses the FTP connector as a sample service to which the REST Gateway is mapped. This test can be performed with other services as well.

## Creating a sample SOAP service for map generation

This example uses the *getListOfFiles* operation of the FTP Connector to provide a SOAP service for map generation.

**Creating the FTP Connector service container**

1. Open the System Resource Manager.
2. Click the **Show** ( ) list and select **All Service Groups**.
3. Click **New** ( ) to open the New Service Group wizard.
4. Scroll to (or search for) and select **FTP**, and then click **Next**.
5. In Provide Service Group Details, provide a name and description (for example, FTP Connector Group) for the service group, select the **Method Set FTP 1.1** web service interface, and then click **Next**.
6. In Provide Service Container details, specify a name and description for the service container, and then click **Next**.
7. Either click **Next**, or, if a proxy server is used, specify its details, and then click **Next**.
8. Click **Finish**.
9. Start the FTP service container.

**Creating an FTP configuration**

1. Open the FTP Configuration Manager.
2. Create a configuration.
3. Enter a name and description and provide the relevant server details (server, port, and credentials).
4. Provide a base directory (this can be set to / to use the user’s home directory.)
5. Test the connection to the FTP server.
6. Click **Save**.

**Testing the FTP Service**

Test the service to ensure it is working prior to creating REST mappings.

1. Open the Service Test Tool.
2. Use the Service Group list and select the service group created above.
3. Use the Web Service Interface list to select **Method Set FTP 1.1**.
4. Use the Web Service Operation list to select **getListOfFilesoperation**.
5. Click **Compose Request**.
6. In the generated request, replace the *PARAMETER* text within the configuration tag with the configuration you created above.
7. Remove the PARAMETER text from the directory tag.
8. Click **Invoke**.

If the response is successful, create the mapping. Otherwise, troubleshoot your service creation.

**Generating REST services**

To generate a REST service mapping, use the Generate REST Services tool.

1. Open the Generate REST Services tool.
2. Use the Organization/ISV Package list to select the package to create a mapping (using the example above, select **Cordys FTP Connector**).
3. Use the Web Service Interface list to select an interface (using the example above, select

### Method Set FTP 1.1).

1. Click **Show** to populate the Operations list.
2. Select an operation (using the example above, select **getListOfFiles**).
3. Modify the Web Service Operation Details accordingly. To use the base directory set within the FTP Configuration, remove the *PARAMETER* text from the directory tag within the Web Service Operation Details.
4. Modify the REST URI Details as follows:
   1. Provide a name for the resource. This will also be the name of the XML file resource in the XMLStore (under **Cordys** > **servicemapping** > **resttosoap**).
   2. Remove the directory and details rows in the URL parameters table of the SOAP to REST mappings.
5. Scroll to the end of the Generate REST Services dialog box and click **Save**.
6. Close the Generate REST Services tool.

**Testing Generated REST Services**

To test a generated REST service, use the Test REST Services tool.

1. Open the Test REST Services tool.
2. Use the Organization/ISV Package list to select the package for which you created a mapping above (using the example above, select **Cordys FTP Connector**).
3. Use the Web Service Interface list to select the interface you created a mapping for above (using the example above, select **Method Set FTP 1.1**).
4. Click **Show** to display services that have been generated matching your selections. If you have more than one service displayed, select the desired service within the Web Service Operation table.
5. In the REST Request section, modify the URI to match your needs (for the example above, replace *{0}* with the FTP Configuration name to use for testing). Also, modify the Request Body as needed for your service (there are no modifications needed for the FTP example).
6. Click **Invoke**.

A successful test will result in a 200 series status code (for example, 226 for ‘Transfer complete’) and the expected response in JSON format.

# Packaging and deploying a REST solution

REST Web service mappings can be packaged in an application package that can then be deployed to a production environment.

## Packaging the REST solution

To package a REST solution, perform the following steps:

1. Under Test REST Services, click **Download**.

Depending on the browser used, a dialog box may display asking whether to open or save the file. The name of the downloaded file will be the name of the resource XML file in the XMLStore.

1. Click **Save**.
2. Use Workspace Documents to create a CWS project, and create the following folder structure under the project:

*Cordys* > *servicemapping* > *resttosoap*

1. In the workspace, right-click the **resttosoap** folder and click **New** > **Other** > **XML Store Definition**.
2. Enter a suitable name and description for the XML Store Definition, click **Save**, and then close it.
3. Right-click the **resttosoap** folder again and click **New** > **Other** > **XML** (create XML documents).
4. Copy the XML contents of the file downloaded in steps 1 and 2 above and paste it into the text area of the XML document.
5. To save the XML, click **Save**. Provide a name and description, and then click **OK**.
6. If needed, update the package properties:
   1. Right-click the project created in step 3 above.
   2. From the context menu that displays, click **Packaging** > **Package Properties**.
   3. Update the properties as needed, and then click **OK**.
7. Generate the package:
   1. Right-click the project created in step 3 above.
   2. From the context menu that displays, click **Packaging** > **Create Package**.
   3. After all the actions for creating the package are complete, click **Download** to download the application package.
   4. Click **Close** to close the packaging status interface.

The resulting application package can be used to deploy the REST mappings to any production environment on which the REST Gateway has been installed and configured.

## Deploying the REST Solution

1. Configure REST on the AppWorks Platform default instance as described in the above section.
2. Install the application package containing the REST mappings
3. On the ***Start Page*** of the Platform instance, click **TEST REST Service**.
4. Click **Activate** to activate the REST mappings.
5. Click **Show** to see the REST mappings.

# Additional information

## Monitoring JMX Settings

In JMX settings, there are two counters through which you can monitor the performance of the REST Gateway. You can view the total time that is spent to process the requests.

* The **Request Processing Counter** displays the time spent in processing the REST request. This includes the time duration for request transformation also.
* The **Response Creation Counter** displays the time spent in creating the REST response (includes response transformation).

In addition to these, Cordys Connector is a managed component, which is used to send requests to the XML store. This can be used to monitor the performance of the reads that are happening on the XML store.

## Limitations

* Providing links to the related resources from a REST resource is not possible.
* Deleting a REST mapping from the *Generate REST Services* or *Test REST Services* user interface is not possible.

### To delete a REST mapping:

1. Open XMLStore Explorer.
2. Go to */Cordys/servicemapping/resttosoap* in the XMLStore.
3. Delete the unwanted resources.
4. Delete the **/Cordys/servicemapping/soaptorest** folder.

However, any mappings left on the system will be left unusable until the *soaptorest* folder is restored to pick up the remaining mappings that were not removed.

**To restore the *soaptorest* folder:**

1. Open Test REST Services.
2. Click **Activate** to recreate the /Cordys/servicemapping/soaptorest and the subfolders.
3. Click **Show** in the Test REST Services user interface to view the mappings.

## Troubleshooting

If the request fails with the HTTP status code **405 Method Not Allowed**, you need to configure the web server to allow the HTTP method you are using. Most web servers, by default, only allow the HTTP methods **get** and **post**.

# REST Gateway Developer Guide

The REST Gateway can be used to expose AppWorks Platform SOAP based web services as RESTful URIs with one of the HTTP method types (GET, PUT, POST, and DELETE).

When the REST Gateway receives a RESTful request from a client, it reads the resource information of the request in XMLStore and converts the RESTful request into an equivalent SOAP request and passes the control to AppWorks Platform. After it is processed, AppWorks Platform sends the SOAP response back to the REST Gateway, which converts the SOAP response back to the REST styled response using the resource information.

## Generating the REST services interface

This interface is used to generate resource XML from SOAP web service operations. This resource is saved in the XMLStore under */Cordys/servicemapping/resttosoap/*. Each resource file contains many restful services.

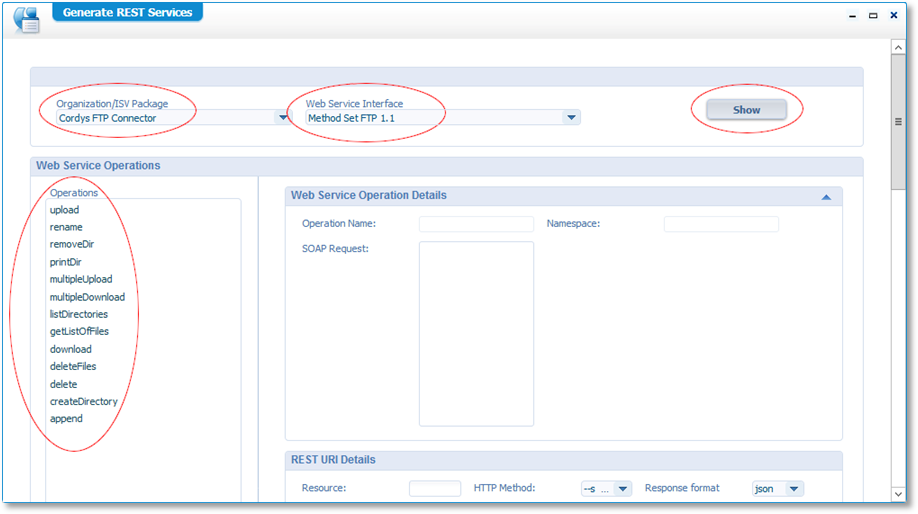
Use of this interface involves three steps:

* ***Selecting the Web service*** for which the REST mappings are to be generated
* ***Selecting the Web service operation*** for which the REST resource is to be generated
* ***Updating the SOAP to REST parameter mappings*** as needed, and then saving the REST resource to the XMLStore

These steps are detailed in the sections below.

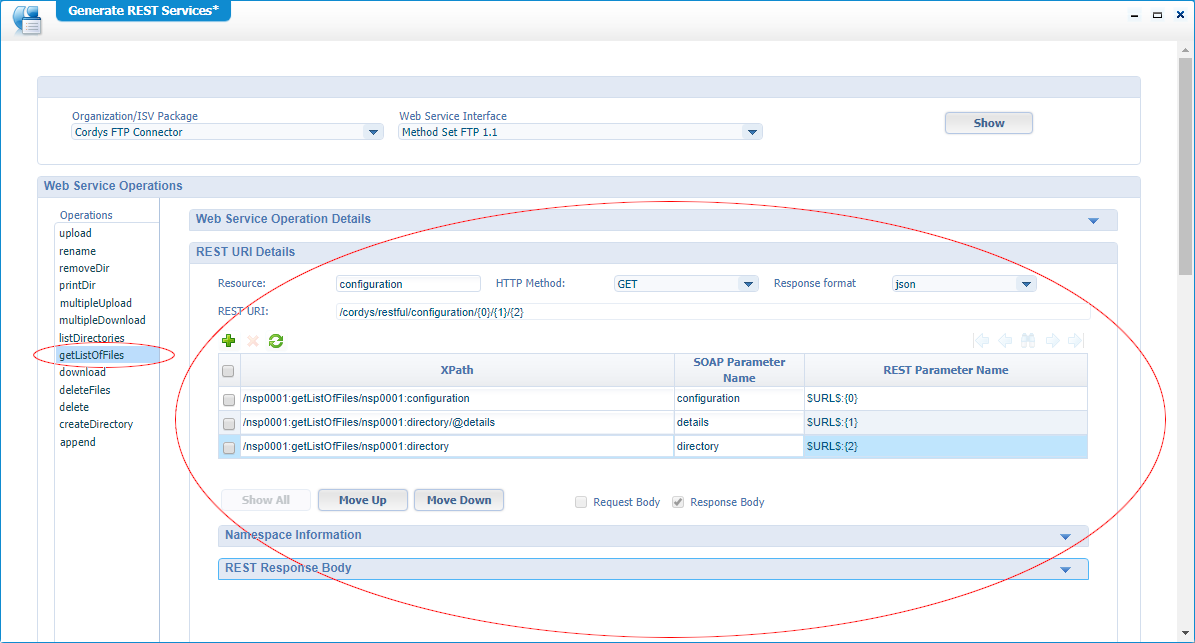
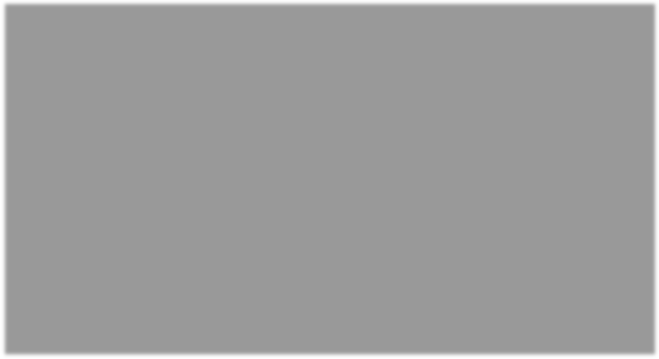
**Selecting the Web Service**

After the Generate REST Services interface is opened, the Organization/ISV Package and one of its associated Web Service Interfaces is selected, and then **Show** is clicked to display a list of operations that are available for the Web service interface. This populates the Web Service Operations list box with the Web service operations available for the Web service:



**Selecting the Web service operation**

When one of the Web service operations is clicked (for example, *getListOfFiles*), the REST mappings are generated with default values that can then be edited as needed:



**Updating the SOAP to REST parameter mappings**

After the Web service and its associated operations are selected, the REST to SOAP parameter mappings can be updated as needed to:

* Name the resource
* Change or select the HTTP method to expose
* Update parameter mappings
* Modify the transformation from the SOAP response XML to REST by editing the XSLT
* Modify the transformation from the REST request XML to SOAP by editing the XSLT

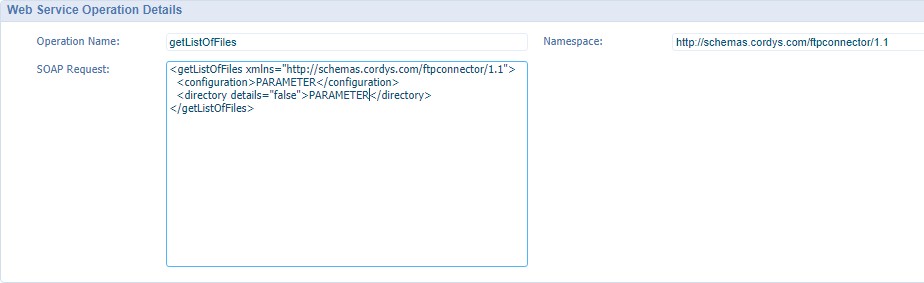
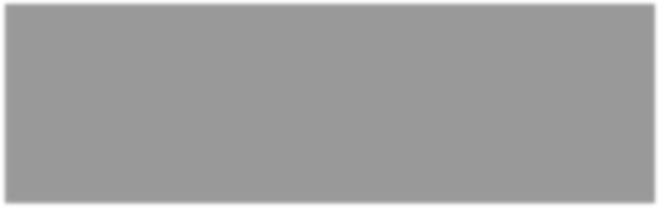
**RESTful operation details**

The REST service details are populated in the following two main sections:

|  |  |
| --- | --- |
| **Web Service Operation Details**: | Displays the SOAP request, which can then be edited, if necessary. |
| **REST URI Details**: | Displays details about the REST URI, which can then be modified as required. |

**Web Service Operation details**

This group box can be expanded by clicking the accordion control and contains the Operation Name, Namespace, and the SOAP Request XML structure of the operation. The SOAP request can be edited to remove unnecessary parameters as required. For example, if *getListOfFiles* is only needed for the FTP user account’s home directory, the directory element can be left empty or removed. Both Operation Name and Namespace come from the Web service operation itself and cannot be edited.



**REST URI details**

This group box is used to map SOAP parameters to REST parameters as follows:

* Specify the name of the resource, the HTTP method, and the response format.
* Specify the SOAP to REST parameters mappings in the URL parameters table.
* Specify namespace information.
* Format the request and the response, if needed.

These values are populated automatically when the operation is selected (for example, *getListOfFiles*) and many can be edited as needed to customize the mapping. The details about the main sections of the REST URI Details group box are detailed in the tables that follow.

**REST URI Details – basic details section**

|  |  |
| --- | --- |
| **Name** | **Description** |
| Resource | Names the resource. A name is provided when the Web Service operation is selected, but it can be edited as needed as long as the name of the resource contains only alphanumeric characters and no embedded whitespace.  **Note**: if the name of an existing resource is used, the REST mapping will be added to the existing resource XML in XMLStore. |
| HTTP Method | Specifies the HTTP Method. Choices are GET, PUT, POST, or DELETE. The interface will default to its best guess (usually a POST) based on the selected Web service operation. |
| Response format | Specifies the format of the response. Choices are JSON (default) or XML. |
| REST URI | Provides the REST URI. This field cannot be edited directly; however, the placeholders for the parameters is updated as entries in the URL parameters table are deleted or added. |

**REST URI Details – URL Parameters table**

|  |  |
| --- | --- |
| **Name** | **Description** |
| XPath | Table column that contains the XPath for the parameter mapping, including namespace prefix. |
| SOAP Parameter Name | Parameter in the SOAP Request to be mapped to a REST parameter placeholder. |
| REST Parameter Name | Parameter string for REST to be mapped to the corresponding SOAP request parameter. |

**Note**: Each row in the URL Parameters table must match a *PARAMETER* in the SOAP request in the Web Service Operation Details.

The table entries are generated automatically by the interface by using the SOAP request of the operation. Additionally, following are the three control options that are available:

* **Show All** – Populates the table with all the parameters available for the Web service. By default, only the first 20 parameters are displayed. For Web services having more than 20 parameters, this option displays all of them.
* **Move Up** – Moves the selected row up by one row.
* **Move Down** – Moves the selected row down by one row.

**REST URI Details – Namespace Information**

|  |  |
| --- | --- |
| **Name** | **Description** |
| Prefix | Prefix for the namespace to use. |
| Namespace | URI of the namespace associated with the prefix. |

The namespace information table contains the prefix and namespace pairs that are used in the XPath. Namespace prefixes generated by the Generate REST Services interface are preceded by the literal substring “nsp000.”

**REST URI Details – REST Request Body (expandable)**

|  |  |
| --- | --- |
| **Name** | **Description** |
| REST Body XML | REST request XML received from the client. |
| XSLT | XSLT stylesheet to transform the REST Body XML into the translated SOAP request XML. The XSLT can be modified to transform the REST request XML as required by the SOAP interface. |
| Translated SOAP Request XML | XML that is created by transforming the REST request XML using the XSLT stylesheet as a template. |

The *REST Request Body* group box is only valid for the HTTP methods PUT and POST and is not displayed when the HTTP methods GET or DELETE are used.

**REST URI Details – REST Response Body (expandable)**

|  |  |
| --- | --- |
| **Name** | **Description** |
| SOAP Response XML | XML structure that is received from the back-end business layer. |
| XSLT | XSLT stylesheet to transform the SOAP response XML into the translated REST response XML. The XSLT can be modified to transform the SOAP response as needed by the REST interface. |
| Translated REST Response XML | XML that is created by transforming the SOAP response XML using the XSLT stylesheet as a template. |

**Using XSLT to transform SOAP to REST**

The XSLT section in the REST Response Body group box can be updated as needed to modify transformations from REST XML to a SOAP request and from SOAP response XML to a REST response.

|  |  |
| --- | --- |
| **IMPORTANT**: | XSLT transformations are an advanced topic that is outside the scope of this document. |

The following example uses XSLT to modify the transformation of SOAP responses to REST. In this example, XSLT is used to transform the SOAP response from the *GetEmployeesObject* operation to REST without the *GetEmployeesObjectResponse*, *old*, and *tuple* elements.

Original REST Response XML:

<GetEmployeesObjectResponse>

<tuple>

<old>

<Employees [xmlns="http://schemas.cordys.com/Northwind">](http://schemas.cordys.com/Northwind)

<EmployeeID>PARAMETER</EmployeeID>{color}

<LastName>PARAMETER</LastName>{color}

<FirstName>PARAMETER</FirstName>{color}

<Title>PARAMETER</Title>{color}

<TitleOfCourtesy>PARAMETER</TitleOfCourtesy>

<BirthDate>PARAMETER</BirthDate>

<HireDate>PARAMETER</HireDate>

<Address>PARAMETER</Address>

<City>PARAMETER</City>

<Region>PARAMETER</Region>

<PostalCode>PARAMETER</PostalCode>

<Country>PARAMETER</Country>

<HomePhone>PARAMETER</HomePhone>

<Extension>PARAMETER</Extension>

<Photo>PARAMETER</Photo>

<Notes>PARAMETER</Notes>

<ReportsTo>PARAMETER</ReportsTo>

<PhotoPath>PARAMETER</PhotoPath>

</Employees>

</old>

</tuple>

<GetEmployeesObjectResponse>

To remove the elements *GetEmployeesObjectResponse*, *old*, and *tuple* from the REST response, use the following XSLT:

<xsl:stylesheet xmlns:xsl="<http://www.w3.org/1999/XSL/Transform>" xmlns:sch="<http://schemas.cordys.com/Northwind>" version="1.0" >

<xsl:output method="xml" />

<xsl:template match="sch:GetEmployeesObjectResponse">

<xsl:copy-of select="sch:tuple/sch:old/sch:Employees" />

</xsl:template>

</xsl:stylesheet>

Target REST response XML, with the response and tuple elements removed:

<Employees [xmlns="http://schemas.cordys.com/Northwind">](http://schemas.cordys.com/Northwind)

<EmployeeID>PARAMETER</EmployeeID>{color}

<LastName>PARAMETER</LastName>{color}

<FirstName>PARAMETER</FirstName>{color}

<Title>PARAMETER</Title>{color}

<TitleOfCourtesy>PARAMETER</TitleOfCourtesy>

<BirthDate>PARAMETER</BirthDate>

<HireDate>PARAMETER</HireDate>

<Address>PARAMETER</Address>

<City>PARAMETER</City>

<Region>PARAMETER</Region>

<PostalCode>PARAMETER</PostalCode>

<Country>PARAMETER</Country>

<HomePhone>PARAMETER</HomePhone>

<Extension>PARAMETER</Extension>

<Photo>PARAMETER</Photo>

<Notes>PARAMETER</Notes>

<ReportsTo>PARAMETER</ReportsTo>

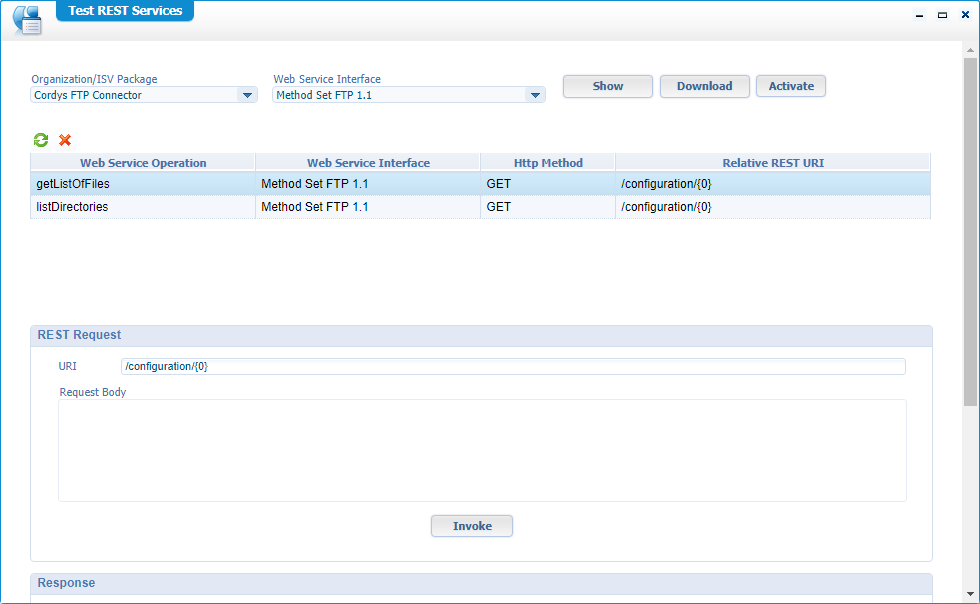
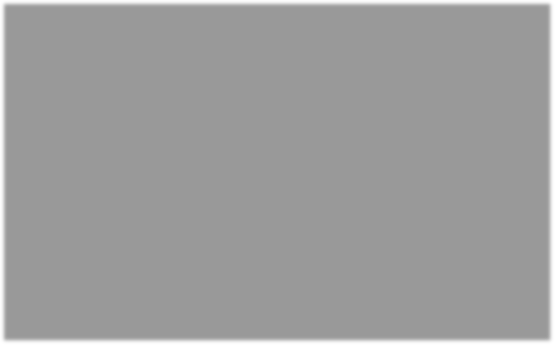
<PhotoPath>PARAMETER</PhotoPath>

</Employees>

## Test REST Services interface

The Test REST Services interface is designed to test the generated REST services by offering GET, DELETE, POST, and PUT. Users can select the generated REST services by selecting the respective Organization/ISVP package and Web Service Interface.

After selecting the Organization/ISVP package and the Web Service Interface, the ***Show*** button can be clicked to list all the REST Services available for the selected Web service interface:



The Test REST Services interface can also be used to download existing resources for packaging and deployment in a production environment, as well as activate a recently deployed REST services package.

### To download a resource for packaging and deployment:

1. Select the desired Organization/ISV Package.
2. Select a Web Service Interface that is available for the organization/package.
3. Click **Show**.
4. Click **Download**.

The resource is downloaded to the default download location as an XML file named:

<*resource\_name*>.xml

Where: <*resource\_name*> is the name of the resource generated by the *Generate REST Services* interface. This XML can then be used and defined as an XMLStore object in a CWS project, which can be packaged and then deployed on another environment.

To activate a REST package that was recently deployed:

1. Select the desired Organization/ISV Package.
2. Select a Web Service Interface that is available for the organization/package.
3. Click **Activate**.

For details about packaging and deploying REST services, see [*Packaging and deploying a REST solution*](#_bookmark17).

# Known Issues and troubleshooting

## JSON generation Issue with REST Gateway 2.0.0 (Java 7-8 version)

This affects the REST gateway in any version of Cordys BOP/OpenText AppWorks Platform from Cordys BOP 4.2 to AppWorks 16.5.x. When first deployed, the REST Gateway is unable to deliver the response in the default JSON format.

**Workaround**

1. Copy the file *<platform\_instance>/com/cordys/web/rest/RESTGateway.jar* to a directory outside of the platform instance directories.
2. Use 7zip or another archive tool to open the JAR file.
3. Extract the manifest.xml file for editing.
4. Insert the following line under the first line in the manifest:

Class-Path: ext/json-lib.jar

1. Add the updated manifest back into the JAR file.
2. Copy the JAR file to its original location in the AppWorks Platform instance.
3. Restart the AppWorks Platform instance to pick up the classpath.

## Classpath issues with REST Gateway 3.0.1 (Java 11 version)

This affects the REST gateway in AppWorks Platform 16.6+ with Java 11. The *RESTGateway.jar* file needs to have its classpath updated. This happens because one of the REST Gateway required dependencies, the Apache *commons-lang-2.3.jar* was dropped in AppWorks Platform 16.6. As a result, the Java 11 version of the REST Gateway is packaged with Apache *commons-lang-2.6.jar*, which is compatible with the REST Gateway. Additionally, because of the JSON issue above also affects REST Gateway 3.0.1, *json-lib.jar* needs to be added to the classpath too.

**Workaround**

1. Copy the file *<platform\_instance>/com/cordys/web/rest/RESTGateway.jar* to a directory outside of the AppWorks Platform instance directories.
2. Use 7zip or another archive tool to open the JAR file.
3. Extract the manifest.xml file for editing.
4. Insert the following line under the first line in the manifest:

Class-Path: ext/json-lib.jar ext/commons-lang-2.6.jar

1. Add the updated manifest back into the JAR file.
2. Copy the JAR file to its original location in the AppWorks Platform instance.
3. Restart the AppWorks Platform instance to pick up the classpath.

## Test REST services interface’s download button does not work

This affects all versions of the REST Gateway. The Download button does not work if the browser’s

pop-up blocker is enabled.

To fix the issue: disable the Web browser’s pop-up blocker or provide permissions to always allow pop- ups for the URL of the AppWorks Platform instance.

## Request fails with http status code 405 Method Not Allowed

When this happens with the HTTP methods PUT or DELETE, the Web server must be configured to allow the HTTP method used. Most web servers will only allow the HTTP methods GET and POST**.**

# Contact Information

OpenText Corporation 275 Frank Tompa Drive Waterloo, Ontario Canada, N2L 0A1

OpenText My Support: [https://support.opentext.com](https://support.opentext.com/)

For more information, visit [www.opentext.com](http://www.opentext.com/)

**Copyright © 2019 Open Text. All Rights Reserved.**

Trademarks owned by Open Text. One or more patents may cover this product. For more information, please visit https://[www.opentext.com/patents.](http://www.opentext.com/patents)

**Disclaimer**

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, Open Text Corporation and its affiliates accept no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication.