Project and Portfolio Management Center

Software Version: 9.40

Excel Reports Cookbook

Document Release Date: September 2016
Software Release Date: September 2016

Hewlett Packard Enterprise

Legal Notices

Warranty

The only warranties for Hewlett Packard Enterprise products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from Hewlett Packard Enterprise required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Adobe® is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

Documentation Updates

To check for recent updates or to verify that you are using the most recent edition of a document, go to: https://softwaresupport.hpe.com/.

This site requires that you register for an HP Passport and to sign in. To register for an HP Passport ID, click **Register** on the HPE Software Support site or click **Create an Account** on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HPE sales representative for details.

The following table indicates changes made to this document since the last released edition.

Support

Visit the HPE Software Support site at: https://softwaresupport.hpe.com/.

Most of the support areas require that you register as an HP Passport user and to sign in. Many also require a support contract. To register for an HP Passport ID, click **Register** on the HPE Support site or click **Create an Account** on the HP Passport login page.

To find more information about access levels, go to: https://softwaresupport.hpe.com/web/softwaresupport/access-levels.

HPE Software Solutions Now accesses the HPSW Solution and Integration Portal website. This site enables you to explore HPE Product Solutions to meet your business needs, includes a full list of Integrations between HPE Products, as well as a listing of ITIL Processes. The URL for this website is https://softwaresupport.hpe.com/km/KM01702731.

Contents

About This Guide

The goal of this document is to present the new Excel reports introduced in 9.30 and the improvements introduced in 9.40, and to guide PPM Administrators through the steps of creating a new Excel report.

> **Note:** You can find a list of sample files used with this guide at HPE Live Network.

What are Excel Reports

Excel reports are a new way to create HPE PPM reports introduced in PPM Center 9.30. They are completely integrated in PPM Center, which means that:

- They run on existing PPM Server and do not require any extra hardware.
- They do not require any extra software installation besides PPM.
- They do not require any extra license fees.

Moreover, Excel reports are built on top of the existing PPM in-app reporting solution, like the JSP Reports or the PL/SQL reports. As a result, like any other PPM report, they are run or scheduled from the Create Report menu of PPM Center, and are defined in a Report Type configured from PPM Workbench.

The major difference with a standard PPM report is that the output of an Excel report is always an Excel file in XLSX format, whereas the output of a JSP report is usually an HTML file.

To summarize, Excel reports are considered to be an improvement of PPM JSP Reports, which means, Excel reports are easier to use, and present the ready-to-use data in Excel files.

Third party libraries for Excel Reports

HPE PPM Center is using the JETT library to fill the Excel reports templates (http://jett.sourceforge.net/). All features supported by JETT are available in Excel reports. Even though you can find the most important features of this library in this document, HPE recommends you to go through the official documentation of JETT in order to learn about all the features available. As of PPM 9.40, JETT version 0.80 is used.

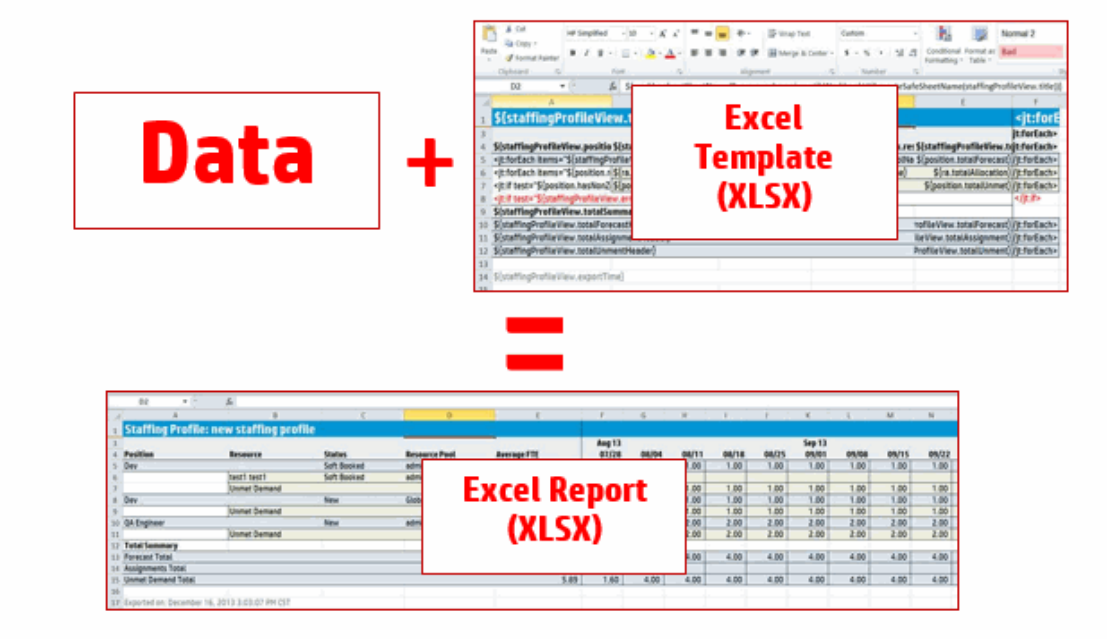
The Apache POI library is also used to insert records in the reports in Large Data mode (http://poi.apache.org/). As of PPM 9.40, Apache POI version 3.10 is used.

What are not Excel Reports

Excel reports are not intended to be a full-fledged Enterprise-class reporting solution. You can consider Excel reports as a potential alternative to PPM JSP Reporting, but Excel reports can not replace reporting tools such as Business Objects, Cognos, or Tableau.

Excel Reports Overview

To get an Excel report, you only need to mix some data with an Excel template.



The data can come from different sources:

- Fields filled in by the user when submitting the report with the **Create Report** menu in the PPM user interface
- Meta-data of the user that created the report, such as user name, full name, security groups, that is automatically passed to the report
- Results generated by custom Java code (Java Preprocessor)
- Data retrieved from a SQL query written directly in the template
- Data retrieved from a PPM Dashboard Datasource directly in the template

Prerequisites for Running an Excel Report

A Report Type Defined in PPM Workbench

The report type is similar to a JSP Report, with the only difference that the special command `ksc_run_jsp_report <path_to_jsp_file>` is replaced by `ksc_run_excel_report <path_to_excel_template>`.

The report type will define the PPM Validations to be used as input data to the Excel reports, as well as the security model of who can run the report from PPM.

An Excel Template (XLSX File)

The Excel template is an XLSX file that looks similar to the final report, where all the data is replaced by placeholders that will be filled with data when the report is generated.

The template can also contain scripts to execute arbitrary code. Scripting in the template offers the same features as code run from a JSP file. However, it is recommended to limit the use because code located in the template is not easy to maintain and is tedious to troubleshoot in case of syntax problem or bug.

All the formatting defined in the Excel template (such as font, style, and cell format) is usually preserved when the report is generated. The other Excel objects (such as tables, charts, and pivot tables) are left untouched during the report generation.

From the template, it is possible to retrieve data from PPM Dashboard Datasources, and to run SQL queries against any database (PPM database, PPM Reporting database, arbitrary external database reachable from PPM Server through JDBC).

(Optional) A Java Pre-Processor

When there are some advanced requirements to process data used in the Report Template, it is tedious to use the scripting features in the template. For this reason, it is possible to define a Java Pre-Processor class that can be run before the report and can feed data to the report. All the data available

to the template is also available to the Java Processor, so that you can modify existing data or add new data to the report.

Even though everything done in a Java Pre-Processor can also be done by in-template scripting, it is recommended to rely on a Java Pre-Processor for every non-trivial operation, such as complex SQL Query generation or invocation of an external web service.

(Optional, since 9.40) A Java Report Postprocessor

If you need to use the low-level Apache POI library to manipulate the report XSLX after it was filled by JETT (to add some charts or pictures for example), you can write a Java class that will get the Apache POI Workbook object as an input and let you manipulate it any way you want using the POI XSSF Interface.

Even though such code can be written directly in the Excel Template (as JETT exposes the POI Workbook and Sheet objects and lets you manipulate them using scripting), it is recommended to rely on a Java Report Postprocessor for every non-trivial POI XLSX manipulation.

(Optional, since 9.40) A Java Report Streaming Class

When you have a huge amount of data to include in your report (tens of thousands of rows or hundreds of thousands of cells), you have to stream the data to the XLSX in order to get good performance and low memory usage. The first option is to use the Large Data Mode (documented in "Large Data Mode" on page 29), however, this approach is very limited in usage: data can only be inserted at the end of the first sheet of the workbook, and you cannot change the style of the cells.

If you want full control of the XLSX streaming process, you can use a Java Report Steaming class as an alternative, which will give you access to the workbook through the POI SXSSF API. This API lets you stream data to any spreadsheet of the workbook, and gives you full POI control of the data inserted (including cell style).

It is NOT possible to use both “Large Data Mode” and a Java Report Streaming class or a Java Report Post-Processor. Trying to do so will result in an error.

Keep in mind that the Apache POI SXSSF API is more limited in usage than the XSSF API: you can only add data to existing spreadsheets, and can only manipulate one row at a time. It is possible to first

modify the workbook with a Java Report Postprocessor and then insert large amounts of data with a Java Report Streaming class, but it will always happen in that order.

Order of the Excel Report Generation Process

Generating an Excel Report will always go through different phases in the following order:

1. (Optional) Java Preprocessor gets the data to make it available to fill the template.
2. JETT fills the XLSX template with the data to create the XLSX report.
3. (Optional) If a Large Data Mode object name is defined, Large Data Mode is used.
4. (Optional) Java Report Postprocessor manipulates the report with POI XSSF API.
5. (Optional) Java Report Streaming class manipulates the report with POI SXSSF API

Where to Store the Excel Template Files

By default, the Excel template files are stored at `<PPM_HOME>/conf/custom_excel_templates`. However, in a clustered environment, a template file must be copied on every PPM_HOME directory of the cluster. This can quickly become tedious to maintain. For that reason, whenever running PPM in clustered configuration, it is advised to define the server parameter `EXCEL_TEMPLATES_PATH` and set it to a shared folder where Excel templates will be stored. This allows to store templates in one single folder, whatever be the size of your cluster deployment. Moreover, it allows to give Reports creators the possibility to access and modify the Excel templates without giving them access to the PPM Server file system, which is a good practice.

It is advised to create a folder under the `custom_excel_templates` folder (such as `Reports`), and organize your templates in sub-folders by module or business department.

Your First Excel Report

This section provides steps on creating a simple Excel report.

This report lists all PPM Resources, with their full name, manager full name, department, start date, end date, and category.

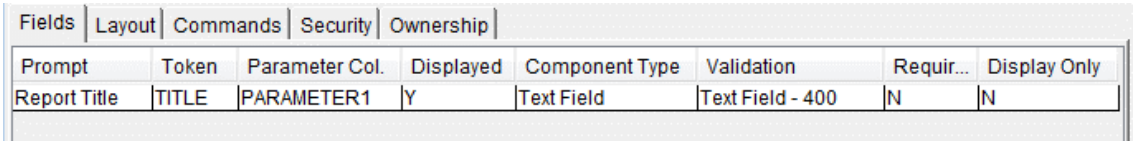
Creating the Report Type in PPM Workbench

You can directly import the report type from `Samples/1 – Your First Report/excelReport_ReportType.zip` or follow the step-by-step instructions.

1. In order to import the report type, copy the zip file in `<PPM_HOME>` then go to `<PPM_HOME>/bin` and run the following command. You need to make sure to replace the username and password values:

   ```
   sh ./kMigratorImport.sh -username <username> -password <password> -action import -filename excelReport_ReportType.zip -i18n none -refdata nochange -flags NNNNNNNNNNNNNNNNNN
   ```

2. To create the report type step-by-step, open the PPM Workbench, go to **Configuration / Report Types** and click the **New Report Type** button.
3. Enter the following values for the fields. You can make changes as needed:
   - **Report Type Name**: Resources Details (Excel)
   - **Reference Code**: RESOURCES_DETAILS_EXCEL
   - **Description**: "Resources Details" Excel report
   - **Enabled**: Yes
   - **Report Category**: Resource Management
4. In the **Fields** tab, create one field of type **Text Field – 400** with Token of **TITLE** and Prompt of **Report Title**. Keep the **Layout**, **Security**, and **Ownership** tabs untouched.



5. Open the **Commands** tab, and click the **New Cmd** button. Enter the following information (feel

free to change everything but the steps):

Command: Run "Resources Details" Excel report

Timeout (s): 3000

Enabled: Yes

Steps:

```
ksc_run_excel_report reports/rm/ResourcesDetailsTemplate.xlsx
REPORT_ID=[RP.REPORT_SUBMISSION_ID]
ksc_end_report_parameters
```

These command steps are very important to properly run the Excel report. They indicate what excel template should be used to render the report.

The second line is also mandatory and is used to pass the report ID to the Excel report engine. It should never be omitted nor modified.

It is possible to add any number of arbitrary parameters between the first and the last line. All these parameters values will be passed to the excel template when the report is generated. For example, to pass a parameter **TIME_PERIOD** with value **weeks**, you can use the following command steps:

```
ksc_run_excel_report reports/rm/ResourcesDetailsTemplate.xlsx
REPORT_ID=[RP.REPORT_SUBMISSION_ID]
TIME_PERIOD=weeks
ksc_end_report_parameters
```

Creating the Excel Template

Most of the time, you do not need to start the Excel template from scratch. You already have an XLSX file that contains the data the way users want it to look like. In this example, it is supposed that some users want a final report that looks like the file `Samples/1 – Your First Report/1 - Target report.xlsx`.

The core feature of Excel reports is how PPM can replace placeholders in the report template at run time and transform the template into the final report. Follow these steps to add placeholders on that template:

- One ${TITLE} placeholder right after the report title. This will be replaced by whatever value with input in the **Report Title** field when submitting the report.
- One ${RPT_TIMESTAMP} placeholder in place of the report creation time. This is a built-in token that is passed by PPM to the template and contains the report creation time.
- One ${TITLE} placeholder in the **Spreadsheet Name** tab, in order to have the Excel Spreadsheet named with whatever value with input in the **Report Title** field when submitting the report in PPM.

The result of these simple changes can be viewed in the file `Samples/1 – Your First Report/2 - Template with simple placeholders.xlsx`.

If you want to learn how to list all the names that can be used as placeholders in your Excel template, you can jump directly to "Debug Mode: How to View What Data is Passed to Your Excel Template" on page 24.

Copying the Excel Template to PPM Server

Even though you have not made the changes to retrieve the list of resources from PPM, you can already use this Excel template to create thr first PPM Excel report. All you have to do is to copy the template to the PPM Excel templates folder and run the report from PPM **Create report** menu.

The folder structure where the report is stored and the template XLSX file name must match whatever path was used in the command steps of the Report Type. In this example, the following path is used: `reports/rm/ResourcesDetailsTemplate.xlsx`.

Rename the template file to `ResourcesDetailsTemplate.xlsx` and store in the templates folder in subfolders `reports` and `rm`.

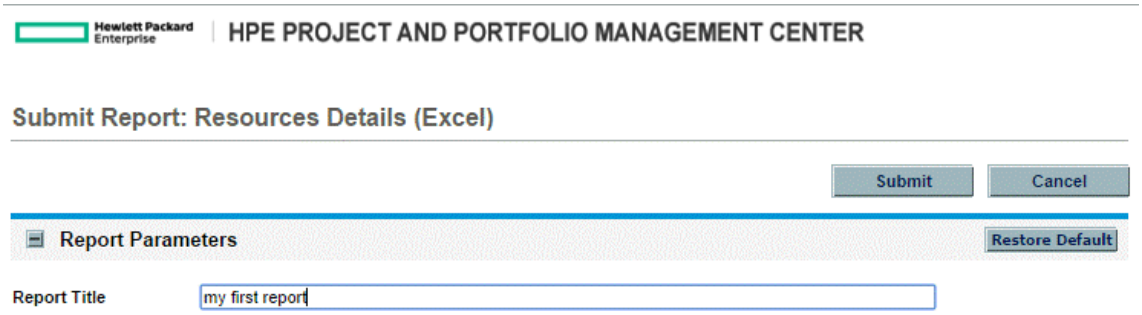
You can find information on where to store the templates in "Where to Store the Excel Template Files" on page 12.

There is no need to restart the PPM Server after updating an Excel template for changes to take effect. Templates are read from disk whenever a report is generated. As a result, the templates files might be locked if a report is being created at the time of replacement. Wait until the report either completes or fails to replace the template file.

Running the Report

In PPM Center, select menu **Create** > **Report**, then select **Resource Management** (or whatever Report Category you set for the report type). Click the report type name (**Resources Details** (Excel)),

input whatever report title you want in the field (“my first report”, for example), and click **Submit**.



When the report is completed, you can download an XLSX file containing the report.

You can find the expected result file in `Samples/1 – Your First Report/3 - Created Report with simple placeholders.xlsx`.

Getting Data from PPM using SQL Query

The first step to get the data into our Excel report is to come up with a SQL query that will return the data that is needed. In this example, the following SQL query (also available in `Samples/1 – Your First Report/4 – SQL to retrieve resources details.sql`) returns the expected results:

```
SELECT
  USERNAME,
  FULL_NAME,
  START_DATE,
  END_DATE,
  DEPARTMENT_MEANING DEPARTMENT,
  MANAGER_FULL_NAME MANAGER,
  RESOURCE_CATEGORY_MEANING CATEGORY,
  roles.ROLE_NAME ROLE
FROM KNTA_USERS_V u,
  RSC_RESOURCES r,
  RSC_ROLES roles
WHERE RESOURCE_FLAG = 'Y'
AND u.ENABLED_FLAG = 'Y'
AND r.user_id       = u.user_id
AND r.PRIMARY_ROLE_ID  = roles.ROLE_ID (+);
```

Use it in the template to retrieve the data, using the following placeholder:

```
${ppmdb.execQuery('SELECT u.USER_ID, USERNAME, FULL_NAME, START_DATE, END_DATE,
DEPARTMENT_MEANING DEPARTMENT, MANAGER_FULL_NAME MANAGER,  RESOURCE_CATEGORY_
MEANING CATEGORY,  roles.ROLE_NAME ROLE FROM KNTA_USERS_V u,  RSC_RESOURCES r,
RSC_ROLES roles WHERE RESOURCE_FLAG = \\'Y\\' AND u.ENABLED_FLAG = \\'Y\\' AND
r.user_id = u.user_id  AND r.PRIMARY_ROLE_ID  = roles.ROLE_ID (+)')}
```

Consider the following when executing SQL queries in the template:

- Any SQL query inserted in the template must be on a single line
- The SQL query is delimited by single quotes (‘). Every single quote used in the query must be escaped by prefixing two backslashes (\\)
- You can pass parameters to a SQL query by inserting a question mark (?) in the query where the parameters stand, and pass as parameters values as extra parameters of the execQuery() method.

  For example, if you have a report field using a validation returning the **resource categories** and pass it in a token **P_RESOURCE_CATEGORY**, you could filter results in the SQL using the following code:

  ```
  ${ppmdb.execQuery('SELECT * FROM knta_users_v WHERE RESOURCE_CATEGORY_CODE =
  ?', P_RESOURCE_CATEGORY)}
  ```

  More information can be found on the web site of JETT, the third-party library used by Excel reports to fill in templates: http://jett.sourceforge.net/misc/jdbc_executor.html

  In order to iterate over each result returned by this SQL, use the <jt:forEach> template tag at the start and end of the line where resources info will go. (http://jett.sourceforge.net/tags/forEach.html). This tag stores the result of the SQL query in a variable resource, and then use placeholders in each cell to display values of the different columns returned by the SQL query. For example, because the resource full name is returned in the column FULL_NAME, use the placeholder `${resource.FULL_NAME}` in the cell where you want to insert the resource full name.

  The corresponding template is available in `Samples/1 – Your First Report/5 – Template with SQL query.xlsx`.

  After replacing the template on PPM Server with this template and re-running the report, the produced Excel report can be found in `Samples/1 – Your First Report/6 - Created Report with SQL query.xlsx`.

Getting Data in Template from Other Sources

This section provides information about how to get data in template from other sources.

Running SQL Queries on PPM Operational Reporting Database

Anyone with advanced reporting requirements should consider using the PPM Operational Reporting DB Schema, especially since the release of Content Pack 2.0. It is not only faster and lighten the load on PPM production database, but also results in much simpler SQL because the data is already aggregated in facts and dimensions tables.

For example, the SQL query listing all resources used earlier in that document hits two tables and one very complex view (KNTA_USERS_V, leveraging 12 database tables). However, running the same SQL query on the PPM Reporting database only needs to get data from one single table (RPT_DIM_RM_RESOURCES) that includes all resources information.

In order to run a SQL query on the reporting database, replace `ppmdb` by `reportingdb` in the placeholder where SQL query is executed. For example: `${reportingdb.execQuery('SELECT * FROM RPT_DIM_RM_RESOURCES')}`. This requires that PPM Reporting Content Pack 2.0 or later be installed and properly configured.

PPM Dashboard Datasource

If you have some Dashboard Datasources already defined in the workbench with the relevant data and filters, or if you prefer to avoid writing hard-to-maintain SQL in the Excel template, it is possible to retrieve data directly from Dashboard Datasources in the Excel template. You can retrieve the Datasource in the template by using the Datasource name.

It is possible because you can easily create a Dashboard Datasource (and a corresponding List Portlet) in a SQL statement from the **Administration Console** > **Administration Tasks** > **SQL Runner** menu. Input your SQL statement, and click **Create Dashboard Datasource**. You may still need to do some column adjustments in the workbench and create all the filters. This saves you the hassle to create all the columns with the relevant types.

In order to get data from a Dashboard Datasource, use the following placeholder: `${datasources.get('My Datasource Name').getData()}`.

If you use more than one language on your PPM instance, this method only works with the datasource name defined in the language of the user running the Excel Report. Therefore, if you plan to have users running this report in multiple languages, you should rather use the method “getFromId()” that uses the datasource ID instead of the datasource name as a parameter:

```
${datasources.getFromId(34567).getData()}
```

You can get the datasource ID from database table `KDSH_DATA_SOURCES` by running the following SQL query on PPM database:

```
SELECT DATA_SOURCE_ID FROM KDSH_DATA_SOURCES WHERE DATA_SOURCE_NAME = '<Insert the name of your datasource here>'
```

You can also add filters to the datasource by calling the method `addFilter`(‘FILTER_NAME’, filter_value) as many time as needed before calling the method `getData()`. For example, you have a report field using PPM User validation stored in the P_USER_ID token. If you want to use the filter `CREATED_BY` on the PPM Dashboard Datasource listing programs, you can use the following placeholder: `${datasources.get('Program List').addFilter('CREATED_BY', P_USER_ID).getData()}`.

Running SQL on External Databases

It is possible to run SQL queries on external databases, but there are some existing limitations and security concerns on the use of this feature in production environments or on business sensitive databases.

Two ways are available to run SQL queries on an arbitrary DB:

- If itis acceptable to store the DB user name and password in the Excel template, you can use `${externaldb.getJdbcConnection (jdbcUrl, username, password).execQuery(…))}`.

  The user name and password are optional parameters. The drawback of this approach is that everyone with access to the Excel template will know of the user name and password, which is an obvious issue (they won’t be available in the report though). Moreover, if the database information changes (JDBC URL, user name, password), you need to edit your templates.

  This solution makes you target non-Oracle databases. However if that is the case you must make sure that the JDBC driver used to connect to the external database is on PPM classpath (for example, in `<SERVER_HOME>/deploy/itg.war/WEB-INF/lib`).
- You can also create a JNDI datasource in PPM Server.

Make a copy of itg-ds.xml located in `<SERVER_HOME>/deploy` and edit the contents to match your database. The new file name must end with `-ds.xml` and it must be located in the `<SERVER_HOME>/deploy/ directory`. Make sure you remove the `<security-domain>` element. You must also add elements `<user-name>` and `<password>` in this JNDI datasource definition. When this is done, run `kUpdateHtml.sh` and restart PPM. There are currently two limitations with this approach:

- PPM does not support yet encryption of user name and password in custom JNDI datasource.
- As of PPM Center 9.40, you can only create JNDI Datasources that target Oracle Databases.

  When the JNDI datasource is added, you can query it from the Excel template using `${externaldb.get(JNDI_name).execQuery(…)}`.

> **Note:** Connecting to external database should not be used in production unless the security implications have been fully understood. In the meantime, you can use the workaround of creating DB Links to the external databases in the PPM database or Reporting database and connect to either of these databases.

Adding a Pivot Table and a Chart

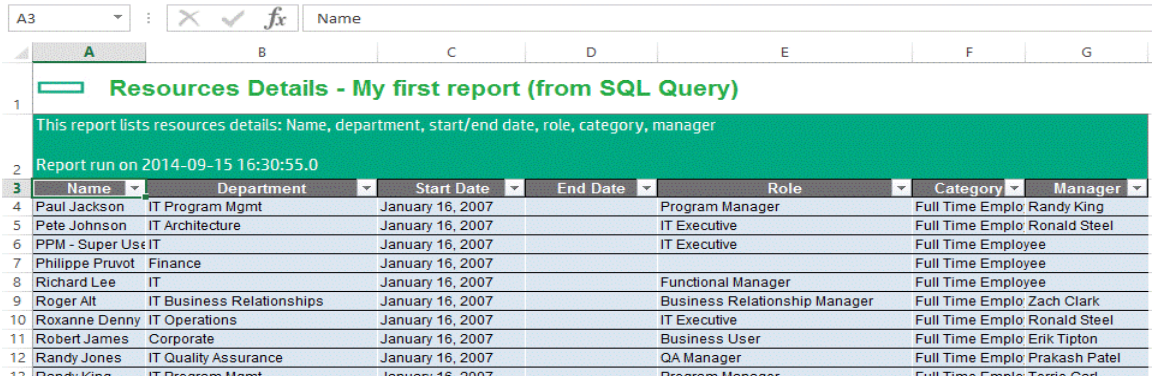
It is useful to export rows and rows of data on an Excel spreadsheet. However, the drawback is that adding charts turns data into information. With Pivot table, you can easily slice and dice the data for deeper analysis.

Moreover, Pivot table enables you to create simpler reports by exporting all the (raw) data in the report with a minimum number of filters on the **Create report** page. Then you can slice, dice, filter, and customize the data directly in Excel and view the result in real time without running the report again with a different set of filters.

Defining a Dynamic Named Range

Because the data is inserted in the template when the report is created, it is impossible to create charts in the template with static range as chart source data. In order to create charts, Excel tables, and pivot tables on data inserted at report creation time, you need to create a Dynamic Named Range.

You can reuse the report created in the "Your First Excel Report" on page 13. You can see that the first row of the data starts in cell A3 if you include the headers.



A3 | Name

	A	B	C	D	E	F	G
1	Resources Details - My first report (from SQL Query)						
2	This report lists resources details: Name, department, start/end date, role, category, manager Report run on 2014-09-15 16:30:55.0						
3	Name	Department	Start Date	End Date	Role	Category	Manager
4	Paul Jackson	IT Program Mgmt	January 16, 2007		Program Manager	Full Time Emplo	Randy King
5	Pete Johnson	IT Architecture	January 16, 2007		IT Executive	Full Time Emplo	Ronald Steel
6	PPM - Super Use	IT	January 16, 2007		IT Executive	Full Time Employee	
7	Philippe Pruvot	Finance	January 16, 2007			Full Time Employee	
8	Richard Lee	IT	January 16, 2007		Functional Manager	Full Time Employee	
9	Roger Alt	IT Business Relationships	January 16, 2007		Business Relationship Manager	Full Time Emplo	Zach Clark
10	Roxanne Denny	IT Operations	January 16, 2007		IT Executive	Full Time Emplo	Ronald Steel
11	Robert James	Corporate	January 16, 2007		Business User	Full Time Emplo	Erik Tipton
12	Randy Jones	IT Quality Assurance	January 16, 2007		QA Manager	Full Time Emplo	Prakash Patel

You can then create a Dynamic Named Range that includes all the data from this list, including the headers, even when rows are added or removed from the list.

The trick is to use the Excel functions OFFSET (to select a dynamic range) and COUNTIF (to count the number of rows with data). If you start at cell A3 and have a table with seven columns, the corresponding formula looks like the following:

```
=OFFSET('${TITLE}'!$A$3,0,0, COUNTA('${TITLE}'!$A$3:$A$999999), 7)
```

This formula basically states:

Select a range starting at Cell A3, with a width of 7 columns and a height equal to the number of cells with a value between A3 and A999999.

This means:

- You should not have any cells with data in the column A after this list.
- You should not have any cell without data in the list for column A. Here, column A is the resource full name, so it is not supposed to be empty; but you cannot pick the Role column for computing the dynamic range because some of the Role values are empty.

Note that the spreadsheet name with the placeholder in the formula (${TITLE}) is included; it is automatically updated when the spreadsheet is renamed during report generation.

If the number of columns is also dynamic, you can replace the columns number by another COUNTIF formula:

```
=OFFSET('${TITLE}'!$A$3,0,0, COUNTA('${TITLE}'!$A$3:$A$999999), COUNTA('${TITLE}
'!$A$3:$ZZ$3))
```

You can then create a Named Range **DataRange** with the previous formula. You can create it in menu **Formulas/Name Manager**, then click **New** and create it with a scope of **Workbook** so that it can be accessed from any spreadsheet. Input **DataRange** for the Name, and the previous formula for the **Refers to:** field.

If you click the Excel **Name Box** from the input field in the top left corner under the ribbon where the active cell number is displayed, and input **DataRange** , it should select the whole list of data in your spreadsheet.

Adding a Pivot Chart and a Pivot Table

If you want to improve your existing **Resources Details List** report to include a pie chart to show resources by category, and a pivot table to count resources by role and category, with filters for manager and department, refer to the example of how this report should look like in `Samples\2 - Adding a Pivot Table and a Chart\1 - Target report with Charts.xlsx`.

Whenever adding charts and pivot tables to an existing Excel report that lists the data, it is advised to follow these steps:

1. Start from the report result (in this case, `Samples/1 – Your First Report/6 - Created Report with SQL query.xlsx`).
2. Create one or more Dynamic Named Range containing the data needed in the charts and pivot tables (see "Defining a Dynamic Named Range" on the previous page for detailed instructions).

3. Create all the charts and pivot tables using the dynamic named ranges. Ensure that the option **Refresh data when opening the file** is selected in the pivot table options on the **Data** tab.
4. Replace the business data with the template tags and placeholders. You can copy them from an existing excel template (in this case, from `Samples/1 – Your First Report/5 - Template with SQL query.xlsx`).
5. Remove every template placeholder or template tag from the pivot tables cells; otherwise they will be evaluated when the report is created and will cause errors. Replace all of these with a single space; the correct data will be inserted when the report is created if you selected the option to refresh data when opening the file in Step 3.

After this is done, you should obtain a document similar to `Samples\2 - Adding a Pivot Table and a Chart\2 - Template with Pivot Chart and Pivot Table.xlsx`.

Copy this template on your PPM Server, run the report. The result should look like `Samples\2 - Adding a Pivot Table and a Chart\3 – Created Report with Pivot Chart and Pivot Table.xlsx`.

> **Note:** If you download the PPM Report in Internet Explorer and use the “Open” menu instead of “Save”, the report file will be opened in read-only mode directly from the PPM Server. It won’t be saved locally, and this will cause pivot datasources not to load correctly when opening the file in Excel 2013 and later. If your report has dynamic pivot tables, please use the “Save” or “Save as” menu when opening PPM Excel Reports in Internet Explorer.

Debug Mode: How to View What Data is Passed to Your Excel Template

When running an Excel report, the Debug mode replaces the report output by a list of all the data available in the Excel template. This includes:

- All fields submitted when creating the report (with token name, code value, and visible value – also called meaning, with the token prefix V)
- Information on the user that is submitted the report and time of submission that is automatically passed to every report
- All built-in objects available in the template to run SQL queries and get data from Dashboard Datasources
- Every data added by a Java Preprocessor.

You can find the result of the first example report run in Debug mode in `Samples\3 - Debug mode\1 - Report run in Debug mode.xlsx`.

Two ways are available to run a report in debug mode:

- In the report type, add the line

  `DEBUG=true`

  anywhere between the first and the last command step. As long as this parameter is present and set to true, the report will be run in Debug mode.
- Display the PPM Debug Console from the PPM user interface by holding the **Alt** key and clicking the PPM Center menu. This requires server parameter `SHOW_DEBUGGING_CONSOLE_PER_USER` to be true, and this parameter can be set directly from the Administration console. Any Excel report run while the PPM Debug Console is active will be in Debug mode.

Using a Java Preprocessor

It is possible to write a custom Java class that is invoked before filling the excel template and can pass some data to the Excel template. This can be used to get data by calling a Web Service, or use Java code to assemble a very complex SQL statement that cannot be done using a Dashboard Datasource, and be too tedious to assemble in the template using scripting.

All the values of report fields are available to the Java preprocessor. It is also possible to use the objects **ppmdb** or **reportingdb** and use method **execQuery(...)** on them. If a simple SELECT statement is not enough, it is also possible to get a JDBC Connection to PPM database or Reporting database using methods **getPPMDBConnection()** and **getReportingDBConnection()**. Do not close the Connection when you do so; it will be automatically closed when the report completes.

The following topics shows you how to create an Excel report by leveraging a Java preprocessor that takes a search text as an input field, and returns all the PPM project names containing the text as well as links from the first result page of Bing.com.

Creating the Report Type

To create the report type, follow these steps:

1. Copy the report type created in "Creating the Report Type in PPM Workbench" on page 13.
   a. In the Workbench, select the report type and click **Copy**.
   b. Name it **Java PreProcessor Excel Report**.
2. Change the prompt of report field from **Report Title:** to **Search for:**, change the attribute **Required** to **Always**, and change the Token from **TITLE** to **SEARCH_STRING**.
3. In the command steps, change the excel template path from `reports/rm/ResourcesDetailsTemplate.xlsx` to `reports/preprocessor/SamplePreprocessorTemplate.xlsx`.
4. In order to use a Java preprocessor, add the following line:

   ```
   DATA_PREPROCESSOR_CLASS=your.preprocessor.class.full.Name
   ```

   between the first and the last line of the command steps.

Because the class will be named `com.hp.ppm.excelreports.SamplePreProcessorClass`, the command steps should look like the following:

```
ksc_run_excel_report reports/preprocessor/SamplePreprocessorTemplate.xlsx
REPORT_ID=[RP.REPORT_SUBMISSION_ID]
DATA_PREPROCESSOR_CLASS= com.hp.ppm.excelreports.SamplePreProcessorClass
ksc_end_report_parameters
```

5. Save the report type after making the changes.

Creating the Java Preprocessor Class

There is only one constraint when creating the Java preprocessor class — It must extend the abstract class `com.mercury.itg.common.excel.exporter.data.ExcelReportDataPreprocessor`, which is included in the file `knta_classes.jar`, located in `<SERVER_HOME>\deploy\itg.war\WEB-INF\lib`.

All the jar files available in this folder are on the classpath when PPM is running, so you can use them as third party libraries when working on your preprocessor class.

You can find the source code of a sample preprocessor class in `Samples\4 - Using a Java pre-processor\1 - Sample Preprocessor source code\com\hp\ppm\excelreports\SamplePreProcessorClass.java`.

This code is provided for educational purposes only. It is extensively commented and demonstrates the features available from the preprocessor. You are encouraged to modify it and add your custom behavior.

> **Note:** If your PPM Server cannot access the Internet without the use of an HTTP proxy, you need to set up the proxy information in the code and recompile it. Otherwise, you can directly use the compiled class included in `Samples\4 - Using a Java pre-processor\excel-reports-sample-preprocessor.jar`.

Deploying the Java Preprocessor Class

To deploy the prerocessor class, all you need to do is to put the compiled class on calsspath of PPM. To achieve this, you can choose one of the following methods:

- Take the class file (you can extract it from the jar) and put it with its package folder hierarchy in `<SERVER_HOME>/deploy/itg.war/WEB-INF/classes`.

This should be used only for testing purpose. PPM will clean up the content of this folder during upgrade, even it is only installing a new PPM Patch release.

- Put the JAR file in `<SERVER_HOME>/lib`, or in `<SERVER_HOME>/deploy/itg.war/WEB-INF/lib` (preferred).

  PPM will clean up the content of this folder during upgrade, even it is only installing a new PPM patch release.

- Put the JAR file in `<JAVA_HOME>/jre/lib/ext`.

  However, this method is not recommended because this will make the class file available to every Java application run with this Java Virtual Machine.

After the class is deployed, start or restart your PPM Server.

Creating the Excel Template

The Excel template can be found in `Samples\4 - Using a Java pre-processor\2 - Template for Java pre-processor.xlsx`. Move it to PPM Server in the templates folder; make sure to put it in `reports/preprocessor subfolder and rename it SamplePreprocessorTemplate.xlsx`.

This template demonstrates some of the features from the JETT third-party library:

- Scripting in ${} placeholders to pluralize words when more than 1 item is in a list
- <jt:if> tag to only show table header cells if there is some data to display in the table
- <jt:hyperlink> tag to create a link with dynamic text and URL

Running the Report

Run the report from PPM and pass any search string you want.

The result of running this report on PPM demonstration environment with the search String ‘ERP’ is provided in file `Samples\4 - Using a Java pre-processor\3 - Created Report with Java pre-processor.xlsx`.

Excel Reports Security Model

Anyone with the ability to modify PPM report types and to add a file in the Excel templates directory can create a new Excel report.

The report type is used to control which PPM users can run the report. However, no security check is performed when the report is run. That means that if a PPM user does not have access to a specific project in PPM but that a report retrieve data from this project (either through a Dashboard Datasource or a direct SQL), the PPM user will be able to view this project's data in the report as long as the user can execute or view the report.

> **Note:** Because scripting is allowed in the Excel template and scripting enables users to run arbitrary code on the PPM Server, only trusted users can have authority to modify the Excel templates stored on the PPM Server. Excel templates on PPM Server should be treated with the same level of caution as JSP files.

Large Data Mode

You can find information about when and how to use large data mode in this section.

What is Large Data Mode

When filling the Excel template with the report data, the Excel reports engine needs to load the whole XLSX file into memory in order to allow complex document manipulation. As a result, if you have a lot of data to insert into your template, you risk poor performance and running out of memory.

With Large Data mode, you can insert data into the Excel report file row by row in a very efficient way, without loading the whole XLSX file into memory. However this improvement on memory and performance comes at a cost. It is not possible to use any of the features of the JETT third-party, such as scripting, placeholders, tags, cells formatting, on the data that is inserted in Large Data mode.

There are other limitations with Large Data mode. The data can only be inserted at the bottom of the first spreadsheet of the workbook; only one list of values can be inserted in a given workbook.

As a result, Large Data mode is typically used to output some large amounts of raw data that will be filtered, formatted, or rendered as charts on other spreadsheets of the workbook by leveraging all the features of Excel reports.

> **Note:** In PPM 9.3X, data inserted in Large Data Mode ignores the style of the cell with the $$placeholders$$ and uses some hard-coded styles for the text font and date formats. This behavior was changed in PPM 9.40, and data inserted in Large Data Mode will now use the same style as the cell (not the column) with the $$placeholder$$. However, if you prefer to keep the old 9.3X behavior with hard-coded styles, go to the PPM Administration Console (or your server.conf file) and set the parameter `AUTO_SET_LARGE_DATA_STYLE` to `true` (default value is `false`).

When to Use Large Data Mode

There is no precise limit as to when to start using Large Data mode. This depends on multiple factors:

- PPM Server hardware configuration

  PPM Server hardware configuration impacts on template rendering time.

- PPM Server heap memory

  More memory means that larger amount of data can be inserted in Template mode without facing memory issues.

- Excel template complexity

  If you use lots of formatting on the data you insert in the template, this will result in a much bigger XLSX file and more memory usage.

Overall, tests show that Large Data mode should be considered when more than 10,000 rows of data can be inserted in the Excel report.

How to Use Large Data Mode

In order to use Large Data mode, you only need to meet the following conditions:

- An object is available in the template containing a collection of all the objects to be inserted in Large Data mode.

  This object can be created in a scripting block of the template, or added by a Java preprocessor.

  The default name of this object is `largeData`, but you can modify it by setting a parameter `LARGE_DATA_OBJECT_NAME` in the command steps of the report type.

- Cells in the last line of the first spreadsheet of the Excel template contain the names of the attributes of the large data collection objects to be inserted, enclosed `$$`.

  It is very important that no empty cell exists after the last `$$` value. If you select the first `$$` cell of the last line and press **CTRL**+**SHIFT**+**<END>**, it should only select all the `$$` values on the last line. No empty cell should be selected on the right or below that line. Failing to do so results in a `NullPointerException` message.

Example of Large Data Excel Report

This example shows how to create a new Excel report in Large Data mode that lists all sequenced task names and the names of the work plan they belong to:

1. Create a new report type with no report field, with the following command steps:

   ```
   ksc_run_excel_report reports/largedata/SampleLargeDataTemplate.xlsx
   REPORT_ID=[RP.REPORT_SUBMISSION_ID]
   ksc_end_report_parameters
   ```

2. Create an Excel template where a block of script code will save the result of the SQL query, listing all tasks in the `largeData` variable.

   Last line of the spreadsheet contains the `$$large data mode$$` placeholders to insert the data.

You can find this template in `Samples\5 - 'Large Data' mode\1 - Template for 'Large Data' mode.xlsx`.

There are a few things to note about this template file:

- `$$large data mode$$` cells are located on the last line of the first spreadsheet. Verify that only the three `$$` cells are selected by selecting the first `$$` cell and pressing **CTRL**+**SHIFT**+**<END>**,
- As long as you are not in a `$$` cell, all the tags and placeholders will work. It is actually possible to dynamically generate the last line with `$$` values, because the XLSX template file will first be rendered like any Excel report, and if a large data object exists, the report engine will look for `$$` cells on the last line of the produced report and will insert the data there using Large Data mode.
- In the template, the `largeData` object is initialized using a SQL query. The `largeData` object must be a collection; each object in the collection must be either a `ResultSetRow` (the method `execQuery()` returns a List`<ResultSetRow>`) or a `Map<String, Object>`. If using a `Map<String, Object>`, a cell `$$ MY_VALUE $$` will be replaced by the value of invoking `map.get("MY_VALUE")`.
- The `largeData` object is initialized with the data from the SQL query with the `${{ … }}` placeholder rather than the standard `${…}` placeholder.

  The only reason is that the cell value will be filled with whatever value is returned by the first expression in the placeholder. If you only have the code `${ largeData = ppmdb.execQuery(…) }`, the cell is filled with the result of the SQL, or more exactly with the dump to strings of all the objects in the collection, which is too much for Excel to handle.

The result of running this report on PPM can be found in `Samples\5 - 'Large Data' mode\2 - Created Report with 'Large Data' mode.xlsx`.

Using a Java Report Postprocessor

What is a Java Report Postprocessor

A Java Report Postprocessor is a java class that will run right after the Excel template was filled with data, and will allow manipulating the XLSX file using Apache POI XSSF API.

This allows basically every operation on the XLSX file (modify charts, add pictures, modify styles, and etc.).

You can retrieve all the data available to the report from the Java code, and add more data that will be available to the Java Report Streaming Class. You can also use the same database-related tools as in the Java Preprocessor Class.

When to Use Java Report Postprocessor

A Java Report Postprocessor should be used when complex operations are required on the XLSX file, and when including these operations as script in the template represent a maintenance challenge.

However, a Java Report Postprocessor is not necessary if you use Large Data Mode to just insert large amounts of data in the document, as the Report Postprocessor requires loading the whole document to memory for processing.

How to Use Java Report Postprocessor

Creating the Report Type

In the report type's command steps, anywhere between the first `ksc_run_excel_report` line and the last `ksc_end_report_parameters` line, add the parameter `REPORT_POSTPROCESSOR_CLASS`, with the value being the fully qualified class name of your Postprocessor class. For example, for the class `com.hpe.ppm.excelreports.SamplePostProcessorClass` that you can find in the attached bundle in the folder `6 - Using a Java report post-processor`, the line would be:

```
REPORT_POSTPROCESSOR_CLASS=com.hpe.ppm.excelreports.SamplePostProcessorClass
```

Creating the Java Report Postprocessor Class

You should create a Java class that extends the existing abstract class `com.mercury.itg.common.excel.exporter.data.ExcelReportWorkbookPostprocessor`. The single method to implement is: `public void processWorkbook(XSSFWorkbook wb)`

You can then directly work on the passed XSSF workbook object to modify the report XLSX document.

For more information, you can refer to the sample postprocessor class `SamplePostProcessorClass.java` located in the attached bundle (in the path `6 - Using a Java report post-processor\1 - Sample Report PostProcessor source code\com\hpe\ppm\excelreports`).

Deploying the Java Report Postprocessor Class

Deploy the Java Report Postprocessor class in the same way as you deploy a Preprocessor class. See "Deploying the Java Preprocessor Class" on page 26.

Creating the Excel Template

There is nothing special to do on the excel template if you are using a Java Report Postprocessor. The only limitation is that you cannot fill the template using Large Data Mode.

Example of Report Using a Postprocessor

You can find an example of an Excel template and the resulting report in file `Samples\6 - Using a Java report post-processor`. The source code and a jar version of the Java Postprocessor are included.

Using a Java Report Streaming Class

What is a Java Report Streaming Class

A Java Report Streaming class is a java class that will run at the end of the Excel Report process, and will allow manipulating the XLSX file using Apache POI SXSSF (Streamed XSSF) API.

This allows inserting data in the template row by row with high performance and low memory consumption, at the expense of limiting the set of POI features available.

You can retrieve all the data available to the report from the Java code. You can also use the same database-related tools as in the Java Preprocessor class.

When to Use Java Report Streaming Class

A Java Report Streaming class, instead of Large Data Mode, should be used when one requires more advanced control on the style of data inserted, or needs to insert data on any spreadsheet of the workbook. It allows the same level of performance and memory usage as the Large Data Mode.

How to Use Java Report Streaming Class

Creating the Report Type

In the report type's command steps, anywhere between the first `ksc_run_excel_report` line and the last `ksc_end_report_parameters` line, add the parameter `REPORT_STREAM_DATA_CLASS`, with the value being the fully qualified class name of your Postprocessor class. For example , for the class `com.hpe.ppm.excelreports.SampleSteamingClass` that you can find in the attached bundle in folder `7 - Using a Java report streaming class`, the line would be:

```
REPORT_STREAM_DATA_CLASS=com.hpe.ppm.excelreports.SampleSteamingClass
```

Creating the Java Report Streaming Class

You should create a Java class that extends the existing abstract class `com.mercury.itg.common.excel.exporter.data.ExcelReportStreamData`. The single method to implement is: `public void streamDataToWorkbook(SXSSFWorkbook wb)`.

You can then directly work on the passed SXSSF workbook object to modify the report XLSX document, but keep in mind that you can only add rows to the spreadsheets with this API.

For more information, you can refer to the sample streaming class `SampleStreamingClass.java` located in the attached bundle (in the path `7 - Using a Java report streaming class\1 - Sample Report Streaming class source code\com\hpe\ppm\excelreports`).

Deploying the Java Report Streaming Class

Deploy the Java Report Streaming class in the same way as you deploy a Preprocessor class. See "Deploying the Java Preprocessor Class" on page 26.

Creating the Excel Template

There is nothing special to do on the excel template if you are using a Java Report Streaming Class. Just ensure that there are no empty rows already written at the end of the sheets you want to modify. You can remove them first with a Java Report Postprocessor class.

Example of Report Using a Java Streaming Class

You can find an example of an Excel template and the resulting report in file `Samples\7 - Using a Java report streaming class`. The source code and a jar version of the Java Streaming class are included.

Frequently Asked Questions

- **Question**: I get data from a Dashboard Datasource in my template, but the cell's formatting does not work on these fields. Why?

  Answer: By default, a Dashboard Datasource returns every value as string. The development team try to convert string representing numbers to actual numbers (using the server locale for parsing), but nothing is done for date. You can either change the datasource to return already formatted values, or switch to a SQL query, where type of returned data is the same as the SQL type. Another option is to use scripting to parse the string values returned by the datasource into whatever type you need; however the scripting code might be complex.

- **Question**: What is the object “verticadb” available in the template? It errors whenever I try to use it.

  Answer: This object has been created in prevision of a possible future Vertica-based Big Data reporting and analytics solution for PPM. It is not to be used for now.

- **Question**: There is an error when running my Excel report, and I canot figure out where the error comes from. How to identify the problem?

  Answer: Troubleshooting a faulty report template can be a very tricky task, especially if someone does not have access to PPM Server logs, where more information may be provided. If that happens, you can try to remove parts of the templates until the report works again, and then focus on the last removed part to see whether a change in syntax can fix the problem. If that is not enough to pinpoint the root cause, another option is to start from a blank template and add content little by little until the error can be reproduced.

- **Question**: I try to insert data from PPM database in an Excel table. Even though my tags are in the table, when the report is created, the Excel table is not automatically expanded to cover all the data inserted. Is that a bug?

  Answer: No, this is by design. During report generation, the reports engine will try to maintain the formatting of source cells, but it does not automatically expand Excel tables. For this reason, you should convert the Excel table to standard cells in the template by right clicking the table and select **Table / Convert to Range** . Another option is to use a Named Range as the source data of your Excel table, like demonstrated in "Adding a Pivot Table and a Chart" on page 21. However, make sure not to include the data headers, as the Excel table has its own headers that are not part of the source data.

- **Question**: What are the supported versions of Excel? Can it be used with other spreadsheet applications?

  Answer: Excel reports work with any Microsoft Excel versions supporting XLSX format, which means Excel 2007 and later. However, it is recommended to use the same version of Excel for viewing the reports as the version that was used to create the templates. Examples bundled with this document have been created with Excel 2013. Any other application supporting XLSX format can be used to create templates and view Excel reports, to the extent of the supported features, such as charts and pivot tables. That includes tools such as LibreOffice, OpenOffice, or Google Docs.

- **Question**: I'm using Large Data mode, but the report keeps failing with a `NullPointerException` message. When I select the last line with `$$` values with the shortcut **CTRL**+**SHIFT**+**<END>**, it selects some blank cell on the right or under the last line, but I cannot get rid of them. How to proceed?

  Answer: You need to copy all your cells with value in a new spreadsheet, and make sure that no cell from your spreadsheet has value if it is located on the right of the last `$$` cell. This very tedious constraint might be fixed in a future PPM version.

- **Question**: When I open an Excel Report with a pivot table in Internet explorer, I'm getting an error message stating "We couldn't get the data from …".

  Answer: If you download the PPM Report in Internet Explorer and use the "Open" menu instead of "Save", the report file will be opened in read-only mode directly from the PPM Server. It won't be saved locally, and this will cause pivot datasources not to load correctly when opening the file in Excel 2013 and later. If your report has dynamic pivot tables, please use the "Save" or "Save as" menu when opening PPM Excel Reports in Internet Explorer.

Related Links

Join the conversation and ask your questions on:

- PPM Customer Support forum (if you are an existing HPE PPM Center customer)
- Public PPM Support and News forum

You can find a list of sample files used with this guide at HPE Live Network.

Send documentation feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Excel Reports Cookbook (Project and Portfolio Management Center 9.40)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to your_IE_team_PDL@hpe.com.

We appreciate your feedback!