**Using the Excel Utilities Service**

This chapter discusses the following:

**• Excel Utilities Service**

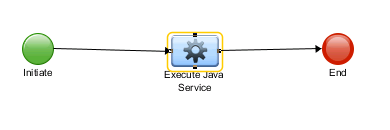
**Excel Utilities Service**

The following xCelerator enables xCP workflows to read and write Microsoft Excel files. It supports Office 97-2003 and 2007 (\*.xlsx) files (\*.xls).

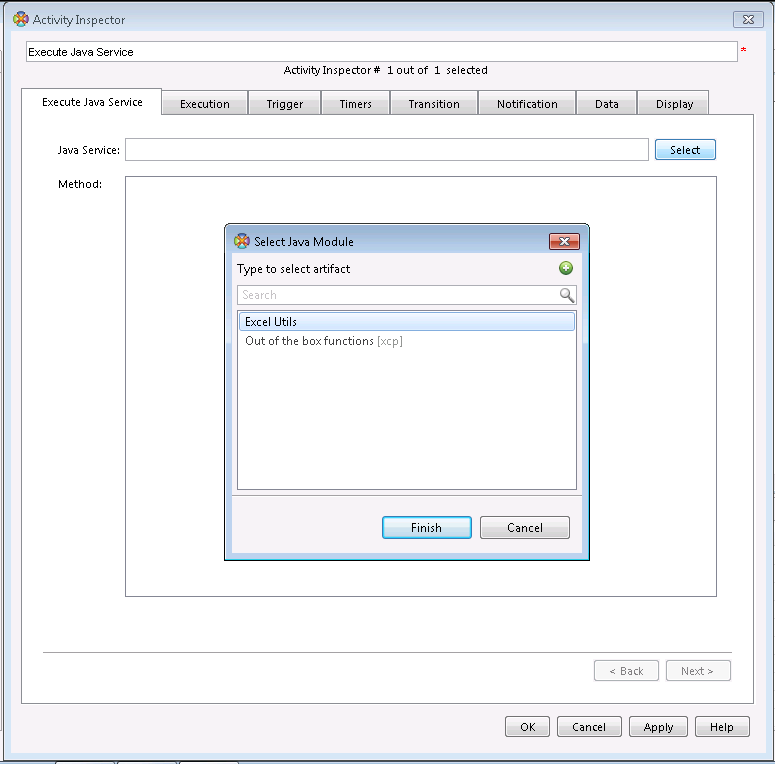
Reading Excel files can be a significant advantage of an xCP solution, as this format is very popular among business users. Populating sheets on the other hand introduces a new level of potential interaction with system users. Potential applications include reporting (with graphs and sophisticated formulas based on the populated data) and interaction with users or systems that do not directly interact with the Documentum system (by sending populated Excel sheet outside).

***Configuration***

When using the Excel Utilitilies service within your process, you must select the service to be executed by the Java Execute Service activity. First, add the Java Execute Service to your workflow.



Open the activity, and choose the Java Service

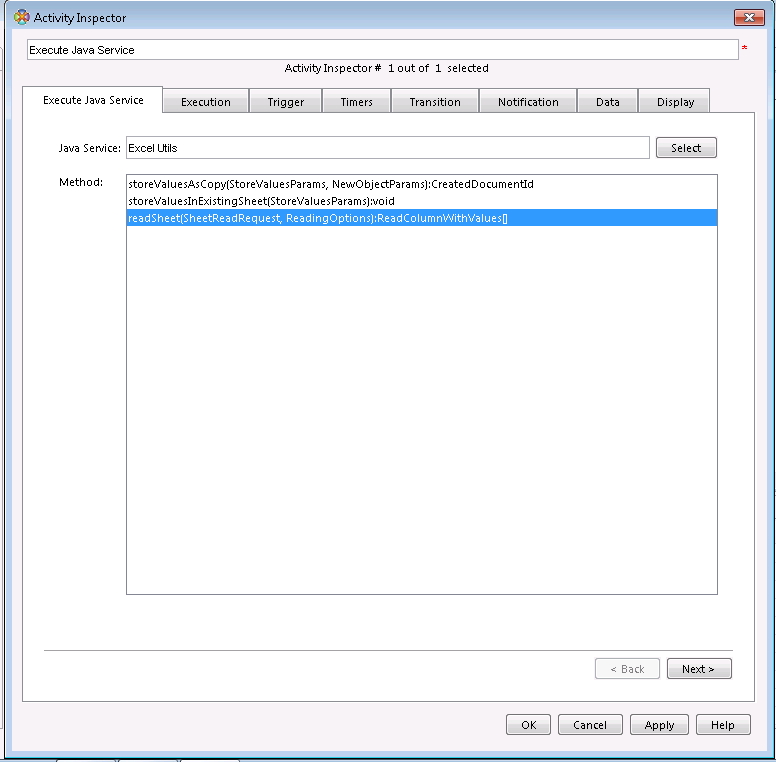


**Reading Excel sheets**

The *ExcelUtil* xCelerator enables xCP workflows to read values from a particular sheet in an excel workbook.

**Module configuration**

Reading an Excel sheet is provided by the readSheet method:

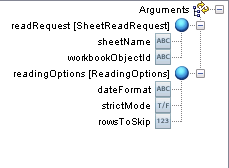


**Input parameters**

Two arguments are required by the module, both are attributes of the SheetReadRequest bean:

1. workbookObjectId: the r\_object\_id of the repository object containing the Excel sheet to be read

2. sheetName: name of the worksheet. It must correspond to an existing worksheet (such as *New employees* on the *Image 8 Sample sheet containing data to be read*).



Optional parameters can be specified in the ReadingOptions bean:

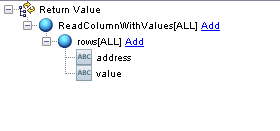
1. dateFormat: specifies the format of the date fields read from the sheet

2. rowsToSkip: specifies how many leading rows should be skipped when a sheet is read. Particularly useful if the first rows include columns’ titles. Setting this value to one when reading the sheet from Image 8 Sample sheet containing data to be read will cause row 1 being ignored.

3. strictMode: if set to true, values that cannot be read (of type that is not supported by the xCelerator) will be ignored. The default value is true, which causes an exception being thrown in such case.

**Result retrieving**

Result is delivered as an array of ReadColumnWithValues beans. Each of them represents a column read from the sheet. Result cells are available in an array called rows. Each entry in this array contains a particular cell read from the column. Two values are provided for each cell: its address (the address attribute) and the actual value (the value attribute).

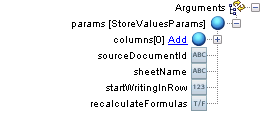


ReadColumnWithValues[0] represents column A, ReadColumnWithValues[1] represents column B and so on. Cell C2 would be accessible in ReadColumnWithValues[2]/rows[1].

**Populating sheets with values**

This part of the xCelerator enables xCP workflows to manipulate Excel files’ contents. There are two result persistence options available: storing the result (i.e. sheet populated with values) as the same repository object, or creating a new one.

The following arguments are common for sheet manipulations, regardless of the result persistence option selected:



**Image 11 Basic options**

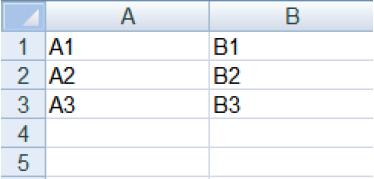
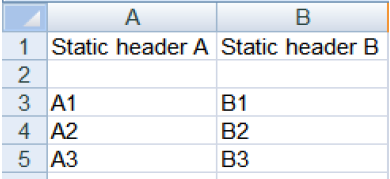
1. sourceDocumentId (required): r\_object\_id of the repository object containing Excel sheet to be filled in.

2. sheetName (required): name of the worksheet in the Excel workbook:



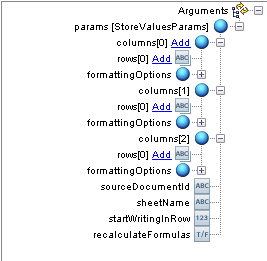
The value passed to sheetName must correspond to an existing sheet (Sheet1, Sheet2, Sheet3 in the example), an exception is thrown in the runtime otherwise.

3. startWritingInRow (optional): specifies the first row that would be filled with data (0 –based). Useful if first rows contain data that should not be overwritten (such as headers).

4. recalculateFormulas (optional): if set to true, all formulas in the workbook will be recalculated after values have been set. If a sheet contains formulas basing on values written by the xCelerator, this should be set to true; formulas will not be recalculated automatically by Excel and will contain obsolete values otherwise.

5. columns: the actual data that is supposed to be written in the sheet:



In order to fill a sheet as in example on the Image 13, the following configuration is required:

columns[0]/rows[0]=A1

columns[0]/rows[1]=A2

columns[0]/rows[2]=A3

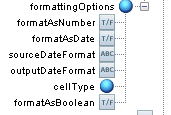
columns[1]/rows[0]=B1

columns[1]/rows[1]=B2

columns[1]/rows[3]=B3

This can be easily achieved by mapping data from an electronic form (see the *Sample workflow*)

6. columns[i]/formattingOptions (optional): specifies format of the cells in the column. Values are stored as text by default, but this behaviour can be modified by selecting one of the formatAsXXX options:



The following options can be set:

a. formatAsNumber: cell will be formatted as number

b. formatAsBoolean: cell will be formatted as a true/false value

c. formatAsDate: cell will be formatted as date. This option requires the following parameters to be provided as well:

i. sourceDateFormat: source format in which date value is passed to the rows parameter

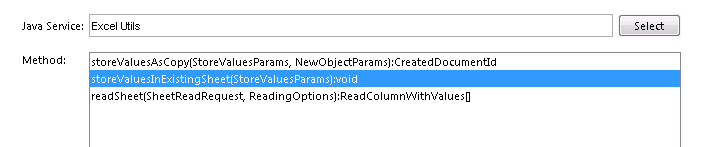
ii. outputDateFormat: format in which date is to be stored in Excel cells in the column

**Result persisting options**

Depending on the module’s method selected, the result (modified Excel file) will overwrite the source object, or create a new one.

**Overwriting the template object**

In order to overwrite the source sheet with the result, the storeValuesInExistingSheet method needs to be selected in the *BOF Module Configuration* tab:

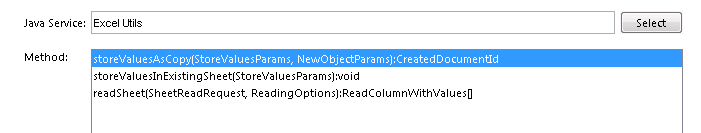


No additional configuration is required.

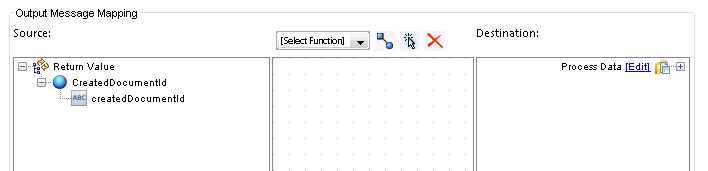
**Creating a new object for the result**

This approach should be used if the source Excel sheet is to be treated as a read-only template. In order to use this approach, the storeValuesAsCopy method needs to be selected in the *BOF Module*

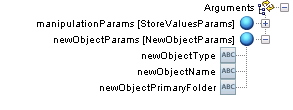
*Configuration* tab:

****

The resulting object can be mapped into a workflow package from the CreatedDocumentId result bean:



Additional configuration options in regards to the new object can be specified (all of them are optional):



1. newObjectType: repository type of the new object. An instance of dm\_document is created by default

2. newObjectName: object\_name of the created object.

3. newObjectPrimaryFolder: primary folder of the created object