Big Excel Generator

On this Page:

With the **BigExcel Generator** you can create Excel documents out of simple structured bulk data.

This adapter generates "simply" structured Excel 97-2003 or Excel 2007 documents. If you have to create more complex Excel documents, have a look at Excel Generator.

The BigExcel Generator uses the HSSF and XSSF component of Apache POI.

Example File (Builder project BigExcelGenerator):

<your example path>\Libraries\BigExcelGenerator.zip

Procedure for Using the BigExcel Generator

The generation of Excel documents involves the following steps. We recommend to implement all workbook handling to a sub-activity (see step 2).

1. Create an Excel 97-2003 (**newXlsWorkbook()**) or Excel 2007 (**newXlsxWorkbook()**) workbook.
2. Create a sub-activity and pass the workbook ID. Implement all workbook handling here, like
   - Specify types for the cells of the subsequent rows (**setTypes()**).
     This step is optional. If you do not set types, the Excel default types will be used.
   - Specify formats for the cells of the subsequent rows (**setFormats()**).
     This step is optional. If you do not set formats, the Excel default formats will be used.
   - Add data to the excel sheet (**addRow()**).
   - Write the Excel document (**generateExcel()**).
3. After the sub-activity, implement some error handling.
4. Release the resources used by the created workbook (**dispose()**).

   It is important to release the used resources using dispose() after having generated the Excel file as well as in case of error. Not releasing them may lead to unexpected side effects.

In order to minimize the data transfer between the JVM and the xUML Runtime, a unique id returned by **newXlsWorkbook()** / **newXlsxWorkbook()** is used to reference the workbook in any other operation call.

Operations of the BigExcel Generator

While the **newXlsWorkbook()** operations require the whole Excel document data structure to be hold in memory, the **newXlsxWorkbook()** operations require to hold only a certain number of rows in memory, while the other parts of the Excel document are swapped to a temporary file.

newXLSWorkbook Operations

`newXlsWorkbook() : String`

creates an Excel 97-2003 workbook

Parameter	Direction	Description
workbookId	return	Reference to the created workbook.

`newXlsWorkbook( sheetName : String ) : String`

creates an Excel 97-2003 workbook containing a single worksheet named **sheetName**

Parameter	Direction	Description
sheetName	in	Name of the worksheet to be contained in this workbook.
workbookId	return	Reference to the created workbook.

`newXlsWorkbook( sheetNames : String[] ) : String`

creates an Excel 97-2003 workbook

Parameter	Direction	Description
sheetNames	in	An array with the names of the worksheets to be contained in this workbook.
workbookId	return	Reference to the created workbook.

`newXlsWorkbook( numberOfSheets : Integer ) : String`

creates an Excel 97-2003 workbook containing *numberOfSheets* worksheets

Parameter	Direction	Description
numberOfSheets	in	Number of the worksheets to be contained in this workbook.
workbookId	return	Reference to the created workbook.

`newXlsxWorkbook( rowWindowSize : Integer, compressTempFile : Boolean ) : String`

creates an Excel 2003 workbook

Paramter	Direction	Description
rowWindowSize	in	Number of rows to be hold in memory (if -1, all rows will be hold in memory).
compressTempFile	in	if true, the temporary file will be compressed.
workbookId	return	Reference to the created workbook.

`newXlsxWorkbook( sheetName : String, rowWindowSize : Integer, compressTempFile : Boolean ) : String`

creates an Excel 2003 workbook containing a single worksheet named *sheetName*

Parameter	Direction	Description
sheetName	in	Name of the worksheet to be contained in this workbook.
rowWindowSize	in	Number of rows to be hold in memory (if -1, all rows will be hold in memory).
compressTempFile	in	If *true*, the temporary file will be compressed.
workbookId	return	Reference to the created workbook.

`newXlsxWorkbook( sheetNames : String[], rowWindowSize : Integer, compressTempFile : Boolean ) : String`

creates an Excel 2003 workbook

Parameter	Direction	Description
sheetNames	in	An array with the names of the worksheets to be contained in this workbook.
rowWindowSize	in	Number of rows to be hold in memory (if -1, all rows will be hold in memory).
compressTempFile	in	If *true*, the temporary file will be compressed.
workbookId	return	Reference to the created workbook.

`newXlsxWorkbook( numberOfSheets: Integer, rowWindowSize : Integer, compressTempFile : Boolean ) : String`

creates an Excel 2003 workbook containing *numberOfSheets* worksheets

Parameter	Direction	Description
numberOfSheets	in	Number of the worksheets to be contained in this workbook.
rowWindowSize	in	Number of rows to be hold in memory (if -1, all rows will be hold in memory).
compressTempFile	in	If *true*, the temporary file will be compressed.
workbookId	return	Reference to the created workbook.

All of the above **newXlsxWorkbook()** operations have variants without **rowWindowSize** and **compressTempFile** parameters. **rowWindowSize** defaults to 100, **compressTempFile** defaults to *false*.

setTypes Operations

`setTypes( workbookId : String, sheetName : String, types : String[])`

sets the type information for the subsequently added rows

Parameter	Direction	Description
workbookId	in	The id of the workbook.
sheetName	in	The name of the worksheet.
types	in	An array of the types of the cells (valid values: "Text", "Boolean", "Number", "IsoDateTime", "TimeTicks" or NULL).

`setTypes( workbookId : String, sheetIndex : String, types : String[])`

sets the type information for the subsequently added rows

Parameter	Direction	Description
workbookId	in	The id of the workbook.
sheetIndex	in	The index of the worksheet.
types	in	An array of the types of the cells (valid values: "Text", "Boolean", "Number", "IsoDateTime", "TimeTicks" or NULL).

setFormats Operations

`setFormats( workbookId : String, sheetName : String, formats: String[])`

sets the format information for the subsequently added rows

Parameter	Direction	Description
workbookId	in	The id of the workbook.
sheetName	in	The name of the worksheet.
formats	in	An array of the formats of the cells.

`setFormats( workbookId : String, sheetIndex : String, formats: String[])`

sets the format information for the subsequently added rows

Parameter	Direction	Description
workbookId	in	The id of the workbook.
sheetIndex	in	The index of the worksheet.
formats	in	An array of the formats of the cells.

Using **setTypes()** and/or **setFormats()** is preferable to using **types** and **formats** in **addRow()**. This reduces the amount of data to be transferred between the JVM and the xUML Runtime by each call of **addRow()**.

addRow Operations

`addRow( workbookId : String, values : String[], types : String[], formats : String[] )`

add a new row to the first worksheet of the workbook referenced by workbookId

Parameter	Direction	Description
workbookId	in	The id of the workbook
values	in	An array holding the values of the cells in that row
types	in	An array holding the types of the cells in that row; will temporarily override any values set by **setTypes()**
formats	in	An array holding the formats of the cells in that row; will temporarily override any values set by **setFormats()**

`addRow( workbookId : String, sheetName : String, values : String[], types : String[], formats : String[] )`

add a new row to worksheet *sheetName* of the workbook referenced by workbookId

Parameter	Direction	Description
workbookId	in	The id of the workbook
sheetName	in	The name of the worksheet
values	in	An array holding the values of the cells in that row
types	in	An array holding the types of the cells in that row; will temporarily override any values set by **setTypes()**
formats	in	An array holding the formats of the cells in that row; will temporarily override any values set by **setFormats()**

`addRow( workbookId : String, sheetIndex : Integer, values : String[], types : String[], formats : String[] )`

add a new row to worksheet *sheetIndex* of the workbook referenced by workbookId

Parameter	Direction	Description
workbookId	in	The id of the workbook
sheetIndex	in	The index of the worksheet
values	in	An array holding the values of the cells in that row
types	in	An array holding the types of the cells in that row; will temporarily override any values set by **setTypes()**
formats	in	An array holding the formats of the cells in that row; will temporarily override any values set by **setFormats()**

There are variants of the **addRow()** operations that do not have **types** and **formats** parameters.

generateExcel Operations

> It is important to release the used resources using dispose() after having generated the Excel file as well as in case of error. Not releasing them may lead to unexpected side effects.

`generateExcel( workbookId : String, filename : String )`

writes the Excel document to file *filename*

Parameter	Direction	Description
workbookId	in	The id of the workbook
filename	in	The name of the Excel file to be written

`generateExcel( workbookId : String ) : Blob`

writes the Excel document to a blob

Parameter	Direction	Description
workbookId	in	The id of the workbook
result	return	The blob to be written

`dispose( workbookId : String )`

releases the resources bound to this workbook

Parameter	Direction	Description
workbookId	in	The id of the workbook

You can extract the **javadoc** folder from **excelgenerator.jar** to get additional information on the Java methods wrapped by this adapter.