QPR PD/EA Excel Integration

User's guide

Table of Contents

1 Introduction

QPR PD/EA Excel Integration add-on enables implementation of analysis and reporting solutions over Microsoft Excel. This support is realized by scripts for import and export of model structures between QPR ProcessDesigner or QPR EnterpriseArchitect and Microsoft Excel.

Solution-specific integrations are configured using a declarative reporting language. The reporting language declares what kinds of model elements, attributes, relations and connectors between the elements are exported or imported between QPR PD/EA and Microsoft Excel. The configuration and solution-specific integrations is done by solution experts. End-users of the implemented reporting and analysis solutions need not to be aware of the configuration details or the reporting language.

This document provides instructions for configuring and utilizing QPR PD/EA Excel Integration solutions.

2 Add-on requirements

- QPR ProcessDesigner / EnterpriseArchitect 2015.1 (or newer)
- Microsoft Excel client
- .Net Framework 3.5 support

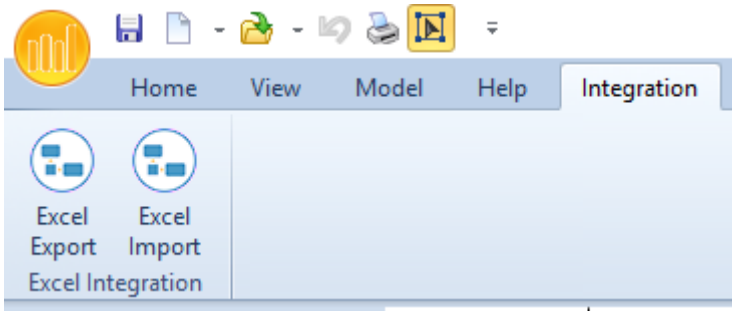
The scripts have been developed and tested over Microsoft Excel 2013.

3 Add-on installation

The add-on is implemented as two separate scripts for export and import. The scripts are included in the add-on package. The export script is named *ExcelIntegrationExport.qprpsc* and the import script is named *ExcelIntegrationImport.qprpsc*.

The scripts should be copied to a place in the file system that is accessible by their intended users. For example, the default location for QPR EA 2022.1 scripts is `C:\ProgramData\QPR Software\QPR 2022\2022.1\Clients\pgscripts`, assuming the client is installed in default location).

The scripts are usable after copying to appropriate place. No other configuration needs to be done. After successful installation the export and import functionality is available through "Integration" tab in the QPR client:



The scripts require .Net Framework 3.5 support. **Having .Net Framework 4 support is not sufficient due to changes in the framework and their default configuration.** In Windows 10 systems this support must be enabled explicitly. The .Net Framework 3.5 is enabled in Windows 10 systems as follows:

- Start *Turn Windows features on or off* application;
  - E.g. open start window and type *turn windows features* and select the application from the suggestions appearing on the right-hand part of the screen.

- Click the checkmark on for feature "*.NET Framework 3.5 (includes .NET 2.0 and 3.0)* (see Figure X below);
- Click *OK* on the dialog;
- Now Windows starts installing the .NET Framework 3.5 support.

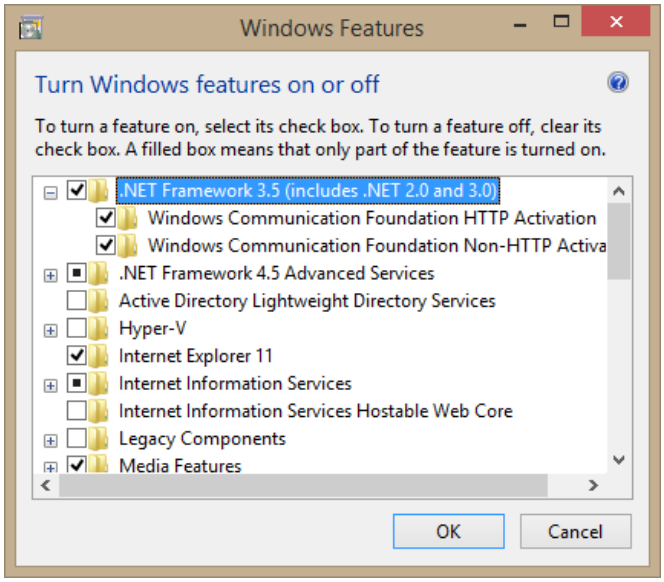


Figure 1: Enabling .NET Framework 3.5 support.

3.1 Deploying scripts as procedures

The scripts can be deployed to be run as procedures as well. For this purpose one needs sufficient rights for creating new model element types for the required procedures and configuring the template accordingly.

The contents of the scripts can then be copied to the created procedures, and customized appropriately. Using the scripts with appropriate customizations may significantly enhance the user experience of domain-specific Excel integration solutions. Implementation of such solutions is not described in this document.

4 Operation principle

The operation of the QPR PD/EA Excel Integration is based on interpretation of report specifications, typically retrieved from Excel workbooks, and retrieval, update or creation of model structures in QPR PD/EA models.

Typical workflow for exporting model structures to Microsoft Excel is as follows:

1. Solution provider (e.g. consultant) creates an Excel workbook that contains appropriate report specifications. Each report specification corresponds to a sheet in the Excel workbook to be generated. Report specifications are declared in a sheet named *QPR Configuration*; each sheet is defined in a distinct row.
2. End-user is running QPR ProcessDesigner or QPR EnterpriseArchitect and has opened a model. Now the end-user executes the *ExcelIntegrationExport.qprpsc* script (or same functionality provided as a procedure).
3. In the default case, the PD/EA client will now ask for an Excel reporting template to be used. From the file dialog, the end-user selects the Excel workbook created by the solution provider.
4. The *ExcelIntegrationExport.qprpsc* script reads the Excel workbook. If it finds a sheet named *QPR Configuration,* it reads the report specifications contained.

5. The export script crawls the active model for each report specification and creates sheets to hold the model elements conforming to the specification.
6. The script notifies the user when the export is ready; the Excel workbook is automatically saved to the home folder of the user with a distinctive name.

Typical workflow for importing model structure from Microsoft Excel to QPR PD/EA model is as follows:

1. Solution provider (e.g. consultant) creates an Excel workbook that contains appropriate report specifications. Each report specification corresponds to a sheet in the Excel workbook to be generated. Report specifications are declared in a sheet named *QPR Configuration*; each sheet is defined in a distinct row.
2. Data is created to sheets conforming to the report specifications. The data can be provided either using the export script, manually, or from 3rd party systems using an appropriate integration to Excel.
3. End-user is running QPR ProcessDesigner or QPR EnterpriseArchitect and has opened a model. Now the end-user executes the *ExcelIntegrationImport.qprpsc* script (or same functionality provided as a procedure).
4. In the default case, the PD/EA client will now ask for an Excel reporting template to be used. From the file dialog, the end-user selects the Excel workbook with all the data in the sheets.
5. The import script goes through all the Excel sheets, and updates the model contents of the active model according to the contained report specifications.
6. The script notifies the user when the import is ready. The active model now contains the data corresponding to the Excel workbook.

The script implementations are based on the QPR API (see *QPR Developer's Guide*). Thus the constraints related to reading, setting or updating element attribute values are the same as with the API. For example, the *Owner* system relations cannot be set through the API, and thus, not through the *ExcelIntegrationImport.qprpsc* script. It is the responsibility of the integration solution provider to take these constraints into account, and design the report specifications and Excel workbooks in such a way (e.g. using Excel cell protection) that the end-user experience becomes flawless.

5 Reporting language specification

Analysis and reporting solutions are configured with a reporting language. The reporting language is a declarative, domain-specific language that is used for specifying what kind of model structures can be exchanged between QPR PD/EA and Microsoft Excel.

The structure and contents of the report specification blocks are described in the following. In the grammar description, the following notational conventions are used:

- Keywords are written in bold: e.g. **report** or **columns**;
- Identifiers are denoted with ID, where an identifier begins with an alphabet and may contain numbers after the first character, e.g. MyIdentifier;
- Integers are denoted with INT, e.g.: 12345;
- Quoted strings are denoted with STRING, e.g.: "This is a string";
- Alternatives between elements are denoted with bar '|': e.g. True | False denotes a choice between boolean constants for true and false;
- Multiplicity of elements:
  - ? = optional (i.e. at most one)
  - * = zero or more
  - + = at least one
    E.g. ID+ denotes one or more identifiers
- Grouping of elements is declared with parentheses: INT (. INT)? specifies a floating point

Each report specification is contained within a report block. A report specification contains an optional set of report options, at least one columns specifications, and at least one query specifications. The basic structure for report specifications is given below. A report name may not contain any whitespaces. The report name corresponds to a sheet name in an Excel workbook. The structure of a report specification is described below.

report *ReportName* **{**

Options

(Value mappings)?

Column specifications

(Hierarchy specifications)?

Query specifications

}

ReportName is a string that does not contain any whitespaces. *Option*, *Column specifications* and *Query specifications* refer to the corresponding specification block descriptions described in subsections below.

Section 7 includes example queries that apply all the features discussed below.

5.1 Options block

One or more report options can be declared in a report specification. Options block may contain one or more *Option* elements, as described in Table 5.1 Options block. Each *Option* is separated from each other by a semicolon.

Options: Option (**;** *Option*)*

Option:
CreateInstanceAlways (**= True** | **False**)?
| **CreateInstanceGrouping** (**= True** | **False**)?
| **ExcelSheetName =** STRING
| **ExportOnly**
| **ImportOnly**
| **InstanceDiagramType =** STRING
| **InstanceHolderCollection** = STRING
| **Language =** STRING
| **StartRow =** INT
| **StartColumn =** INT

Table 5.1 Options block

The *Option* element may be any of the alternatives described in Table 5.2 Option values.

- **CreateInstanceAlways** (**= True** | **False**)?: Create an instance to the instance diagram always during an import. Default value is True, if not defined explicitly. Used during import.
- **CreateInstanceGrouping** (**= True** | **False**)?: Create group elements in the instance diagram around created instances. Default value is True, if not defined explicitly. Used during import.
- **ExcelSheetName =** STRING: Declares the name of the Excel-sheet to be used as a source (import) or target (export) of the report query. Is utilized especially in situations where import and export is executed using separate report query specifications.

- **ExportOnly:** Declares that the results of the report are not to be imported. Enables creation of read-only reports that may have e.g. results of multiple queries within a single report (corresponds to an Excel sheet).
- **ImportOnly:** Declares that the report specification defines a read-only query, i.e. it shall not be used during exports.
- **InstanceDiagramType =** STRING: Name of the instance diagram type. E.g. "*Application Area Diagram*". Used during import to declare the name for generated target diagram.
- **InstanceHolderCollection** = STRING: Name of the diagram collection that will hold the created instance diagram. E.g. "*Application Areas*". Used during import.
- **StartColumn =** INT**:** Defines the start column in Excel sheet used for importing and exporting elements. Defaults to 0.
- **StartRow** = INT**:** Defines the start row in Excel sheet used for importing and exporting model elements. Defaults to 0.
- **Language** = STRING**:** Declares the modeling language to be used during import. The identifier must be one of the language identifiers available in the QPR repository, e.g. *"ENG"* or *"FIN"*.

Table 5.2 Option values

Example:

```
report ApplicationImport {
    ExcelSheetName=”Application Portfolio”; StartRow = 5; StartColumn = 2; Language = “ENG”; ImportOnly
    …
}
```

5.2 Value mapping block

Value mapping block contains declarations used for definition of value mappings. A value mapping enables e.g. conversion of values between Excel representation and QPR EA model representations.

Value mappings: (Value mapping)+

Value mapping: **valuemapping** *ValueMappingName* **= {** *MappingPair+* **}**

ValueMappingName: ID

MappingPair: ColumnValue : ElemAttributeValue

ColumnValue: STRING

ElemAttributeValue: STRING

Value mappings are used in element queries with value mapping usage declarations (see Section 5.5).

Example query which during import converts "0" to "Failed" and "1" to "Passed" and during export converts "Passed" values to "1" and "Failed" values to "0":

```
report AllControlStatuses {
 valuemapping StatusConversion = {
  "1" : "Passed",  "0" : "Failed" }
  columns {
    "Control Name" : ctrlname, "Control status" : ctrlstatus
```



```
  }
  queries {
   [Control]{ name => ctrlname, control test outcome => ctrlstatus}
using StatusConversion on { ctrlstatus };
   }
}
```

5.3 Column specifications block

Columns specifications block contains declarations that are used for mapping model elements attributes to columns in an Excel sheet. The columns specifications are given within a block denoted by keyword **columns**. The grammar for columns specifications is described in Table 5.3 Columns specifications.

Column specification: **columns {** *ColumnSpec+* **}**
ColumnSpec: STRING **:** ID (**=** *DefaultValue)? (@ PropertyName)?*
DefaultValue: Either a quoted string or integer value.
PropertyName: ID

Table 5.3 Columns specifications

Each column specification is of form *"Column name" : ColumnId (=DefaultValue)? (@ PropertyName)*. The column name corresponds to a column name in the Excel sheet, while *ColumnId* is a column identifier that is used in query specifications to assign values between Excel cells and model element attributes. A default value can be assigned to the column; this is used during imports for setting values if no value has been defined in the corresponding Excel cell.

A column can be declared as a reference attribute column with the *@PropertyName* notation. In this case, the value in the column cell is interpreted as a name or symbol of an element and a reference is set to the found element using a property declared in *PropertyName*. This notation allows e.g. setting several reference attribute values, with optional empty values, to elements without explicit link specifications (see Section 5.5).

Column specifications are separated from each other by commas (**,**).

Example:

```
columns {
   “Application Name” : applname, “Description” : appldesc,
“Application lifecycle state” : applstate = “In production”, “Owner” :
applowner@owner
}
```

The example defines columns for describing name, description and lifecycle state for applications. In addition, references to application owners are set in the “Owner”-column.

5.4 Hierarchy specifications block

Hierarchy specifications block enables utilization of hierarchical Excel-sheets during element imports. Hierarchical Excel-sheets describe e.g. application portfolio hierarchies in a human readable way. An example of a hierarchical Excel-sheet is given below:

	A	B
1	**Parent Process**	**Child Process**
2	Process A	
3		Process A.1
4		Process A.2
5		Process A.3
6	Process B	
7		Process B.1
8		Process B.2
9		Process B.3

A hierarchy specification defines a hierarchy between column identifiers declared in the column specifications block. Syntax for hierarchy specifications is given in the table below:

Hierarchy specification: **hierarchycolumns {** *ColumnName (/ ColumnName)** **}**
ColumnName: ID

Example query using hierarchy specifications block:

```
report ProcessHierarchyImport {
 ImportOnly;
  columns {
    "Parent Process" : cpname, "Child Process" : spname
  }
 hierarchycolumns { cpname / spname }
  queries {
   [Core Process]{ name => cpname}/subelement/[Subprocess]{name
=> spname};
   }
}
```

5.5 Query specifications block

Query specifications block contains one or more model queries. Each model query defines a path of one or more components connected by links. Components in a query are either model elements (e.g. process steps or conforming to user-defined element types) or connectors, while links correspond to relational attributes.

Query specifications are contained in a block denoted by **queries** keyword, as described in Table 5.4 Queries specifications. Each query specification consists of an optional scope declaration, and a query specification. When multiple queries are specified within a *queries* block the result will hold a set union of the corresponding queries.

An element filter declaration followed by an optional attribute specification part. The scope declaration can be either **$diagram**, **$active** or **$ID**.

Query specifications: **queries {** *QuerySpec+* **}**
QuerySpec: (Scope)? PathQuery (WithClause)? (MappingUsage)?
Scope: ***$(*****diagram** | **active** | ID)
PathQuery: *ElementSpec* (*/ LinkSpec / ElementSpec*)*
ElementSpec: **[** *ElementFilter* **]** (**{***AttributeSpecs***}**)?

ElementFilter: FilterSpec (**,** *FilterSpec*)*
FilterSpec: PropertyName (ComparisonOp Value)?
ComparisonOp: **=** | < | > | !=
Value: Quoted string or integer value.
AttributeSpecs: AttributeSpec (**,** *AttributeSpec*)*
AttributeSpec: PropertyPath (PropertyMapping)?
*PropertyPath: PropertyName (/ PropertyName)**
PropertyMapping: **=>** ID | **<=** ID
PropertyName: AttributeName | **diagram** | **parentdiagram** | **outgoingflows** | **incomingflows**
AttributeName: Unquoted string (may contain spaces).
LinkSpec: PropertyName | *RelConnectorResolution* | **subelement**
RelConnectorResolution: **#** *RelAttrName* (**<** | **>**) *ConnectorName*
RelAttrName: PropertyName
ConnectorName: PropertyName
WithClause: **with {** *ElementFilter* **}**
MappingUsage: **using** *ValueMappingName* **on {** *PropertyNameList* **}**
*PropertyNameList: PropertyName (, PropertyName)**

Table 5.4 Queries specifications

Diagram-specific scope denoted by **$diagram** means that model structure export is constraint to the diagram currently selected by the user. Scope denoted by **$active** means that the export is constrained to the specific model element currently selected by the user. Finally, if the scope is of form **$**ID, where ID is an identifier (an unquoted string with no whitespaces), the export is limited to paths that begin from the model element identified by symbol specified in the ID part of the scope element.

A path query contains at least one element specification. Typically the first element specification is followed by a link specification and another element specification. The element and link specifications are separated by slashes (**/**). A path query effectively defines a path between model elements where each model element is related to the next one through a relational attribute (the link specification).

An element specification comprises an element filter and an optional set of attribute specifications. During export an element filter is used for selecting appropriate model element from the active scope. If model element conforms to the filter, it is selected to the set of filtered elements.

If a link specification is defined, a new scope is created from the set of filtered elements: for each filtered element, if a relational attribute is found as defined by the link specification, the target element of the relational attribute is put to a new model scope. Such an alternation between filtering and linking of elements in scopes and filtered sets happens iteratively, until a path query is completely traversed.

There are a few special link names that are used for traversing e.g. diagram hierarchies while creating new model elements. These special link names are:

- **subelement** Declares that the element after the link is a sub-element of the previous element. In this case, the previous element MUST BE a diagram-type element. Applicable only when importing elements from Excel workbook to QPR PD/EA.

Result of a path query contains all complete paths that begin from the initial scope and have all the necessary links and intermediate elements to reach the end of the query. No incomplete paths are included in the result. If the model doesn't contain any paths conforming to the element and link specifications, the result set will be empty.

An element filter contains one or more filter specifications. A filter specification declares either a filter based on the type of the element or property value of the element. If a type restriction is used, the filter specification is simply a *PropertyName* that refers to an element type name defined in the model template, e.g. *Business Process*. In the case of property value comparison, the filter specification is for example *Business Value > 200000*; in this example case, only elements are selected whose *Business Value* attribute has an integer value that is more than 200000. Comparison is supported between integers, strings and dates with comparison operators '=' (equal), '<' (less than), '>' (greater than) and '!=' (not equal).

Individual filters in a filter specification are interpreted as conjunctive, i.e. an element must conform to all element filters to be included in the result. For example, a filter specification of *Resource type, name != "CEO", name != "CFO"* matches elements that are of type "Resource type" and do not have a value "CEO" or "CFO" in the *name* attribute.

An attribute specification defines a mapping between model element attributes and columns defined in the columns specifications, or alternatively, a value coercion. Each attribute specification defines a mapping between a property path and column identifier.

In the simple case, the path corresponds to an immediate attribute name. For example, an attribute specification *Name => cpname* maps the value of a model element property named *Name* to a column identified by *cpname* in column specifications.

Property names can be names of any properties in model elements that are accessible by the QPR API *GetProperty* method (see QPR *Developer's Guide*). In addition, the scripts provide a few hardcoded property names to retrieve often required information associated with model elements. These hardcoded property names are:

- **diagram**: fetches the diagram image of the element identified by the preceding property path to a separate Excel sheet;
- **parentdiagram**: returns the parent diagram element of the element identified by the preceding property path;
- **outgoingflows**: returns the set of outgoing connectors associated by the current element; and
- **incomingflows**: returns the set of incoming connectors associated by the current element.

Property paths can be used to fetch property values not residing immediately at the current element. For example, **parentdiagram/diagram** fetches the diagram image associated with the parent diagram of the current element; this is quite typical use case for exporting diagram images associated with individual model elements. Each diagram is exported only once in the resulting Excel workbooks during exports.

Value coercion is specified using *AttributeName <= STRING* syntax where *AttributeName* is a name of a model element (e.g. *namei)* and *STRING* is a quoted string. Value coercion sets a value during Excel-import for all elements. For example, to set description of all imported elements to a predefined value use the following syntax: *description <= "An imported element. Set description before approval."*

Link specifications correspond to relational attributes linking model elements the active model. Firstly, a link specification can define a name of a relational attribute. For example path query *[Subprocess]{name => cpname}/owner/[Resource type]{name => cpowner}* defines a path

between *Subprocess* and *Resource type* through a relational attribute named *owner* and contained in sub-process elements.

Secondly, a link specification can be used for definition of a relational connector resolution. A relational connector resolution returns the connector associated with the relational attribute, instead of returning the target element of the relational attribute. A relational connector resolution begins with a hash (**#**), followed by the name of the relational attribute (*RelAttrName*), a connector direction mark (**<** for incoming and **>**, and a connector name (*ConnectorName*). For example, a relational connector resolution **#Relation Ends.Aggregation From>Aggregation** returns the set of outgoing *Aggregation* connectors that have set values in *from* relational attribute of the current element.

Query specifications can be applied to specific elements using filtering declared by *with*-specifications. A *with*-specification declares a condition in form of a filter specification discussed above.

Finally, value conversions can be attained during import and export by value mapping usage declarations. A value mapping usage declarations specifies which value mapping is used in conjunction with which columns (denoted by column identifiers).

Section 7 includes example queries that apply all the features discussed above.

6 Configuration of Excel integrations

Integration between QPR ProcessDesigner/EnterpriseArchitect is configured using an Excel workbook that contains report specifications as declared in Chapter 5. The Excel workbook containing the report configurations is a macro-enabled workbook that contains:

- A sheet named "QPR Configuration" that holds the report specifications; and
- One or more sheets that contain data according to the report specifications (in case of import from Excel)

The sheet containing the report specifications contains one column with the first column acting as a header. The report specifications are declared beginning from the second row of the first column. Each row contains an individual report specification and each report specification represents an Excel sheet. The name of a report corresponds to an Excel sheet name. An example workbook view is illustrated in Figure 2: An example workbook containing report specifications.

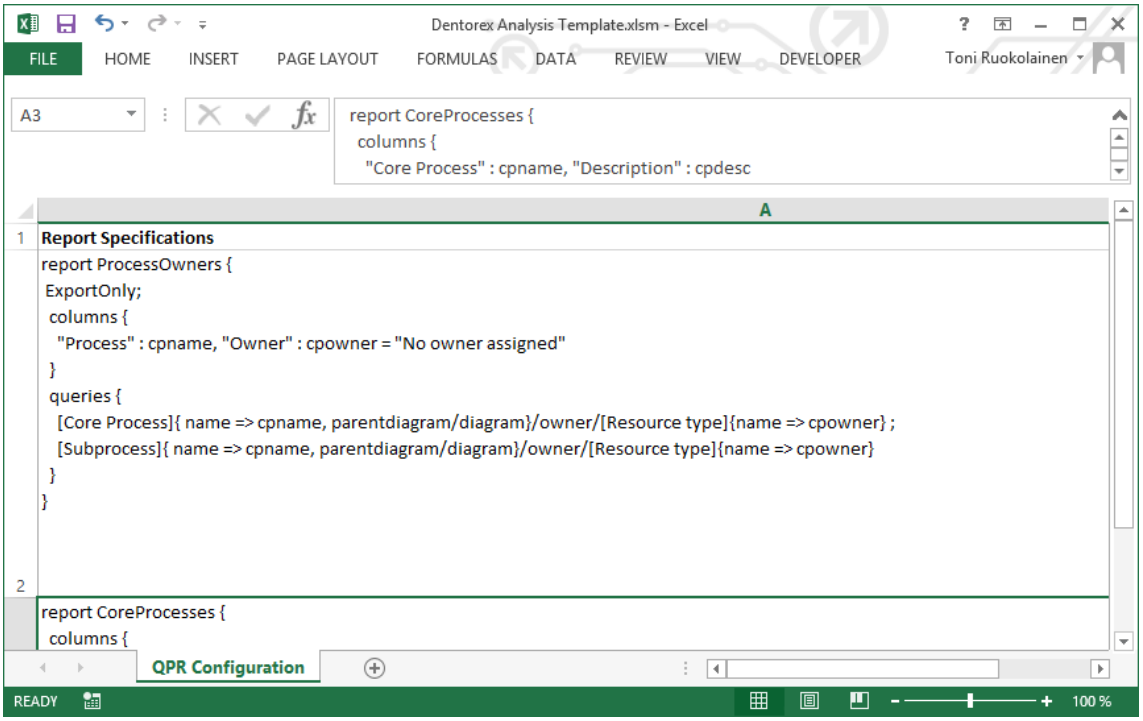


Figure 2: An example workbook containing report specifications.

6.1 Using export pre- and post-processing macros

During Excel export, the *ExcelIntegrationExport.qprpsc* script checks if specifically named macros exists in the Excel workbook containing the report configurations. If found, a macro function named *PreProcessing* (with one parameter) is executed before the export operation, and a macro function named *PostProcessing* (with one parameter) is called after the export operation. The parameters are not configurable nor set automatically by the export and import scripts. Parameter values can be set dynamically by solution provided via customization of the scripts.

Using the pre- and post-processing macros the solution provider may include complex processing to the export process. For example, data can be first fetched from 3rd party systems using the pre-processing macro function, then data is imported from the QPR PD/EA, and these separate data sets are conjoined in the post-processing macro function. Especially, data analysis and charting can be enabled using the post-processing macro functions. For example, pivot charts could be created by the macro functions based e.g. on application component costs and other attributes. An example of post-processing macro function in Microsoft Visual Basic for Applications is illustrated in Figure 3: Developing a post-processing macro function in VBA.

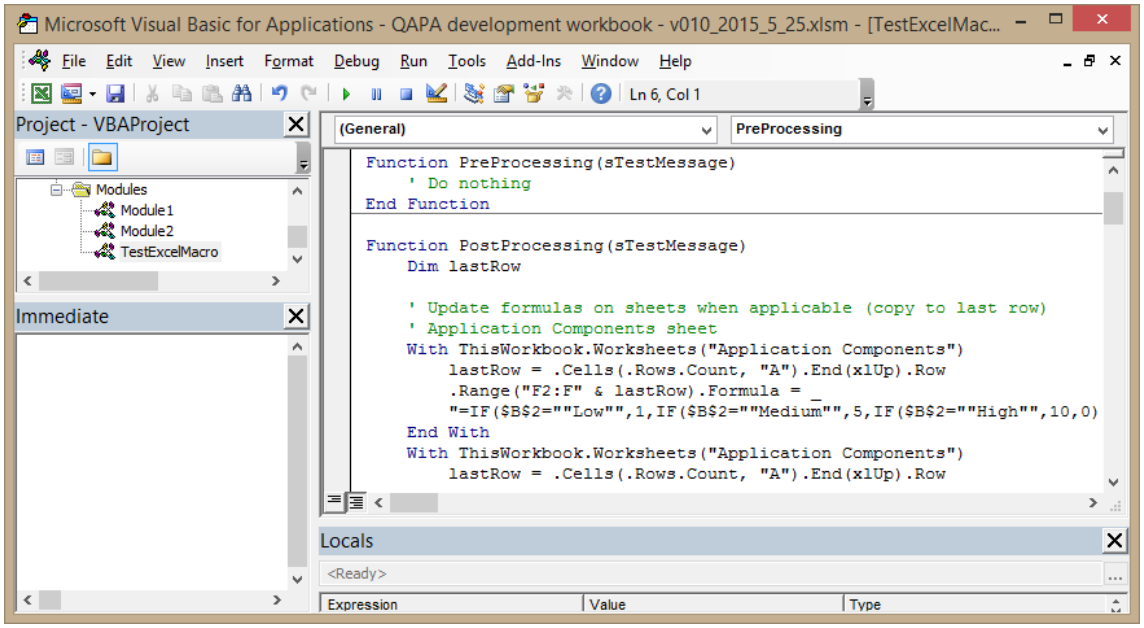


Figure 3: Developing a post-processing macro function in VBA.

7 Example Excel Integration template usage

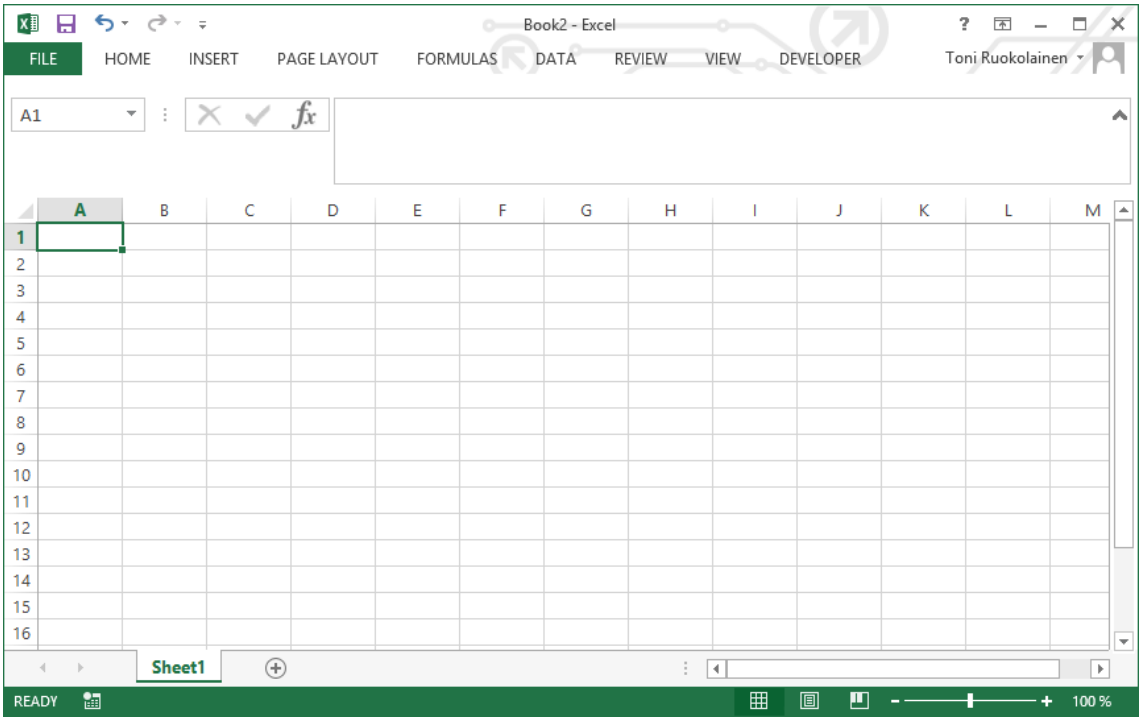
Application of Excel integration using the *ExcelIntegrationExport.qprpsc* and *ExcelIntegrationImport.qprpsc* scripts are defined in the following. The scenario is based on the Dentorex model delivered with the QPR ProcessDesigner and QPR EnterpriseArchitect clients. The scenario provides step-by-step instructions for typical round-trip integration between QPR PD/EA and Microsoft Excel.

7.1 Creating the report specifications Excel workbook

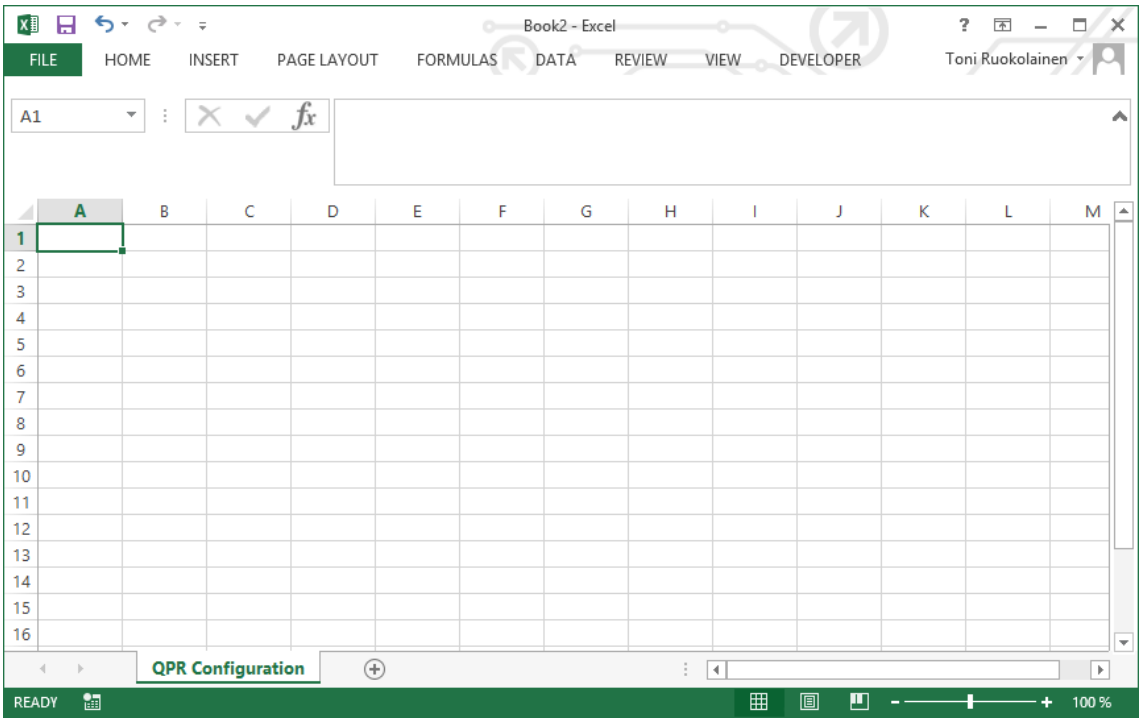
Steps for creating the report specifications Excel workbook:

1) Rename the default sheet to QPR Configuration
2) Open Microsoft Excel and create a new blank workbook

7.1.1 Open Microsoft Excel and create a new blank workbook



7.1.2 Rename the default sheet to QPR Configuration



7.1.3 Fill in the report specifications

At column A1 type "Report Specifications"

In the next rows in column A type in the report specifications from the following tables. The order of the reports does not matter, although it denotes the order of the Excel sheets to be created during exports.

```
report AllProcesses {
 ExportOnly;
  columns {
    "Process" : cpname, "Owner" : owner, "Parent diagram" : pdname
  }
  queries {
   [Subprocess]{ name => cpname, owner => owner, parentdiagram/name => pdname};
   [Core Process]{ name => cpname, owner => owner, parentdiagram/name => pdname};
   }
}
```

Table 7.1 AllProcesses query

AllProcesses query fetches all core and sub-processes to an Excel sheet containing process, owner and parent diagram names in three columns.

```
report AllControlStatuses {
 valuemapping StatusConversion = { "1" : "Passed",  "0" : "Failed" }
  columns {
    "Control Name" : ctrlname, "Control status" : ctrlstatus
  }
  queries {
   [Control]{ name => ctrlname, control test outcome => ctrlstatus} using StatusConversion
on { ctrlstatus };
   }
}
```

Table 7.2 AllControlStatuses query

AllControlStatuses query exchanges control representations between an Excel sheet named ”AllControlStatuses” and PD/EA model. Values of “1” and “0” are expected in “Control status” column of the Excel sheet; these values are mapped to “Passed” and “Failed” values in model elements of type Control, correspondingly.

```
report ImportOnlyPassedControls {
 ImportOnly; ExcelSheetName = "ControlStatuses"
 valuemapping StatusConversion = { "1" : "Passed",  "0" : "Failed" }
  columns {
    "Control name" : ctrlname, "Control status" : ctrlstatus
  }
  queries {
   [Control]{ name => ctrlname, control test outcome => ctrlstatus}  with { ctrlstatus = 1 }
using StatusConversion on { ctrlstatus };
   }
}
```

Table 7.3 ImportOnlyPassedControls query

ImportOnlyPassedControls query imports only those controls from ControlStatuses Excel-sheet that have ”1” in their ”Control status” columns.

```
report ProcessHierarchyImport {
 ImportOnly;
  columns {
    "Parent Process" : cpname, "Child Process" : spname
  }
 hierarchycolumns { cpname / spname }
  queries {
   [Core Process]{ name => cpname}/subelement/[Subprocess]{name => spname};
   }
}
```

Table 7.4 ProcessHierarchyImport query

ProcessHierarchyImport query will import all processes described hierarchically in Excel-sheet named "ProcessHierarchyImport". Corresponding process hierarchies will be created in the target model with Core Process elements representing parent processes and Subprocess elements representing child processes.

```
report CEOProcesses {
 ExportOnly;
  columns {
    "Process" : cpname
  }
  queries {
   [Subprocess]{ name => cpname}/owner/[Resource type, name = "CEO"];
   [Core Process]{ name => cpname}/owner/[Resource type, name = "CEO"]
   }
}
```

Table 7.5 CEOProcesses Query

CEOProcesses query will contain all core and sub-process elements what are owned by CEO.

```
report CFOProcesses {
 ExportOnly;
  columns {
    "Process" : cpname
  }
  queries {
   [Subprocess]{ name => cpname}/owner/[Resource type, name = "CFO"] ;
   [Core Process]{ name => cpname}/owner/[Resource type, name = "CFO"]
   }
}
```

Table 7.6 CFOProcesses query

CFOProcesses will contain all core and sub-process elements that are owned by CFO.

```
report ProcessOwners {
 ExportOnly;
  columns {
    "Process" : cpname, "Owner" : cpowner = "No owner assigned"
  }
```

```
  queries {
    [Subprocess]{ name => cpname, parentdiagram/diagram}/owner/[Resource type, name !=
"CEO", name != "CFO"]{name => cpowner} ;
    [Core Process]{ name => cpname, parentdiagram/diagram}/owner/[Resource type, name
!= "CEO", name != "CFO"]{name => cpowner}
   }
}
```

Table 7.7 ProcessOwners query

ProcessOwners query will contain all core and sub-process elements that are not owned by CEO or CFO.

```
report CoreProcesses {
  columns {
    "Core Process" : cpname, "Description" : cpdesc
  }
  queries {
    [Core Process]{ name => cpname, Description => cpdesc}
  }
}
```

Table 7.8 CoreProcesses query

CoreProcesses query will contain all core process elements with descriptions.

```
report RisksForCoreProcesses {
  columns {
    "Core Process" : cpname, "Description" : cpdesc, "Risk for Core Process" : cprisk, "Risk
Category" : riskcat
  }
  queries {
    [Core Process]{ name => cpname, Description => cpdesc}/Risk for Core
Process/[Risk]{name => cprisk, Risk Category => riskcat }
  }
}
```

Table 7.9 RisksForCoreProcesses

RisksForCoreProcesses query will contain all core processes that have a risk associated with them. Process description and risk category is shown in the report.

```
report MultiQueryReport {
  columns {
    "Core Process" : cpname, "Description" : cpdesc
  }
  queries {
    [Core Process]{ name => cpname, Description => cpdesc} ;
    [Core Process]{ name => cpname, Description => cpdesc}
  }
}
```

Table 7.10 MultiQueryReport query

MultiQueryReport query will contain all core processes twice. This is just to show the functionality of import during multi-query reports. Any elements in multi-query reports are neglected during import.

```
report ControlsSinceSep {
 columns {
  "Control" : cname, "Description" : cdesc, "Date of last test" : cdate
 }
 queries {
  [Control, Control Test Date (Last) > "1.9.2014" ]{ name => cname, Description => cdesc,
Control Test Date (Last) => cdate}
 }
}
```

Table 7.11 ControlsSinceSep query

ControlsSinceSep report will contain control elements, their descriptions and dates of last test. Only controls that have been tested after 1st September 2014 are included in the report.

```
report ControlsBeforeSep {
 columns {
  "Control" : cname, "Description" : cdesc, "Date of last test" : cdate
 }
 queries {
  [Control, Control Test Date (Last) < "1.9.2014" ]{ name => cname, Description => cdesc,
Control Test Date (Last) => cdate}
 }
}
```

Table 7.12 ControlsBeforeSep query

ControlsBeforeSep query will contain control elements, their descriptions and dates of last test. Only controls that have been tested before 1st September 2014 are included in the report.

```
report ControlsAtFirstSep {
 columns {
  "Control" : cname, "Description" : cdesc, "Date of last test" : cdate
 }
 queries {
  [Control, Control Test Date (Last) = "1.9.2014" ]{ name => cname, Description => cdesc,
Control Test Date (Last) => cdate}
 }
}
```

Table 7.13 ControlsAtFirstSep query

The report corresponding to Table 7.9 will contain control elements, their descriptions and dates of last test. Only controls that have been tested exactly at 1st September 2014 are included in the report.

```
report ControlsNotAtFirstSep {
 columns {
```



```
  "Control" : cname, "Description" : cdesc, "Date of last test" : cdate
 }
 queries {
  [Control, Control Test Date (Last) != "1.9.2014" ]{ name => cname, Description => cdesc,
Control Test Date (Last) => cdate}
 }
}
```

Table 7.14 ControlsNotAtFirstSep query

ControlIsNotAtFirstSep query will contain control elements, their descriptions and dates of last test. Only controls that have been tested in date different from 1st September 2014 are included in the report.

Save the Excel resulting workbook (with xlsx file-ending). The resulting workbook should look something as illustrated in Figure 4: Example report after configuration.

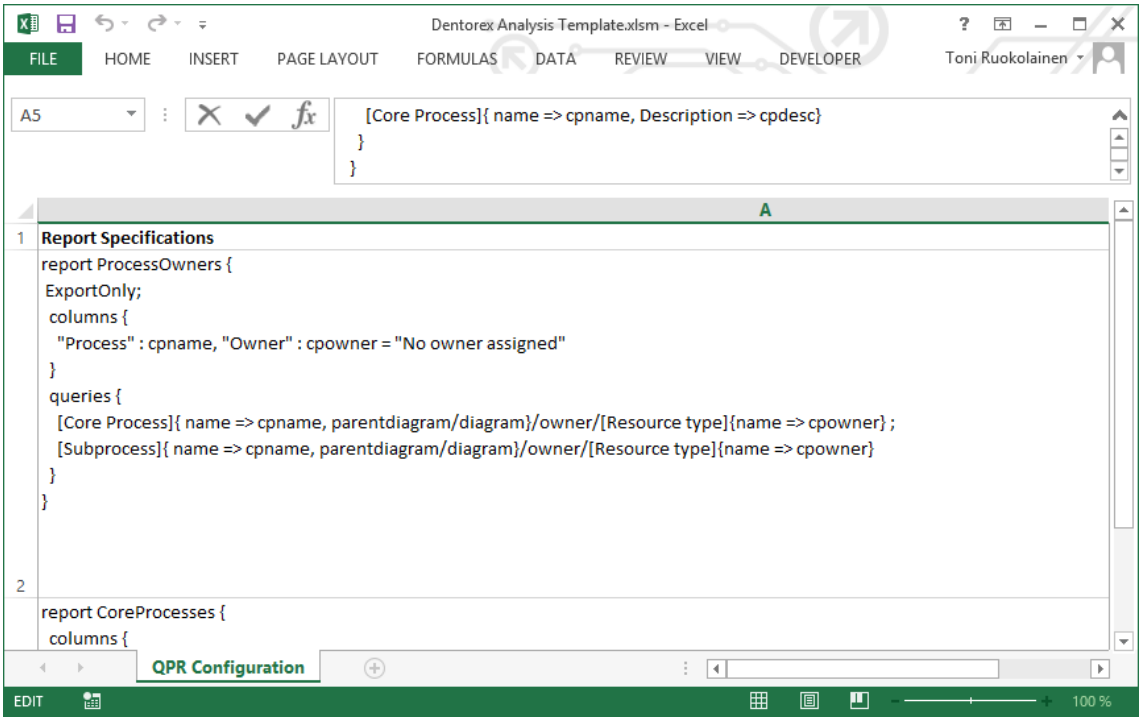
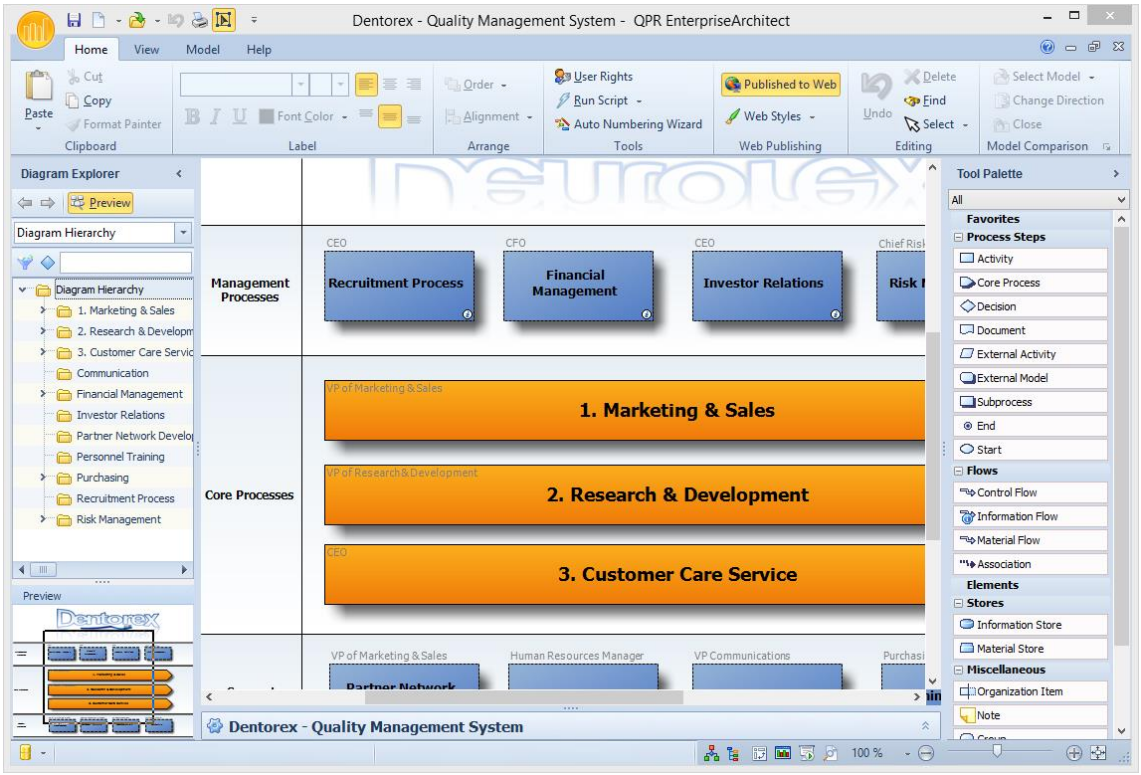


Figure 4: Example report after configuration.

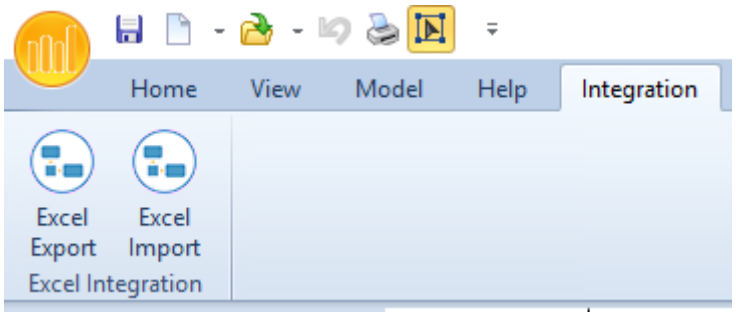
7.2 Exporting Dentorex model contents

7.2.1 Open the Dentorex demo model

First open QPR ProcessDesigner or QPR EnterpriseArchitect, and then open the Dentorex demo model.

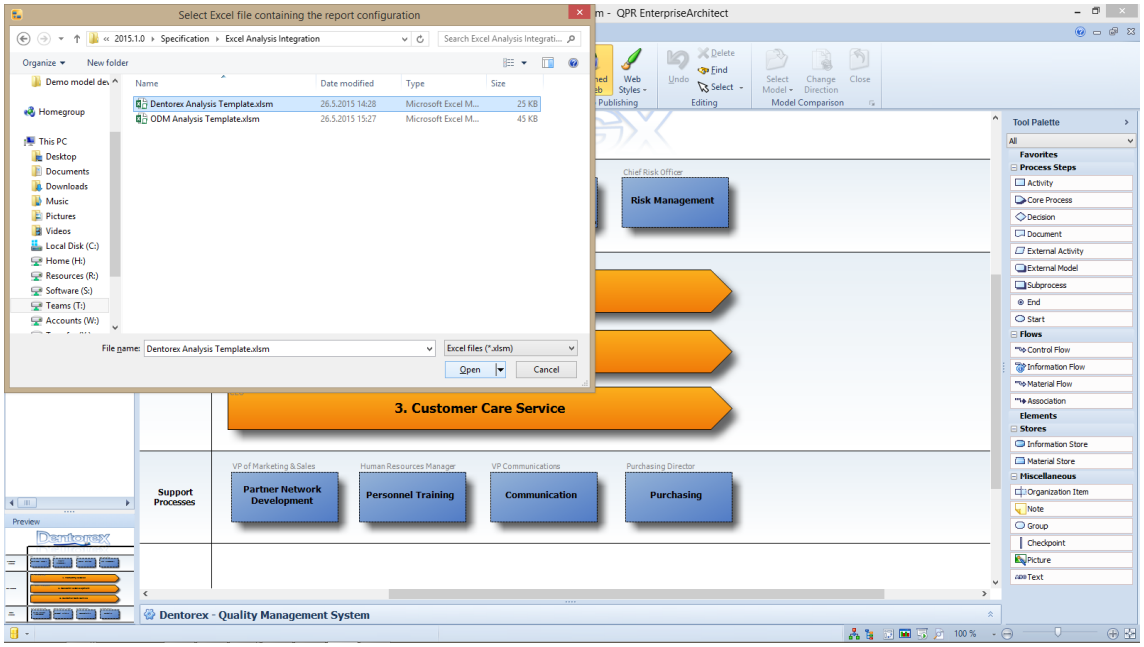


7.2.2 Run the *Excel Export* integration procedure

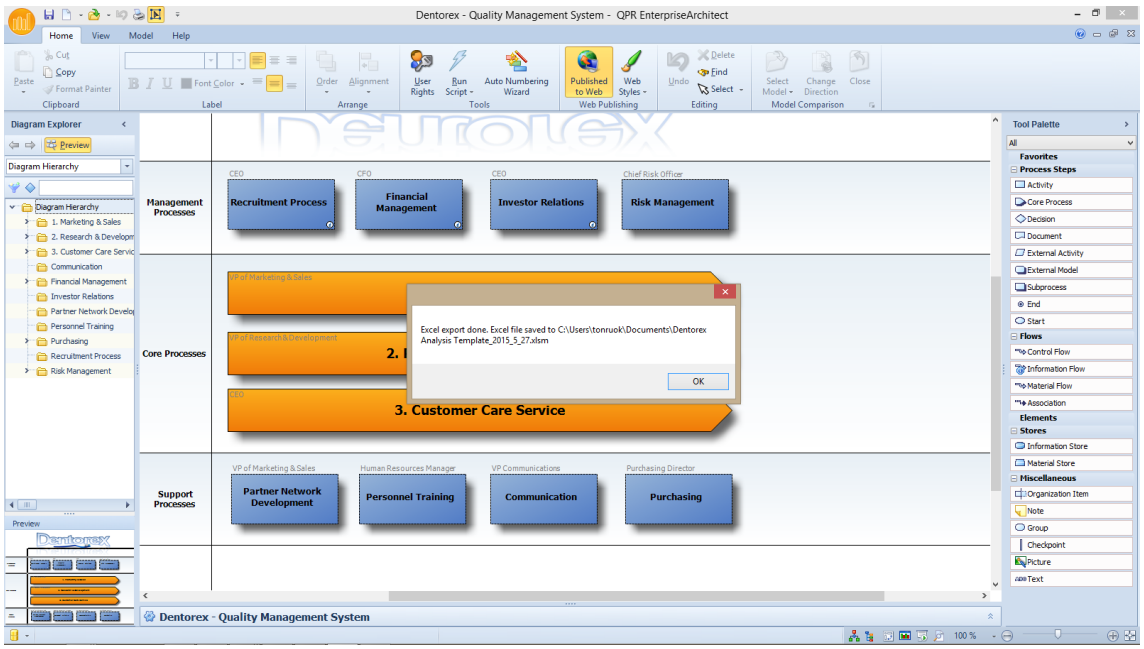


The button should be visible in the "Integration" tab after successful installation.

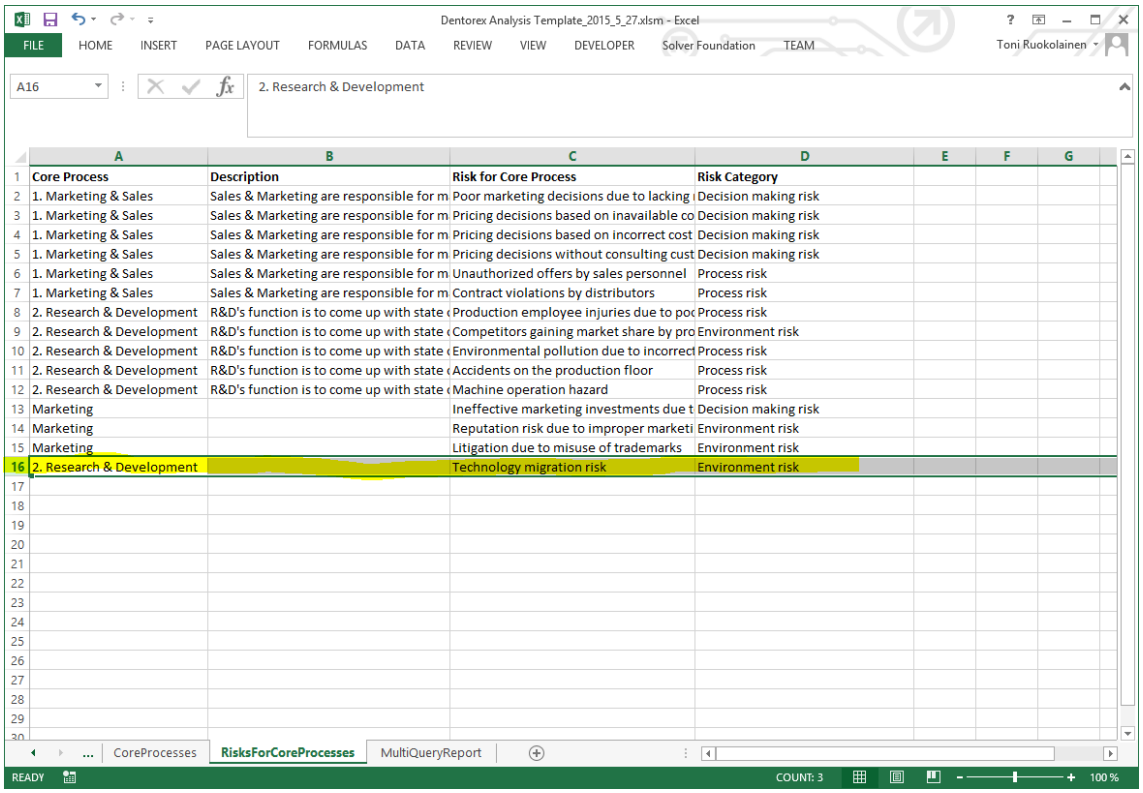
7.2.3 Select the Dentorex integration Excel workbook when asked



7.2.4 Export is ready



7.3 Use the generated Excel workbook for reporting, analysis or data collection



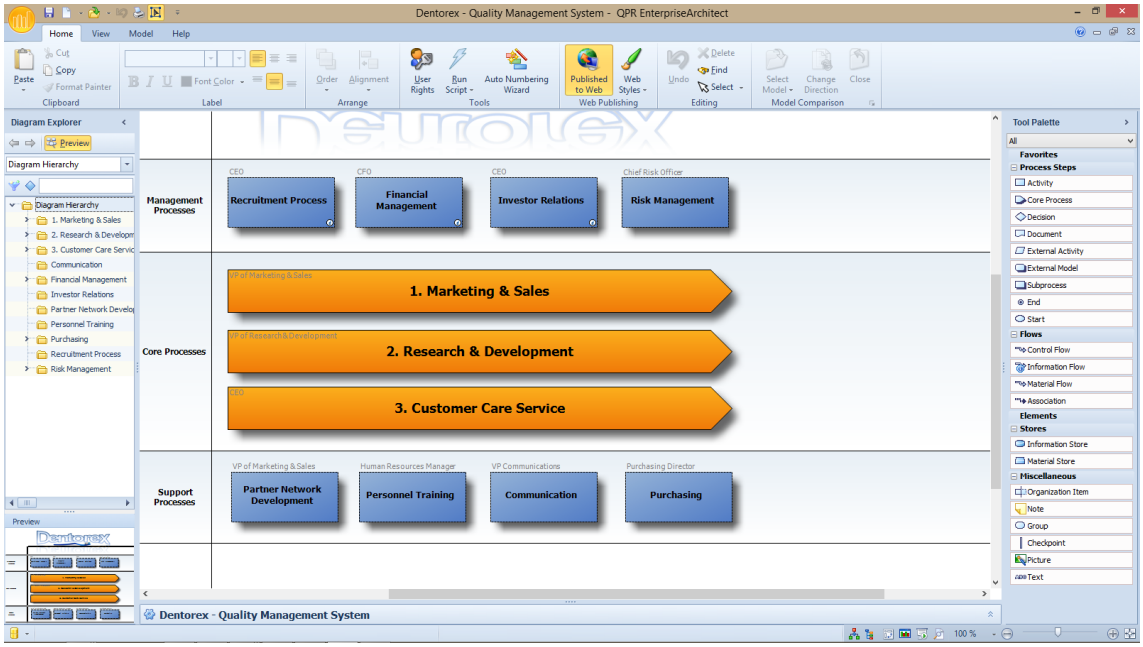
	A	B	C	D	E	F	G
1	Core Process	Description	Risk for Core Process	Risk Category			
2	1. Marketing & Sales	Sales & Marketing are responsible for m	Poor marketing decisions due to lacking	Decision making risk			
3	1. Marketing & Sales	Sales & Marketing are responsible for m	Pricing decisions based on inavailable co	Decision making risk			
4	1. Marketing & Sales	Sales & Marketing are responsible for m	Pricing decisions based on incorrect cost	Decision making risk			
5	1. Marketing & Sales	Sales & Marketing are responsible for m	Pricing decisions without consulting cust	Decision making risk			
6	1. Marketing & Sales	Sales & Marketing are responsible for m	Unauthorized offers by sales personnel	Process risk			
7	1. Marketing & Sales	Sales & Marketing are responsible for m	Contract violations by distributors	Process risk			
8	2. Research & Development	R&D's function is to come up with state	Production employee injuries due to poo	Process risk			
9	2. Research & Development	R&D's function is to come up with state	Competitors gaining market share by pro	Environment risk			
10	2. Research & Development	R&D's function is to come up with state	Environmental pollution due to incorrect	Process risk			
11	2. Research & Development	R&D's function is to come up with state	Accidents on the production floor	Process risk			
12	2. Research & Development	R&D's function is to come up with state	Machine operation hazard	Process risk			
13	Marketing		Ineffective marketing investments due t	Decision making risk			
14	Marketing		Reputation risk due to improper marketi	Environment risk			
15	Marketing		Litigation due to misuse of trademarks	Environment risk			
16	2. Research & Development		Technology migration risk	Environment risk			

In the above figure, a new R&D risk was identified and added to the Excel sheet. This risk is not yet included in the model.

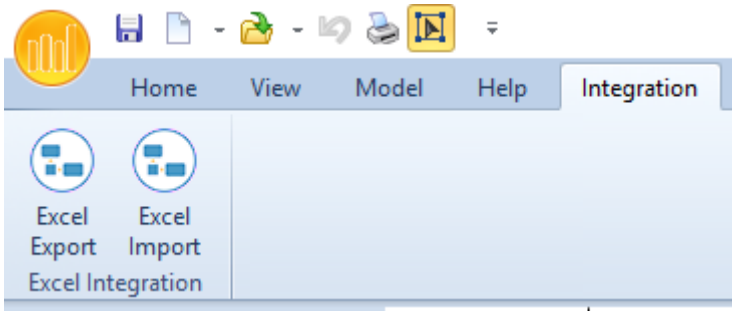
Export script has created an initial validation list for the Risk Category column. Such validation lists are used in case of enumerated attributes. It should be noted that the validation list does not contain all the enumerated values contained in the corresponding PD/EA enumeration, only the ones that have been encountered during the export. In the example case, the validation list contains accepted values of "Decision making risk", "Process risk", and "Environment risk". Values can be set e.g. by the post-processing macro functions, or manually in the Excel sheet.

7.4 Import updates from the Excel workbook to the model

7.4.1 Open the Dentorex model

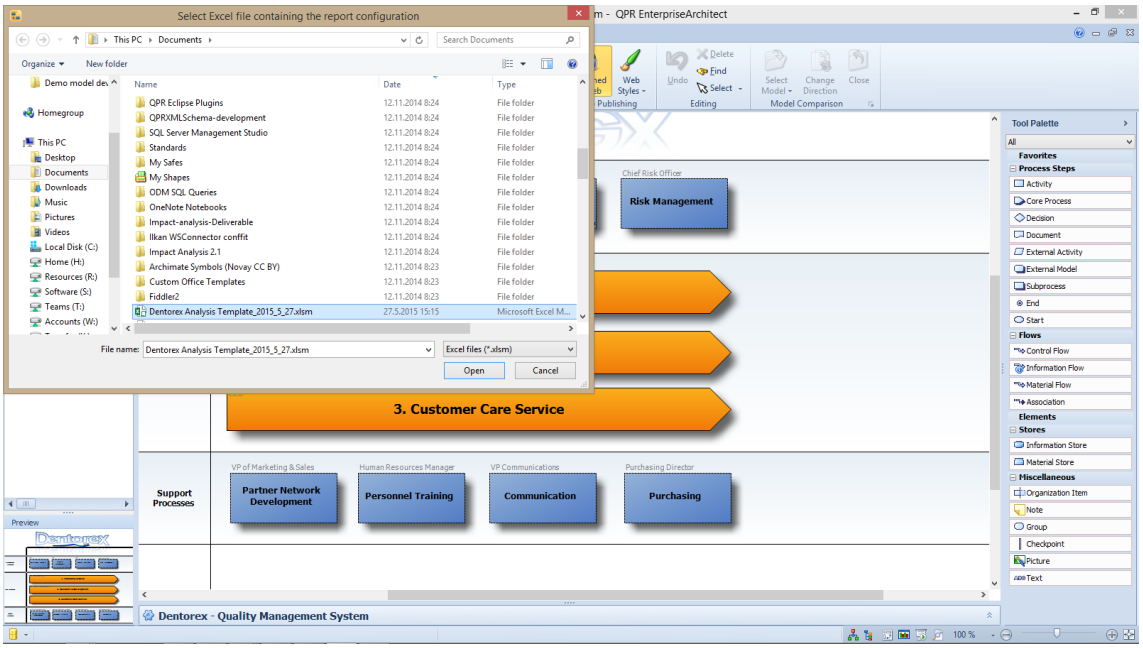


7.4.2 Run the *ExcelIntegrationImport.qprpsc* script



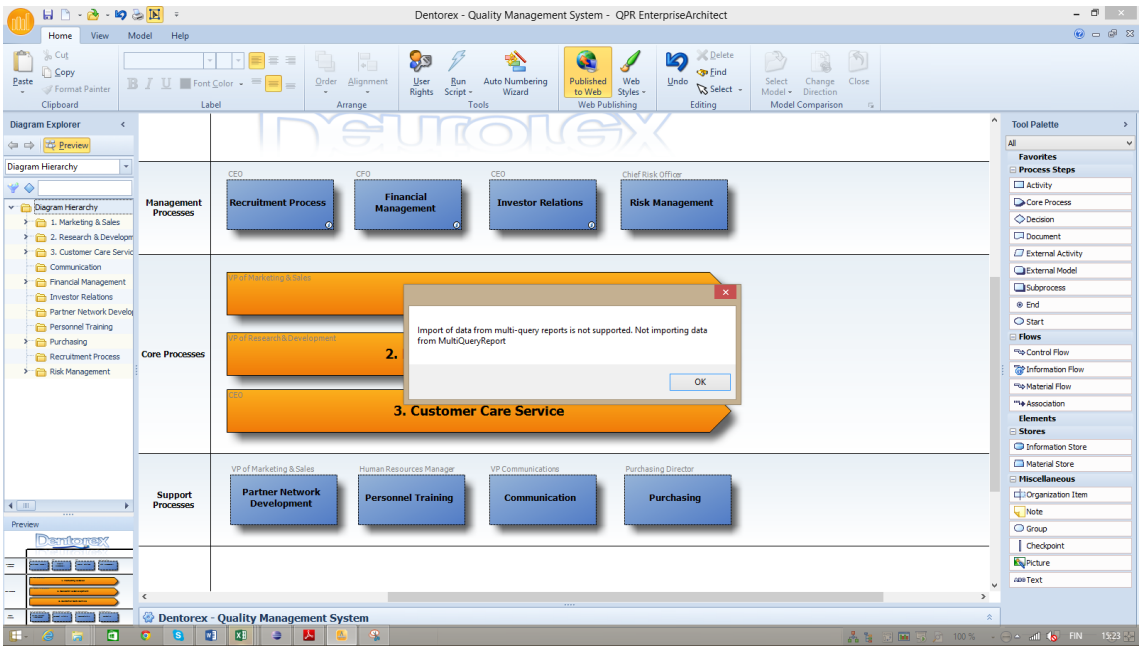
The button should be visible in the "Integration" tab after successful installation.

7.4.3 Select the Dentorex Analysis Template Excel workbook that was generated and updated previously



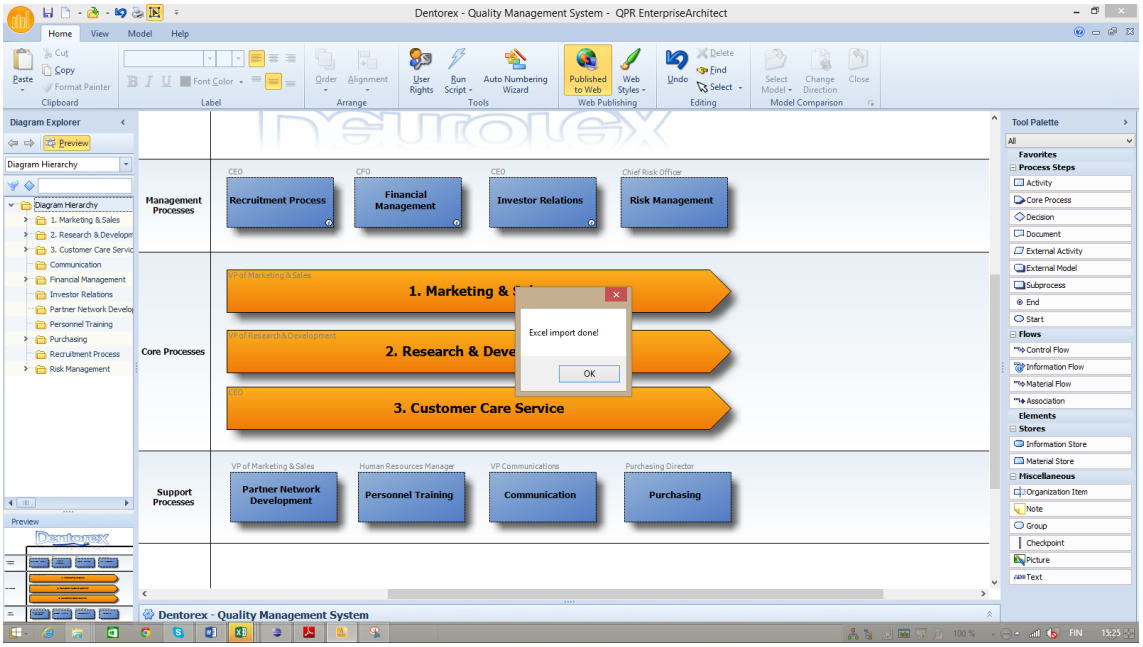
7.4.4 The import script says that multi-query imports are not supported.

Imports are only supported for reports that contain a single query. Multi-query imports might have ambiguous semantics.

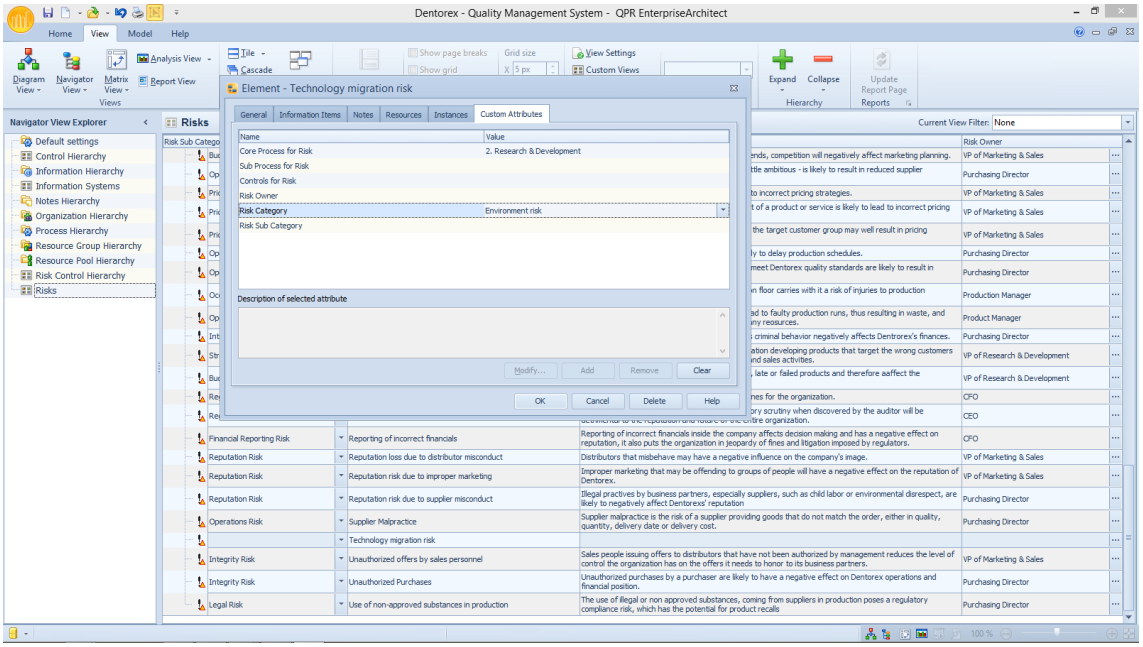


No data is imported from the sheet named *MultiQueryReport* – data from all other sheets are updated to the model.

7.4.5 The import is done



7.4.6 A new risk item was imported to the model and visible in the Risks navigator view



Risk category "Environment risk" was set to the relation relational attribute "Risk category" in the Risk element.

