
Dr. Einar W. Karlsen

March 04, 2014

The third article in a series that discusses best practices for generating compliance-relevant reports. This article shows how to use Rational Publishing Engine to generate archetypical design documents by extracting model information from IBM Rational Software Architect Design Manager.

View more content in this series

More articles on document generation

For other articles on this topic, please see the four-part series by Einar Karlsen on "Generating compliance documentation using Rational Publishing Engine"

Topics include:

- Overview and workflow (part 1)
- Document design and layout (part 2)
- Reporting over multiple data sources (part 3)
- Testing, review, and deployment (part 4)

This article demonstrates how to generate archetypical design documentation using IBM® Rational® Publishing Engine by extracting information from IBM® Rational® Software Architect Design Manager. The techniques introduced go beyond generating design documentation — they apply equally well when you need to generate documentation for business processes, use cases, or implementation models.

Basic requirements

The requirements for model-based documentation are much the same regardless of the client. These requirements typically include:

- Section titles are extracted from Unified Modeling Language (UML) package names.
- Section numbers reflect the hierarchical structure of the model (that is, package depth).
- Figures are extracted from diagrams found in a package or embodied in UML elements. The name of the diagram is used as title for the figure.
- Text is extracted from the associated description fields of the UML elements or the diagrams.
Detailed information about the UML elements is extracted either in textual form as a sequence of text with headings (used for child packages, for example) or as tables (often used for UML elements such as classes, attributes, and operations).

A sample document reflecting these requirements is shown in Figure 1.

**Figure 1. Rational Software Architect Design Manager reports**

The documentation might have additional requirements. Some users might ask that empty tables be replaced with a line of text such as "There are no operations for this class." Parameterized templates frequently are needed so that the level of detail can be controlled by parameters. There may be a need for cross-domain reporting, so that all requirements for a given UML element must be extracted, as well. This in turn requires that the elements are linked to the requirements (for example, by using Open Services for Lifecycle Collaboration [OSLC] links) and that the information can be extracted from the relevant requirement management tool (such as IBM® Rational® Requirements Composer or IBM® Rational® DOORS®) following these links.

This article demonstrates step by step how to build the template that generates the design document shown in Figure 1. Ideally, the development of model guidelines and reporting templates go hand in hand, so when you set up model guidelines, keep in mind the overall reporting requirements. If you don't, you may find it more difficult later on to generate the required reports because the information isn't available in the model, isn't easily retrievable, or the team members didn't impose standards.

**The model structure**

Assume a model structure where the top level has no elements. The top-level package in the model contains overview diagrams and packages containing the design information. These child packages may be structured hierarchically in up to five levels of nesting.
As a sample model for the templates, I'll use the UML Profile-based Integrated Architecture (UPIA) model that comes with Rational Software Architect. The model has been imported into a project and workspace both named **UPIAEnterprise**. Basically, the model contains views of the enterprise architecture such as business architecture, enterprise strategy, and IT architecture. The IT architecture view is structured further into subpackages, which may contain diagrams, classes, and interfaces. See Figure 2.

**Figure 2. Rational Software Architect Design Manager model**

REST schema and resource discovery

The Rational Software Architect Design Manager offers reporting through a Representational State Transfer (REST) service offering URLs. There are entry points to retrieve schemas and data sources for all types of UML models, including packages, classes, operations, properties, and so on. Additional URLs provide access to stereotypes and tagged values, collaboration links to external tools, reviews, and comments. You can also use field parameters to limit the scope of data being returned — that is, to retrieve all classes referenced (contained) by a single package or all operations referenced by a specific class.

Discovering schemas

Discovering the infrastructure of schemas and data sources is best done using the REST resource discovery wizard of the Rational Publishing Engine. The command **data>Schema Discovery>REST Schema Discovery** needs to be invoked first to retrieve the appropriate schema for template authoring. The root URL for the REST services is **https: //<servername>:<port>/dm/reporting** and the REST services need to be accessed using the **OAuth** authentication mechanism.
Starting from the root, drill down to the required schema using the wizard, which may need several steps. For example, retrieving the schema for a UML model requires that you select the project and workspace first, then the adapter for UML, and finally the UML type adapter, before you can select the schema for UML models (see Figure 3).

Figure 3. Schema discovery

Configuring the data source

You can also use the REST resource discovery wizard to configure the data source for the model that is in scope for reporting. The data sources and schemas can also be discovered using a browser, but using the REST resource discovery wizard is better. You can observe this from the structure of the URL in Figure 3: it is nontrivial, and goes beyond the base URL of parameters that define the context such as the project and the workspace where the model is managed. With Rational Publishing Engine 1.2.1, this wizard has been improved to better support (and remember) the context selection that Design Manager needs.

Basic template structure

The schema for models provides generic information for the UML model elements such as name, type, applied stereotypes, and description, amongst others (see left side of Figure 4). The schema for models also provides queries for extracting information related to the model itself (such as
name and description), as well as queries for extracting information related to nested packages, diagrams, and UML elements. See Figure 4:

Figure 4. Schemas for UML models and UML interfaces

The schema for models does not, however, provide information specific to UML element types, such as interface operations or class attributes. If this information is needed for a report (and it usually is), you’ll have to connect to the data source for the given UML element using a dynamic data source connection and a UML element–specific schema such as the schema for interfaces (shown in the right half of Figure 4).

Template authoring can start after the schemas have been loaded into Document Studio. Begin with the part of the template that prints out the packages contained directly on the top level of the model, as well as the diagrams within those packages. These require the queries in the schema UML Model.

The initial template extracts the model description first ($63). A container has been defined to extract all nested packages (query $1) of the model, including the package name (as heading) and package description. The diagrams (query $4) are extracted next with the UML diagram image and description. Notice that the query named Description is used to extract rich text descriptions from model elements, whereas the query named description is used for the similar purpose for diagrams. See Figure 5.
**Figure 5. Package details and diagrams**

![Figure 5: Package details and diagrams](image)

**Reusing templates**

Always reuse existing templates as much as possible to save work. Before Rational Publishing Engine 1.2, reuse was supported in terms of *snippets* - small template fragments that could be registered and inserted directly using the palette. With Rational Publishing Engine 1.2, this has changed. You can now import template fragments directly, either by physically embedding the templates or by dynamically referencing a template.

Embedding is good if the template needs to be modified in the current context; dynamic referencing is appropriate if the reused template needs to be maintained centrally for all referencing templates. The reusable templates can be stored directly on the file system or in a central library on a web server.

**Importing a referenced template**

In the template shown in Figure 5, the master page (top element) has been imported as a referenced template. This choice has the advantage that company layout standards for headers and footers can be maintained centrally and reused (and modified) in all templates, even existing ones.

Following the container for rendering diagrams, define a table to extract information regarding the UML classifiers of the package. You can do this by iterating over all elements (query $78$) filtering those that are classifiers (see Figure 6).
The filter is defined as a condition with the following definition:

```java
superTypes.indexOf("uml:Classifier") != -1 && concreteType != "uml:Association"
```

The next step in defining the template is to turn it into a template that extracts information (recursively) about the packages. At the same time, it lets the package hierarchy be reflected by proper numbering of the section titles. See Figure 7.

**Figure 7. Recursion and section numbering**

To achieve recursion, first set two properties of the query ($1$) that iterates over the nested packages (see left half of Figure 7):

- **Recursive Level**, which defines the depth of the recursion (set to 5)
- **Recursive Segments**, which defines the segments of the query to be repeated for each recursion (2, which denotes the last two query segments, `nestedPackage/Package`)

Next, define the style of the section title that renders the name of the package. To do this, set two properties (see the right half of Figure 7):

- **Heading Offset Level**, which has been set to 1 insofar that style H1 is used for the title of the chapter. Consequently, style H2 is used for the topmost section rendering package details, H3 for the next level, and so on.
- **Style Name**, which in this case has been defined using the predefined variable `_element_level`, which defines the current level of recursion.
Interface details

Now extend the template so that it generates a report containing information about the interfaces in a package and the provided operations (if any) for each interface. The operations are rendered in a table showing operation name, description, and visibility (real life reports typically also show the signature and stereotypes of the operation, amongst other characteristics). See Figure 8 for the elements and queries that achieve this effect.

Figure 8. Interface details

The template fragment starts by iterating over all packaged elements of a nested package ($18). It then filters the elements that are interfaces (concreteType == 'uml:Interface') – but only if the corresponding document specification has been configured to enable interface details (PrintClassOperations== "true"). To retrieve information about the interfaces such as the associated properties and operations, you need another schema, DM Interfaces. This schema can be imported into the template using the resource discovery wizard (see the right part of Figure 4). After you import the schema, you must use a dynamic data source connection to connect to the data source for UML interfaces and retrieve the information associated with the interface. The following properties must be defined for the data source:

- The schema (DM Interface)
- The URL of the resource (defined using the href for the UML element)
- The user name and password (inherited from the data source DM Model)

Now you can use the queries of the schema DM Interface to iterate over all operations of the interface (table row, query $37) and retrieve operation details such as name, description, and visibility (table cells). The table was inserted into the template from a library of reusable components.

Omitting empty tables and figures

The disadvantage of the basic template is that it prints out empty tables with just the header row and table caption when an interface has no operations. Rational Publishing Engine has no built-in set queries or properties that allow a template author to filter on whether an element set is empty or not.
One solution to this problem is to define a variable **Has Elements** that is initialized to **false**. Then iterate over all UML operations ($^\text{38}$), and if one is found, set **Has Elements** to **true**. You can then use **Has Elements** as a condition for the container that holds the table of operations to ensure that the table is included in the documentation only if there are elements to render. See Figure 9.

**Figure 9. Omitting empty tables**

You can use a similar technique to avoid printing out image and figure captions for diagrams that are empty; that is, for diagrams that don't contain any elements.

**Note:** Rational Publishing Engine 1.2.1 supports a new property for tables called "remove if empty" which deletes tables that are empty or have only header rows (rows with "row repeat at page beginning" set to "true"). However, this property works only for tables, whereas the template provides a general pattern for all kind of elements — tables, figures, subsections, and so on.

**Optimization**

The basic template is defined to iterate over all interfaces in a package and then dynamically connect to the data source that contains the information related to the interface, such as the interface operations. However, each interface requires one call to the REST service, which introduces runtime overhead when the document is generated.

You can optimize this behavior by requesting that the REST service return all interfaces referenced by the current package in one REST service call. To do this, you have to manipulate the URLs. What you have is the URL for the package that contains the interfaces. This URL defines (amongst other things) the type of the UML element (**Package**) and the unique identifier (parameter **id**):

https://win2k8server:9443/dm/reporting/uml/Package?name=Acquisition&id=<package_id>&project=....

The URL for retrieving all interfaces referenced by a specific package looks almost the same as the URL of the package itself:

https://win2k8server:9443/dm/reporting/uml/Interface?referencingId=<package_id>&project=UPIAEnterprise/workspace=UPIAEnterprise&rmps.context=...

These URLs have two main differences (highlighted in bold font): the type of the URL element needs to be changed from **Package** to **Interface** and the parameter named **id** needs to be
changed to referencingId. Listing 1 shows the definition of a Javascript that takes a URL for a package and returns a URL for retrieving all referenced interfaces of the package.

Listing 1. Javascript computing the URL for all interfaces referenced by a package

```javascript
// Change UML element type in base URL
index = href.indexOf("/reporting/");
baseURL = href.substring(0,index+11);
baseURL = baseURL + "uml/Interface"

// Change parameter name "id" to "referencingId"
index = href.indexOf("?");
if (index > 0) {
  params = href.substring(index);
  params = params.replace("?id=","?referencingId=");
  params = params.replace("&id=","&referencingId=");
  baseURL + params
} else {
  baseURL
}
```

The resulting template is shown in Figure 10. Unlike with the basic template, here you don't have to iterate over all packaged elements and then filter out those that are interfaces using a condition. Rather, the template fragment starts by assigning the value of the URL for retrieving all interfaces of the current package to a variable named ClassifierURL. The value of this variable is then used as the value of the URI parameter in the dynamic data source connection. The rest of the template remains unchanged. See Figure 10.

Figure 10. Optimization

Conclusion

This article discusses how to use Rational Publishing Engine to create design documents extracting information from Rational Software Architect Design Manager. In particular, techniques to define archetypical design documents, including documentation, diagrams, element details, and stereotypes following the hierarchical structure/nesting of the UML model are covered. Specific topics related to recursion, avoiding empty tables and diagrams, use of Javascripts, URL fields, and runtime optimization were introduced. For more details about design management reports, Kevin Cornell and Adim Delibas present a comprehensive tutorial on Jazz.net.
Related topics

- Learn more about Rational Publishing Engine features and benefits:
  - Check the overview, get installation help, and read the documentation in the information center.
  - Watch the demo to see how you can use Rational Publishing Engine, including how to configure templates.
- Read other articles in this series.
- Read "Rational Solution for Collaborative Lifecycle Management Traceability Templates with Rational Publishing Engine" (Geoff Rosenthal, Jazz.net, October 2012)
- For additional tips and tricks, visit Rational Publishing Engine Actual.
- Gather more information for the Reporting Area Web Publisher.
- To learn more about Rational Software Architect:
  - Start at the Rational Software Architect family of products page and be sure to check the developerWorks page.
  - Also, explore the product overview, the information center for installation and use instructions, and the Rational Software Architect wiki for more resources.
  - Check Jazz.net to find out more about Rational Software Architect Design Manager.
- For more help with templates and schemas:
  - For more tips, tutorials and sample templates, visit the Rational Publishing Engine Community wiki and check the Rational Publishing Engine Template library.
  - If part of what you want to include in your reports is in Microsoft Office documents, here's where you can find help: Go to either 2007 Office System: XML Schema Reference to download XSD schemas for Microsoft Office products. Navigate to Microsoft Office Project 2007 XML Data Interchange Schema to download the XSD schema required to report on Microsoft Project data. Read the Microsoft article titled Create an XML data file and XML schema file from worksheet data to learn more about how to create appropriate schemas for reporting Excel data.
  - Improve your skills. Check the Rational training and certification catalog, which includes many types of courses on a wide range of topics. You can take some of them anywhere, anytime, and many of the Getting Started ones are free.
  - Download any of these for a trial at no charge:
    - Rational Software Architect
    - Rational Software Architect for WebSphere Software
    - Rational Software Architect Design Manager

© Copyright IBM Corporation 2014
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)