Using IBM Rational Publishing Engine to generate compliance documents: Part 2. Report generation for Rational Quality Manager, showing how to extract information from grid sections and report on test results and related defects

Dr. Einar W. Karlsen

Generating compliance-relevant reports is complicated, because document requirements can depend on local software customizations, regulatory requirements, or internal standards. This article shows how to use the IBM Rational Publishing Engine to generate archetypical test management documents extracting information from IBM Rational Quality Manager.

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)

Generating compliance-relevant documents requires incorporating the different reporting needs and requirements that users of IBM® Rational® Quality Manager have. Local customizations of categories, attributes, and sections, as well as specific setup and regulatory requirements or internal standards, can affect the process.

Part 1 of this article series about reporting data in Rational Quality Manager using IBM® Rational® Publishing Engine focused on test plans, test cases, test results, and links to requirements.

Part 2 discusses how to extract more information regarding the test cases — in particular, information contained in grid sections. Next, you'll see how to retrieve information about the test scripts associated with the test cases, including the keywords and test steps. Then you'll learn about reporting on test results, including individual test steps. Finally, you'll see how to report on
related defects managed by IBM® Rational® Team Concert™ should a test case fail and a defect created in response.

**Test case details**

The schema for test cases provides access to standard test case properties such as `webId` and `title`, as well as user-defined rich text sections (see the left half of Figure 1). Queries in the form of URLs (`hrefs`) provide access to linked test scripts, which means that dynamic data source connections are required to retrieve the information related to these artifacts. The right half of Figure 1 shows the schema for test scripts. The information relevant for the test steps includes the test `stepIndex`, `description`, and `expectedResult`, as well as the test script name (`title`) and `webId`.

**Figure 1. Test case and test script schemas**

The template fragment in Figure 2 shows how to start reporting on test case details. A dynamic data source connection links from a test plan to the test cases of the test plan using the schema `RQM TC`. For each test case (query `$259`), a header is rendered containing the web identifier and test case title, followed by the test `description` and `Test Equipment`. (`Test Equipment` is a custom section defined in Rational Quality Manager.) Reporting on test equipment has been implemented using an external template (snippet) that was imported by reference (shown in green). Next, all test scripts for the test case (query `$260`) are rendered, and the details for each are fetched from the reporting data services using a dynamic data source configuration bound to the schema for test scripts (`RQM TS`). See Figure 2.
Grid sections

Part 1 showed how to report on rich text sections, but Rational Quality Manager also offers grid sections, which are basically tables with a fixed set of named columns but a variable number of rows. You can use grid sections to define a set of test equipment needed for a given test case (see Figure 3).

Figure 3. Grid sections in Rational Quality Manager

The challenges of reporting on grid sections

Reporting on grid sections in Rational Quality Manager introduces a challenge, because — unlike rich text sections — grid sections aren’t exposed by the imported schema. These import limitations do have a workaround: you can declare the section as a rich text section in Rational Quality
Manager first, then import the schema into Document Studio and change the section definition from a rich text to a grid section in Rational Quality Manager. However, if you do this, you must manually maintain (for example, edit) the schema in Document Studio if you try to re-import it (if anything changes in the Rational Quality Manager schema).

Even with the workaround, one challenge remains: grid sections return the values in JSON format and need to be parsed. The raw text received from the REST service contains a title row with column names and one or more value rows:

```
{"columnNames": ["No", "Title", "Quantity"], "value": [["5s", "iPhone", "1"], [" 5", " Camera", " 1"]]}
```

Such a string can be rendered as a table, but parsing the string and extracting the individual cell components requires extensive use of JavaScript. A snippet that can be reused everywhere has been developed to do exactly this. It works with two global variables: `_GridValue`, which contains the JSON string delivered by Rational Quality Manager, and `_GridTableTitle`, which contains the title of the table to be generated. The snippet was imported by reference and is shown in green in the template in Figure 2. The template also contains the initialization of the variables `_GridValue` and `_GridTableTitle` so that the correct values are passed to the template.

**Using the JavaScript snippet**

Look at the template in Figure 4. In the first JavaScript code element, the string is split into two parts: the value rows (assigned to variable `_GridValueRow`) and the title row (assigned to variable `_GridTitleRow`). To keep things simple, you'll see how to split the title row into individual table cells in the second Java Script code element.

An iteration has been defined that continues iterating as long as `_GridTitleRow` isn’t empty. Immediately within the iteration, an assignment is done to the variable `_GridCell`, which fills the next title row cell with a value. To stop the iteration, `_GridTitleRow` is stripped off the content of the cell just being rendered. The JavaScript code needs to distinguish between two cases: more than one title row cell needs to render (for example, a comma can be found in `_GridTitleRow`), or there is just one final title to render (co comma can be found in the string).

Parsing the sequence of value rows is more complicated, but similar in principle. Complexity is increased because there is only one title row, but there can be between 0 or n value rows in the JSON string. See Figure 4.
**Test scripts**

The template in Figure 2 showed how to get access to the test scripts associated with a test case by using a dynamic data source connection associated with the schema **RQM TS**.

After you do this, rendering the details of the test script (such as the name and description) is straightforward — and similar to reporting on test plan and test case properties. The test steps are rendered in a table with columns for the test step, the description, and the expected result (see Figure 5). The second row of the table has a query ($7) that iterates over all test steps and retrieves the index, description, and expected results using simple queries which are put in the appropriate table cells.
Keywords

The template in Figure 5 would work perfectly well for extracting test steps if it weren’t for keywords. Rational Quality Manager allows a user to define keywords — reusable fragments of test scripts — which can then be referenced in a test script. Keywords typically are used to define frequently repeated actions such as login and logout. The keyword has an associated test script that defines the user actions associated with the keyword. Basically this means that if you want to report on test scripts that contain keywords, you need to do more than what Figure 5 shows. You can see these additional steps — navigating the "chain" from test step to keyword to the test script associated with the keyword — in Figure 6.
To include information regarding keywords, you must first query the keyword linked to the step, if any (see query $216$). To retrieve the detailed information of the keyword, you must use a dynamic data source connection using the schema **RQM Keyword**. For the retrieved keyword ($217$), the **step index** and **title** are retrieved first and put in a table row of its own.

However, you're not done yet. The test steps of the keyword aren't directly stored with the keyword; rather, they're defined by an associated test script (the **defaultscript**). Consequently, the default script associated with the keyword must be retrieved using a query ($219$). Insofar that the default script is represented as a URL to the script, another dynamic data source connection using schema **RQM Keyword TS** is required to fetch the details for the default test script. From there, it's just a matter of iterating over the test steps and extracting the test step index, description, and expected result. (This template works 100% only if the default script of the keyword doesn't reference other keywords.)

### Test logs and test results

The result of executing a test case for an execution work item is an **execution result record**, which has several properties relevant for reporting (see the schema **RQM ER** in the left part of Figure 7).

- Points attempted, blocked, deferred, failed, inclusive, or passed
- Execution state (for example, verdict)
- Start-time and end-time
- List of step results, including step index, expected result, and actual result
- Associated defects in terms of URLs
Defects can also be associated with the individual test steps. The defect is simply referenced by a URL, which means that a dynamic data connection is needed to extract the details of the defect — basically a Rational Team Concert work item. The schemas for work items (see the right part of Figure 7) provide access to basic work items properties such as `id`, `title`, `priority`, and `severity`, just to mention a few.

**Figure 7. Execution result and defect schema**

Extracting information about test results

You saw how to get from a test case to an execution result through an execution work item in Part 1 of this series. Figure 8 shows the relevant template fragment that extracts the execution results from an execution work item, including the rendering of some basic properties like the verdict and start time.
The execution work item has a URL to the current execution result (query $197). Given this URL, you can use a dynamic data source connection to fetch the execution result using the schema RQM ER. Here it's important that the URL is appended with the field specification "?calmlinks=true" to enable reporting on associated links (to defects managed by Rational Team Concert). If this specification isn't provided, the REST service won't return any links to the defects. After you retrieve the execution result ($198), extracting the verdict and start time (amongst others) is quite simple.

Notice that the execution verdict has been computed from the basis of the execution result state (query $200). Result states are returned as Java names, so the variable _ExecutionVerdict is defined as:

```java
index = _value.lastIndexOf(".")+1;
_value.substring(index)
```

Moreover, the color of the text field has been defined, again using JavaScript, as shown in Listing 1.

**Listing 1. Using JavaScript to define the color of the text field**

```javascript
if (_ExecutionVerdict.equals("failed") || _ExecutionVerdict.equals("blocked") ) {
  "FF0000";
} else if (_ExecutionVerdict.equals("error")) {
  "FF8000";
} else if (_ExecutionVerdict.equals("passed") ) {
  "8000";
} else {
  "0";
}
```
The basic logic for extracting the details of the execution result steps is defined in Figure 9 (you can find the schema in Figure 7). Notice a table rendering the step index, Description, Expected Results, Actual Result, and verdict (State). The query ($222) in the second row iterates over all execution step result records, and the relevant properties of the step result are used to fill the appropriate table cells with data. At times, Rational Quality Manager may return the string "<data attribute not found>" for the expected results or actual result. Conditions have been defined to ensure that the cells are left empty should this be the case.

Figure 9. Execution result steps

Filtering according to test iteration

The template shown in Figure 9 for extracting test results doesn't filter the test results according to criteria such as the test environment or test milestones. However, you can filter test execution work items according to test environments easily if you encode the test environment in the name of the test execution record in Rational Quality Manager. In that case, a filter simply tests if a certain string is contained in the name of the test execution record.

Filtering according to iteration is a bit trickier. The iteration associated with a test run is available through the testphase query of the execution work item (see query $303 in Figure 10). However, this information is just the URL of the test phase, so you need a dynamic data connection using the schema named RQM TP to retrieve the test phase. This name of the test phase ($304) can be assigned to a variable and then used elsewhere to determine if the given execution work item is really in scope for the current report or not.
Figure 10. Test iteration

Extracting information about associated defects

The execution result records of Rational Quality Manager provide information about the associated defects in form of URLs to the defects. Obtaining detailed information for the associated defects in a table, therefore looks straightforward. As Figure 11 shows, you can simply connect to the schema for work items using the URL that Rational Quality Manager returns, then iterate over the work items ($204), extract the work item (id, summary, state, resolution, severity, and so on) as needed, and assign the value of the corresponding queries to the appropriate table cells.

Figure 11. Defects

However, behind the scene it is a bit more complicated than that. The basic tricks for reporting on work items associated with test results have already been documented (see Related topics), but reporting using Rational Team Concert works differently from reporting using Rational Quality...
Manager. The main difference lies in the kind of URL supported by the two reportable REST services.

The URLs to the defects associated with execution results in Rational Quality Manager are returned in the form:


Access to all Rational Team Concert work items, however, are provided through a URL of the form:

https://localhost:9443/ccm/rpt/repository/workitem

This URL would return all work items in the repository. Of course, you could iterate over all the work items until the referenced work item is found, but such an approach works only with small data sets. However, Rational Team Concert provides field specification that can narrow the scope of work items returned to a single work item. For example, the next URL returns the properties for the work item with id equal to 91:

https://localhost:9443/ccm/rpt/repository/workitem?fields=workitem/workItem[id="91"]/(* | */*)

Therefore, the JavaScript expression shown in Listing 2 is used to configure the data source connection RTC WI (it basically constructs a URL similar to the last one).

**Listing 2. Configuring the data source connection "RTC WI"**

```javascript
URL = _RTC_I_WI_URL;
baseURL = URL.substr(0,URL.indexOf("/ccm/")) + "/ccm/rpt/repository/workitem";
index=URL.lastIndexOf("/");
id = URL.substr(index+1,URL.length);
field = "fields=workitem/workItem[id=" + id + "]/(* | */*)";
baseURL + "?" + field
```

The internal variable _RTC_WI_URL stores the URL to the defect. The URL is basically defined through an assignment to be the result of the query:

`executionresult/defect/href`

The second line of code shown in Listing 2 computes the base URL for Rational Team Concert. The next two lines fetch the id of the work item, and the two last lines compute the field specification and append it to the base URL. Extracting information about the defects associated with individual test steps follows in an equal manner.

**Conclusion**

**Part 1** of this article series about Rational Quality Manager reporting with Rational Publishing Engine covered the data model, Reportable REST service URLs, schema import, and basic template authoring for test plans, test cases, test results, and test requirements (fetching information from Rational Requirement Composer). Here in Part 2, more advanced concepts for Rational Quality Manager reporting — including reporting on grid values, test scripts, test logs, and associated defects — were introduced.
Acknowledgement

Thanks to Dragos Cojocari, Peter Staubrandt, Abish Mathew Zachariah, and Gerd Schiering for providing valuable input to the two articles on using Rational Publishing Engine for reporting over data in Rational Quality Manager.
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.
- 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 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.
  - Check the Rational Quality Manager pages on developerWorks for links to product documentation, articles, tutorials, courses, downloads, and other useful areas.
  - Explore the Rational Quality Manager Information Center for technical details, check the Rational Quality Manager Roadmap for links to more resources, and browse the IBM Quality Management page to learn more about what else is available.
  - For how-to tips, read these developerWorks articles by Michael Kelly:
    - Lab Management using IBM Rational Test Lab Manager: Extending IBM Rational Quality Manager for collaborative, comprehensive test planning and test asset management (developerWorks, October 2008)
    - Getting started with IBM Rational Quality Manager (developerWorks, December 2008)
    - Managing your first project with IBM Rational Quality Manager (developerWorks, October 2008)
  - Go to Jazz.net to find out more about Rational Quality Manager Reportable REST API.
  - 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 the free trial version of Rational Quality Manager.