12
Practical Example: Building Reports for Bugzilla

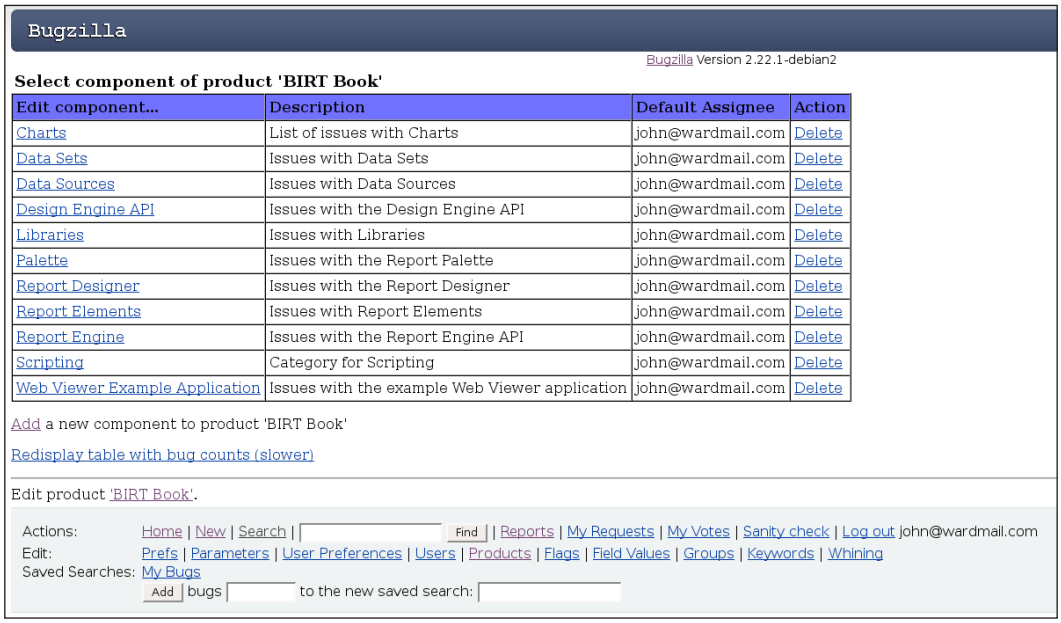
We have seen all the components of building reports with BIRT. By this time, we are now familiar with how to navigate the Eclipse BIRT Report Designer perspective, how to insert components into reports, create data connections and datasets based on queries and scripting, and various other topics related to BIRT.

As promised in the beginning, we are going to look at a practical example of building a reporting site. In this following sections of this chapter, we will be looking at a scenario where we have a Bugzilla instance setup with a series of bugs related to BIRT. We are going to look at a series of requirements and build the reports necessary to fulfil those requirements.

The environment

In this setup, we are utilizing a Bugzilla 2.22.1 instance set up on Ubuntu 7.10 Gutsy Gibbon. Bugzilla has been set up to run under Apache and is connected to a MySQL 5 database. While newer versions of Bugzilla and Ubuntu are available, the steps should remain the same.

There is a single product, called BIRT Book. Under this product, there are several components as illustrated in the following screenshot:



Requirements

Although we haven't talked about it much, the most important thing to have prior to building reports is the requirements. Imagine being a carpenter trying to build a house without any blueprints. We need to have some idea what we are trying to build before we undertake the task of building reports. The following list shows a set of requirements that we have for this project. These are actually fairly sparse in terms of requirements. In my experience, requirements range from an extremely detailed set of Use Case documents, to mock-ups done in spreadsheets or some graphic format.

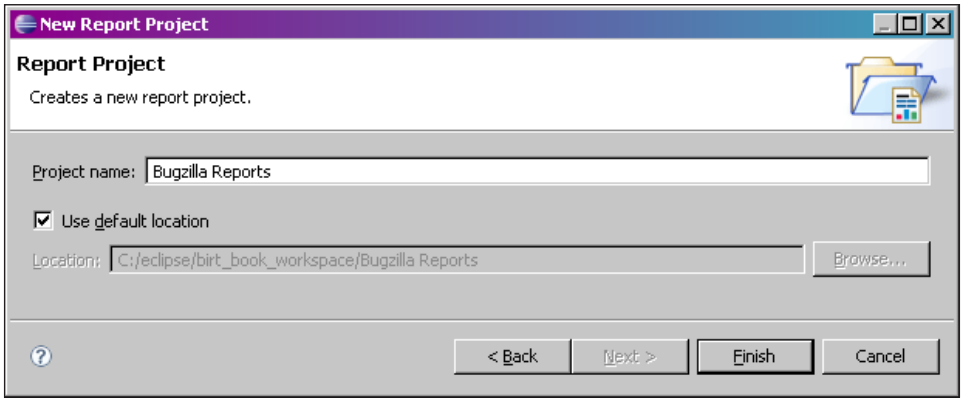
These are the reports the users are looking for:

- Detailed report about bug. Show who it is assigned to, take in bug ID as a parameter, will be target for all drill down hyperlinks in other reports.
- Report to show overall status of issues. This report will drill down to a detailed list of issues.
- A report showing list of bugs assigned to a developer when provided with login details.
- Performance report for users, showing percentage of issues in the form of finished state versus open state.

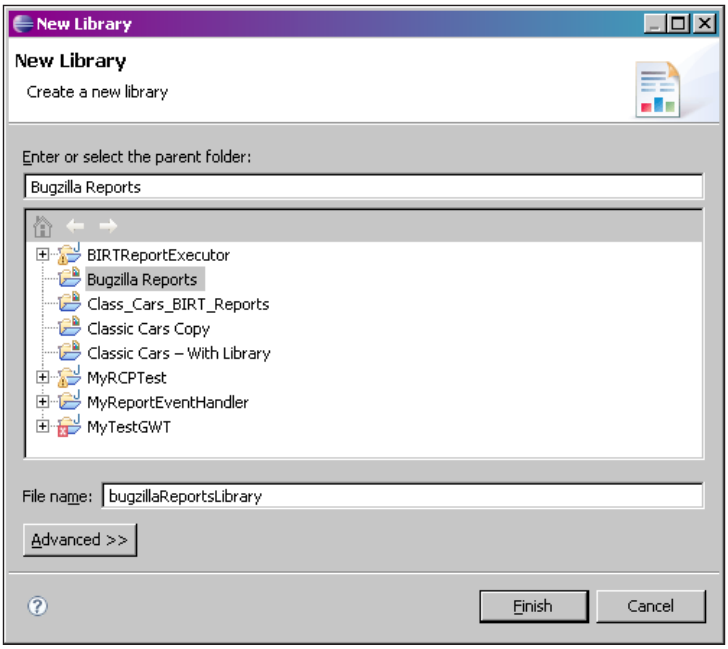
Creating libraries

In the case of these reports, we have a pretty good idea what kinds of things we want reusable. First, we know that all of these reports will contain the same data source, a MySQL connection to the Bugzilla database. We will also want to create a consistent header and layout for the reports. So, we will create a library containing the data source and the header, and create a template containing both. We will also create the stylesheets we want to use to make things consistent throughout the reports. Let's get on with this:

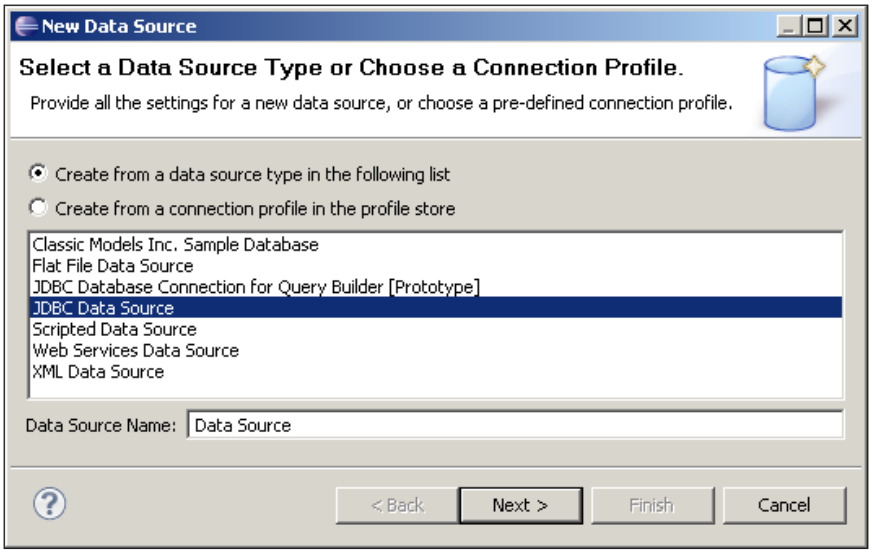
1. Create a new reporting project called **Bugzilla Reports**.



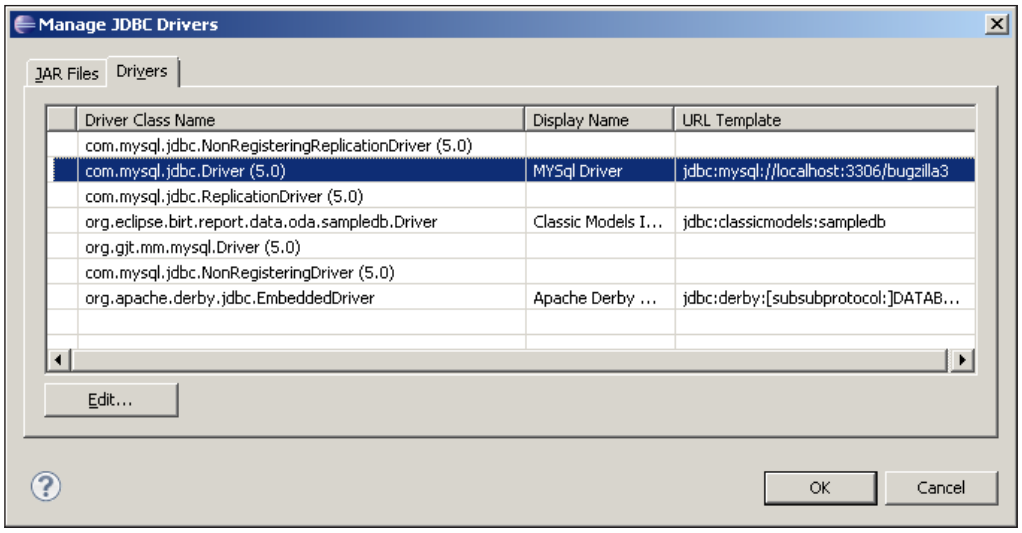
2. Create a new library called `bugzillaReportsLibrary.rptLibrary` in the newly created project.



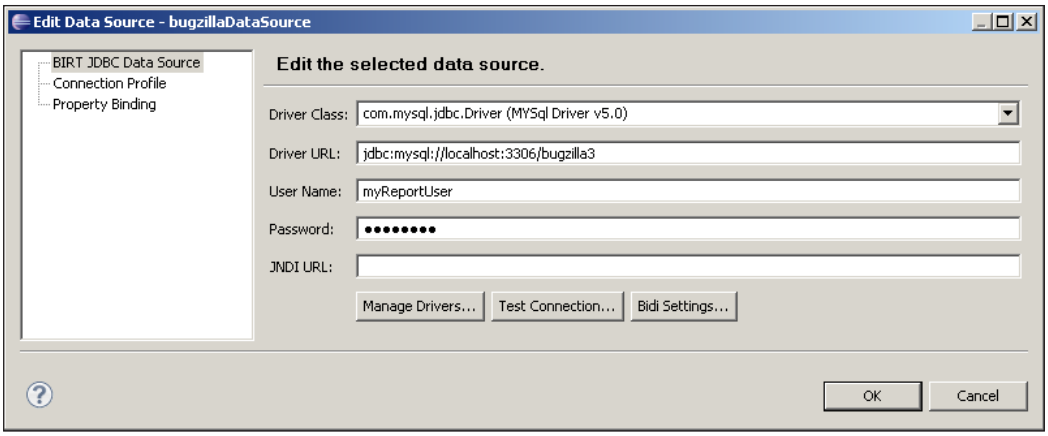
3. Switch to the **Outline** view in `bugzillaReportsLibrary.rptLibrary`. Create a new JDBC data source called `bugzillaDataSource`.



4. As we are using MySQL, we need to use the **Manage Drivers** dialog under the **Data Source** setup to install the MySQL JDBC driver. The following is the dialog screen where we edit the MySQL Connector-J driver to have a description and template JDBC URL:

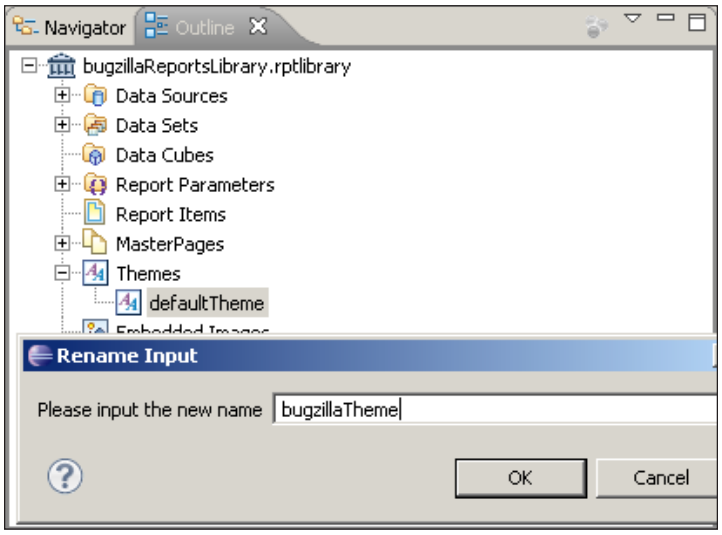


5. Input the correct JDBC URL and driver for Bugzilla.

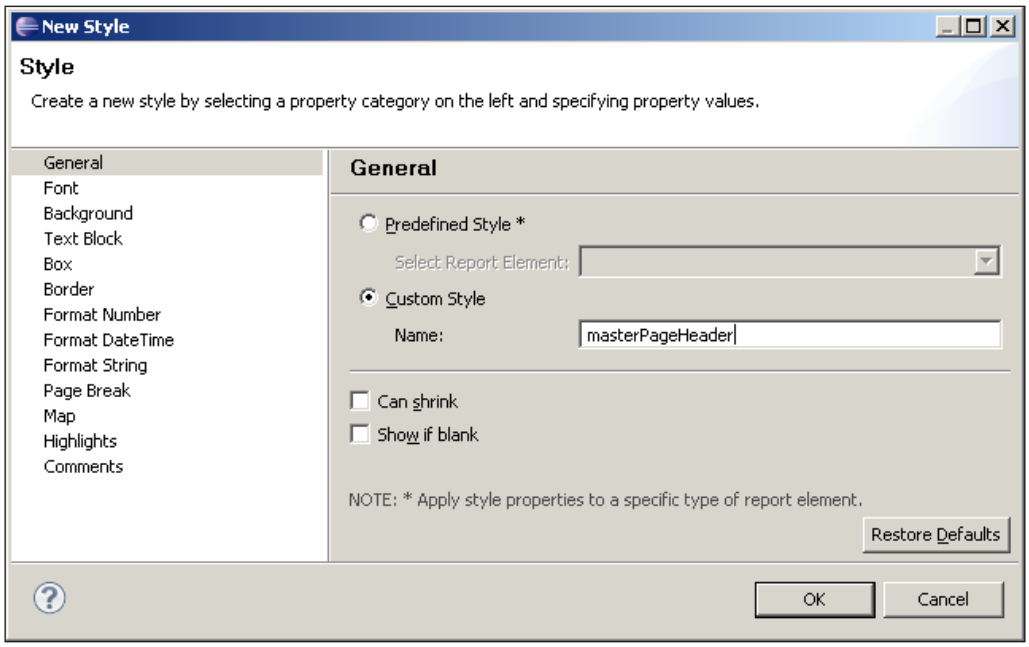


6. Select the **Themes** option under the **Outline** tab.

7. Change the name of the theme to `bugZillaTheme`.



8. Create a new custom style called `masterPageHeader`.



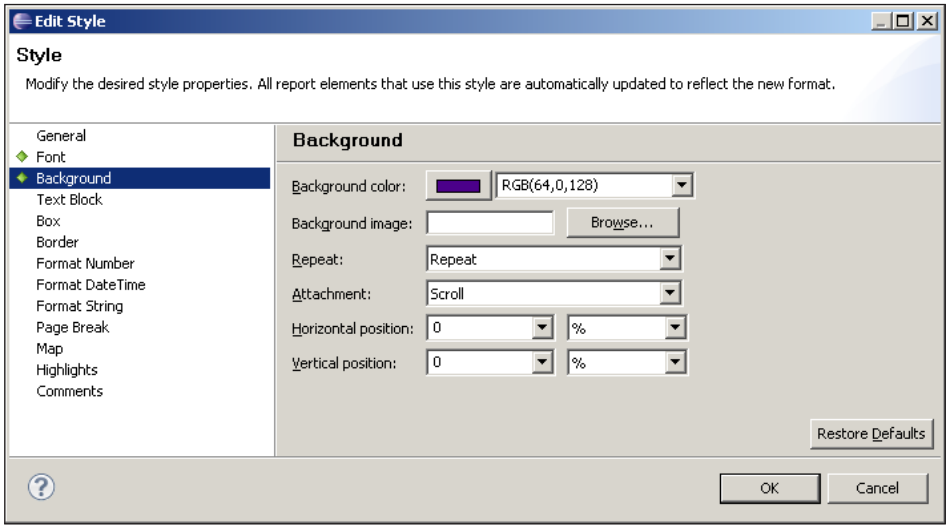
9. Input the following parameters for the Style:

   Under the **Font** tab:

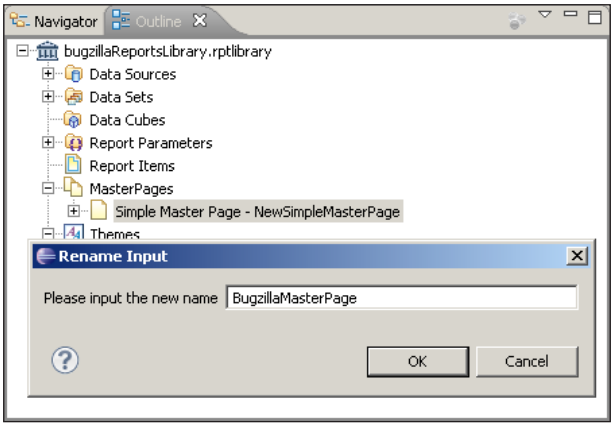
   - **Background color**: **White**
   - **Weight**: **Bold**
   - **Font**: **Sans-Serif**

   Under the **Background** tab:

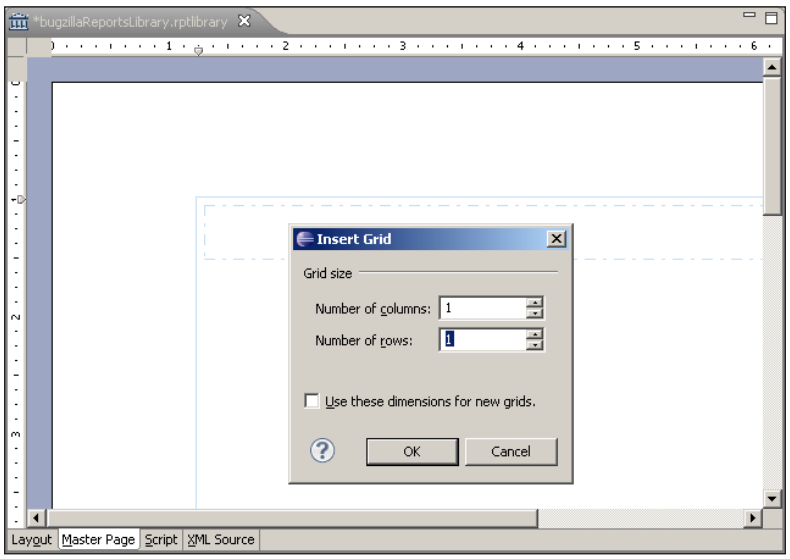
   - **Background color**: **RGB(64,0,128)**



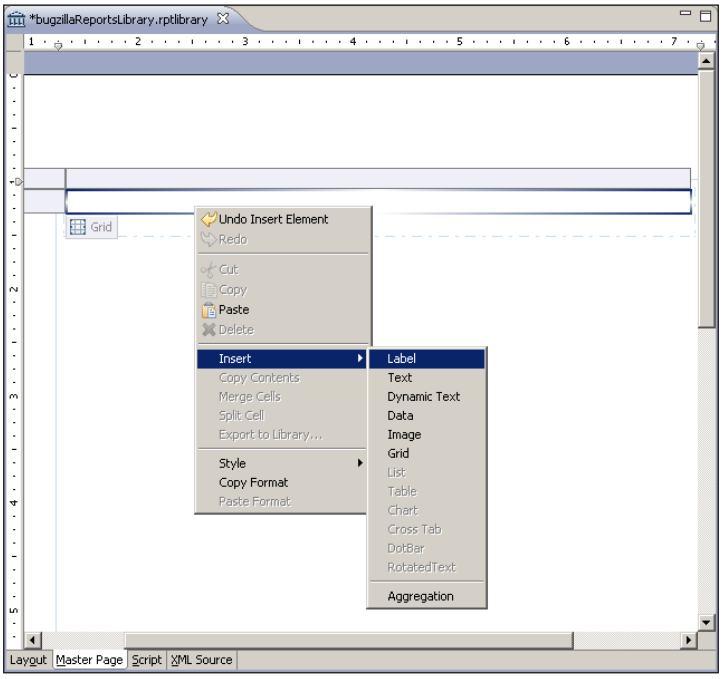
10. Select the **Master Pages** element under the **Outline** tab. Change the name of the Master Page from `Simple Master Page` to `BugzillaMasterPage`.



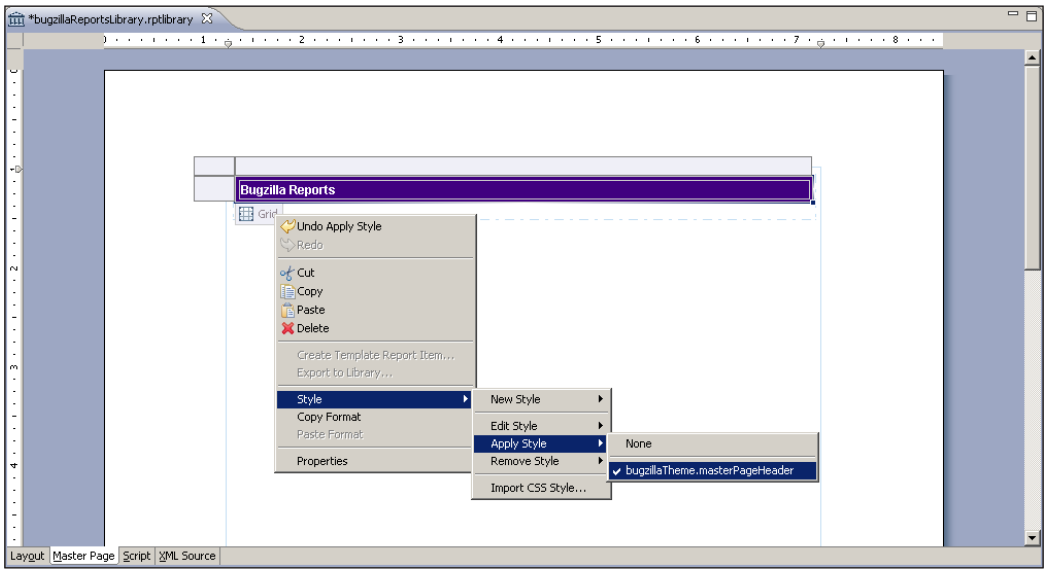
11. Select the **Master Page** tab in the **Report Designer**.
12. Insert a grid into the header with 1 column and 1 row.



13. Insert a Label component into the grid cell.



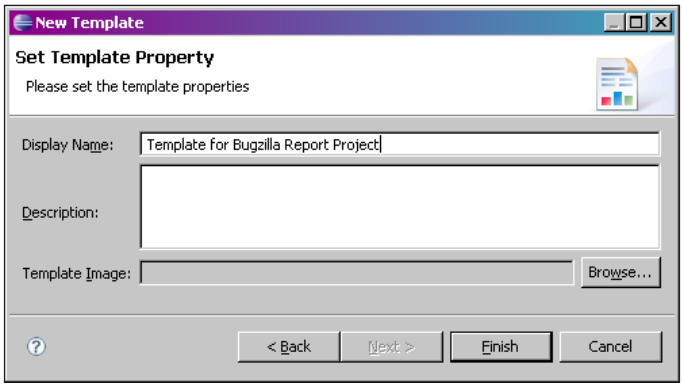
14. Enter the text as **Bugzilla Reports**.
15. In the Report Designer, right-click on the **Grid**, and select **Style**. Under **Style**, select **Apply Style | bugzillaTheme.masterPageHeader**. The following is the master page with the grid and style applied:



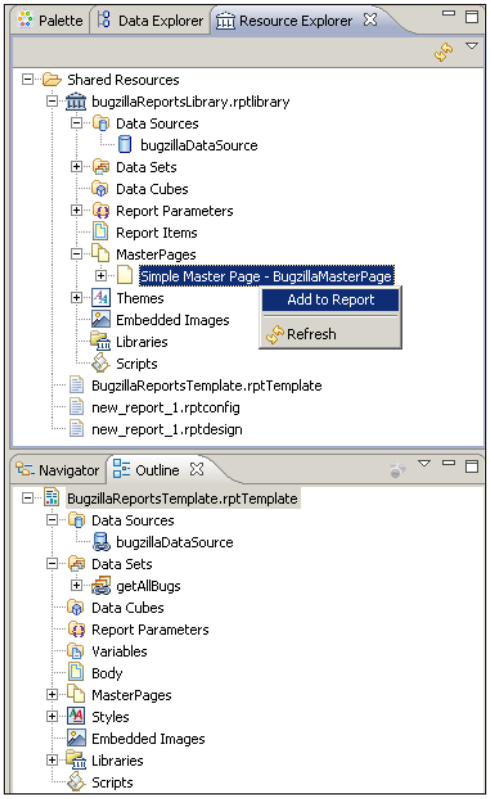
16. Create a new dataset called `getAllBugs`. Use the following query:

```
SELECT
            bugs.bug_id,
        bugs.bug_severity,
        bugs.bug_status,
        bugs.short_desc,
        profiles.userid,
        profiles.login_name,
        profiles.realname,
        components.id,
        components.name,
        components.description
FROM
        bugs,
        profiles,
        components
WHERE
        bugs.component_id    = components.id
        AND bugs.assigned_to = profiles.userid
```

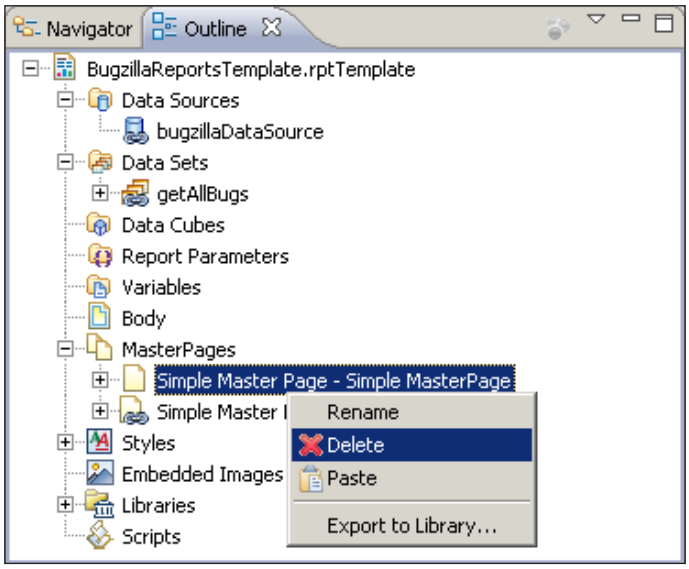
17. In the project, create a new template called `BugzillaReportsTemplate.rptTemplate`.



18. In the newly created template, open the **Resource Explorer** tab and the **Outline** tab. Drag-and-drop the `bugzillaDataSource`, `getAllBugs`, and `BugzillaMasterPage` components from the library to the template.



19. From the **Outline**, delete the **Simple Master Page** from the template.



20. Save the template.

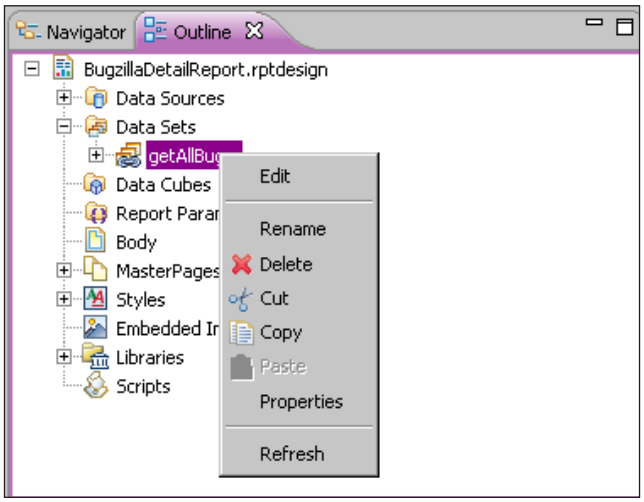
First report—bug detail report

Now that we have the groundwork laid for our project, we can start building the actual reports. In a typical situation, one wouldn't know beforehand every possible element to add to a report library. So, in this next example, we are going to build the report, then edit it to add some formatting elements such as styles that will be used in the remainder of the reports.

The following report is fairly straightforward. We already have the query to retrieve information about bugs from the template we will use, so we will need to modify it in two ways. First, we need to parameterize it so that we return only the bug we are looking for. We will also need to pull in the bug history. These are both simple modifications, so let's take a look.

1. Open `BugzillaReportsTemplate.rpttemplate`.
2. Save this report as `BugzillaDetailReport.rptDesign`.

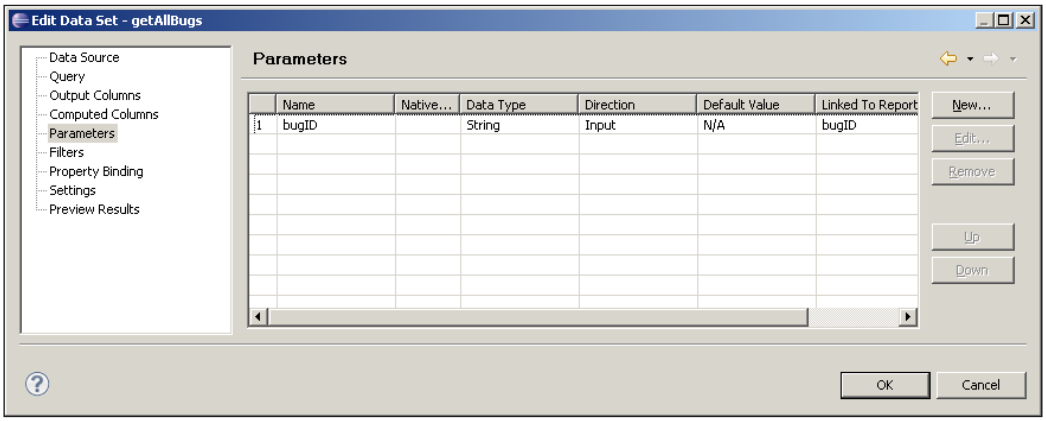
3. Edit `getAllBugs` dataset.



4. Edit the query to read like the following example:

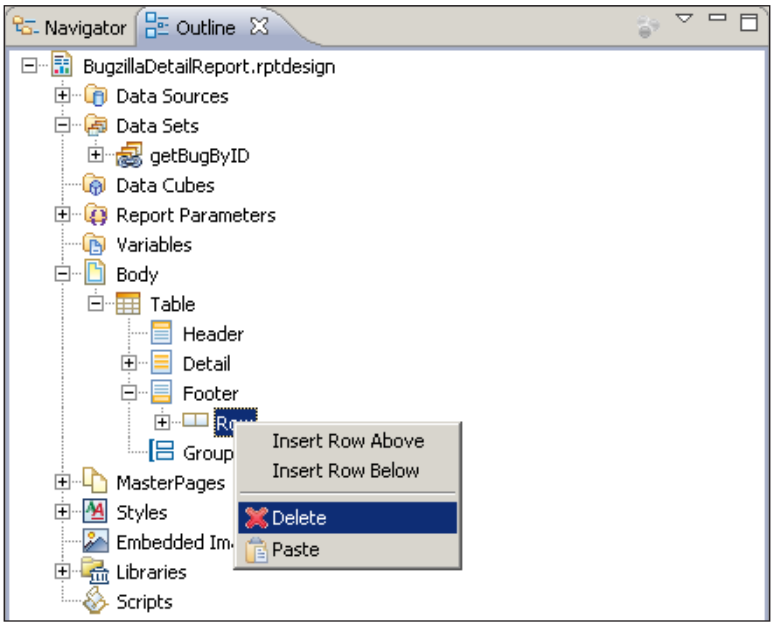
```
SELECT
            bugs.bug_id,
        bugs.bug_severity,
        bugs.bug_status,
        bugs.short_desc,
        profiles.userid,
        profiles.login_name,
        profiles.realname,
        components.id,
        components.name,
        components.description
FROM
        bugs,
        profiles,
        components
WHERE
        bugs.component_id    = components.id
        AND bugs.assigned_to = profiles.userid
        and bugs.bug_id = ?
```

5. Create a new dataset parameter and call it `bugID`. Link it to a report parameter.

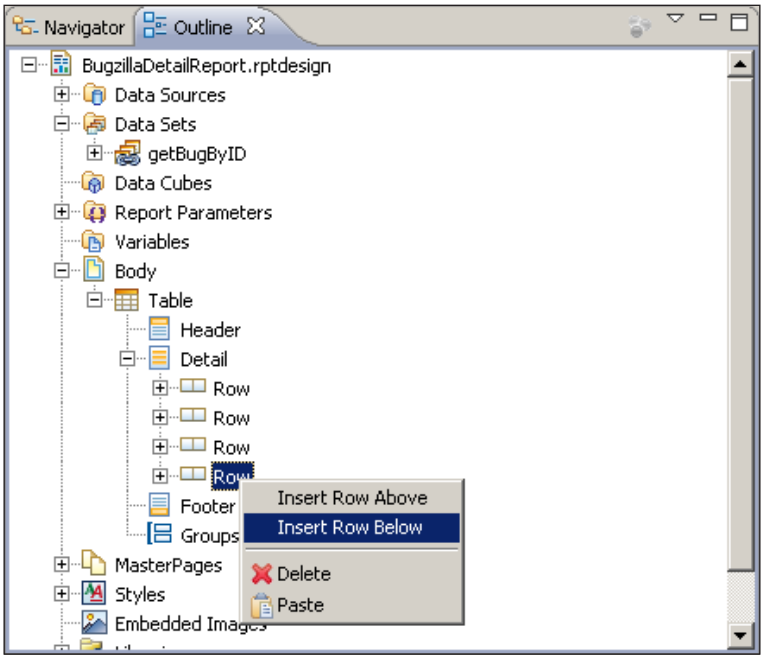


6. Save the changes to the dataset. From the **Outline** tab, right-click and choose rename. Rename the dataset to `getBugByID`.
7. Now, we want to create a Table element in the report that displays the bug information vertically instead of horizontally. To start, drag `getBugsByID` to the Report Designer.

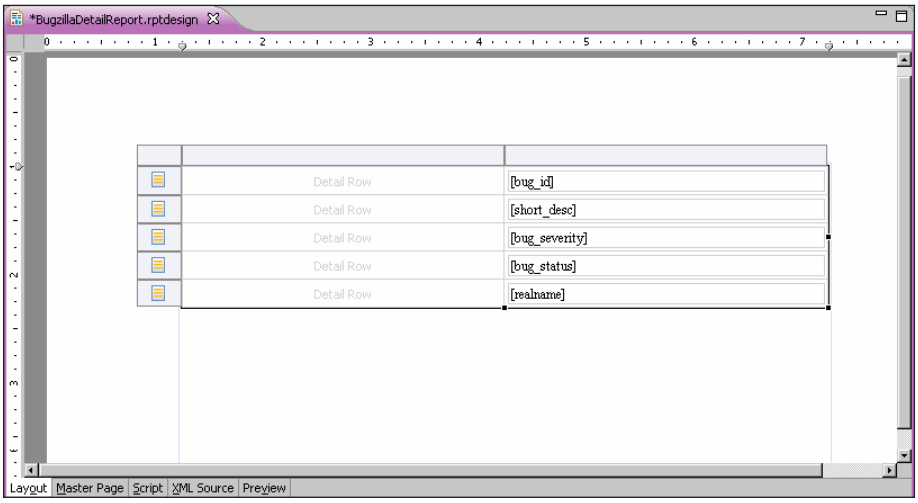
8. Delete the header and footer rows.



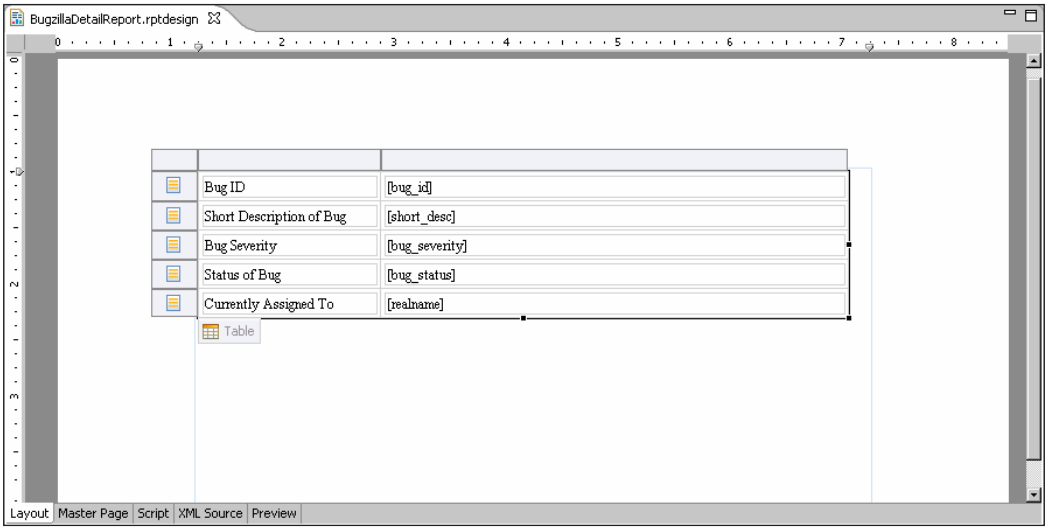
9. Insert four rows into the **Detail** section.



10. Move the following fields into the order as illustrated in the following screenshot. Remove the remaining columns.



11. Add descriptive labels to each row as seen in the following screenshot:



12. Now we want to create two Styles in our library and embed them into the report. This way the styles will be available in other reports also. Open `BugzillaReportsLibrary.rptLibrary`.
13. Under the **Themes** section, create a new style called `BugDescriptionHeaderLabel`.

14. Use the following attributes.

    Use the following settings for **Font**:

    - **Color**: **White**
    - **Size**: **Large**
    - **Weight**: **Bold**

    For **Background**:

    - **Color**: **Blue**

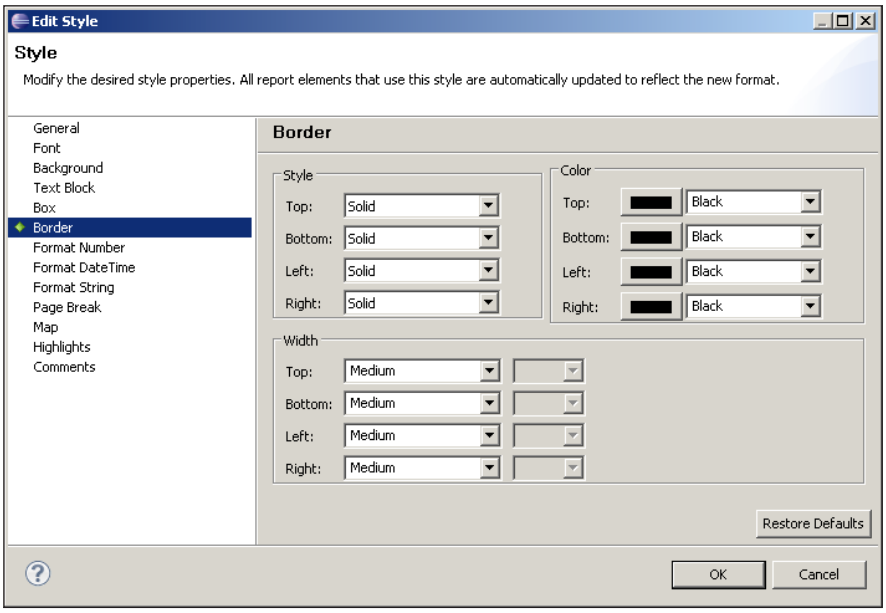
    For **Border**:

    - **Top**: **Solid**
    - **Bottom**: **Solid**
    - **Left**: **Solid**
    - **Colors**: **Black**

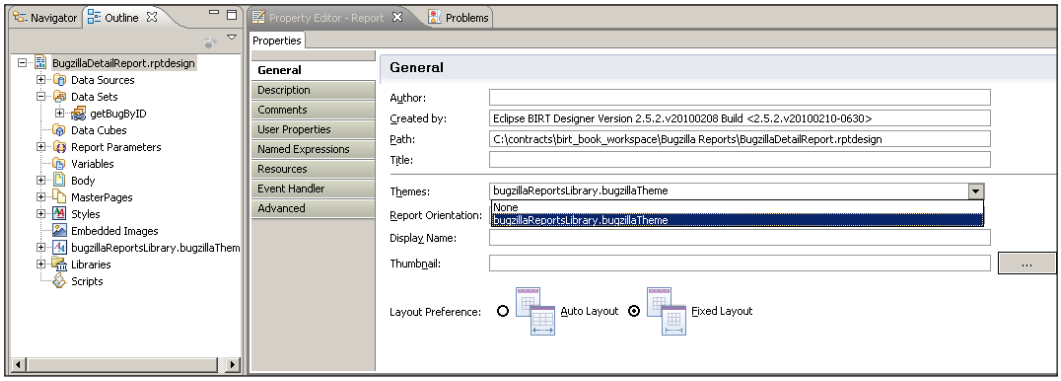
15. Create a new style called `BugDescriptionHeaderData`.

    Under the **Border** tab:

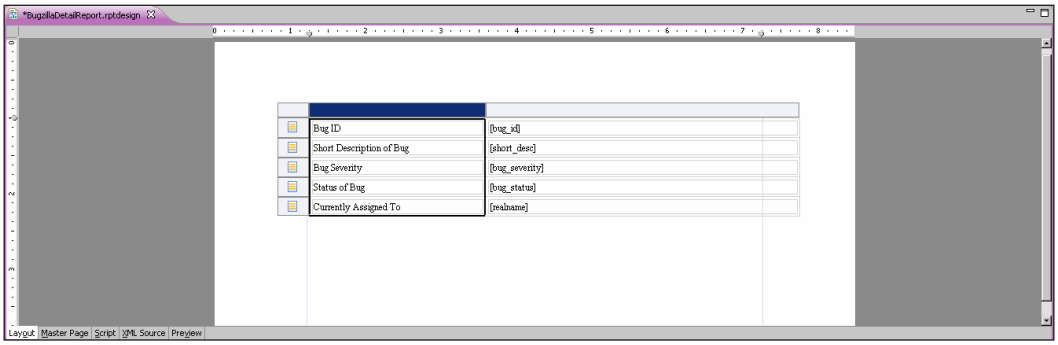
    - **Top**: **Solid**
    - **Bottom**: **Solid**
    - **Right**: **Solid**
    - **Colors**: **Black**



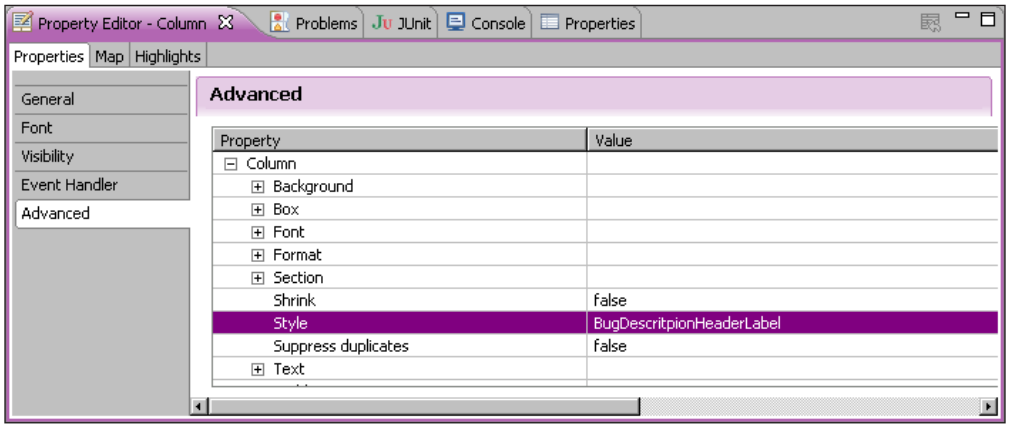
16. Save the library.
17. We have to close `BugDetailReport.rptDesign` before the new styles become visible. Go ahead and do so and reopen `BugDetailReport.rptDesign`.
18. In the **Outline** view, select the root element. Apply the `bugZillaTheme` theme.



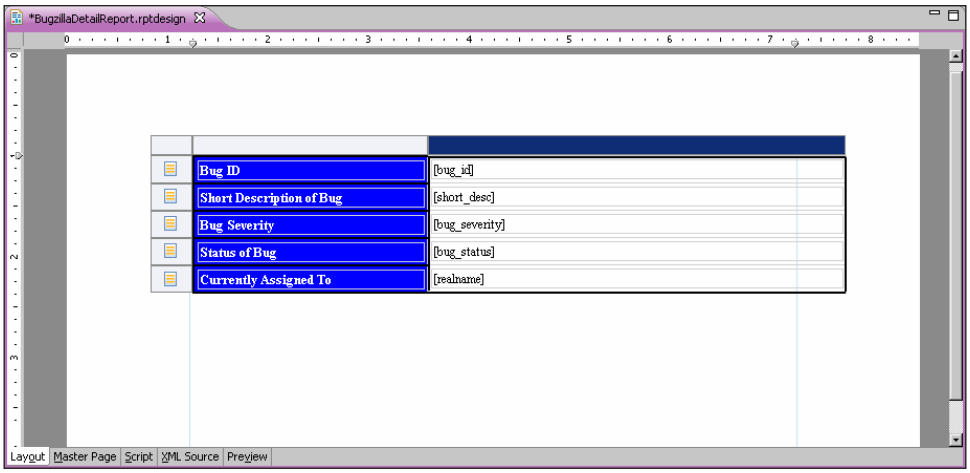
19. In the Report Designer, select the column containing the labels.



20. In the **Property Editor**, select the **Advanced** tab. Under style, select `BugDescriptionHeaderLabel`.



21. Select the column containing the data and apply the `BugDescriptionHeaderData` style.



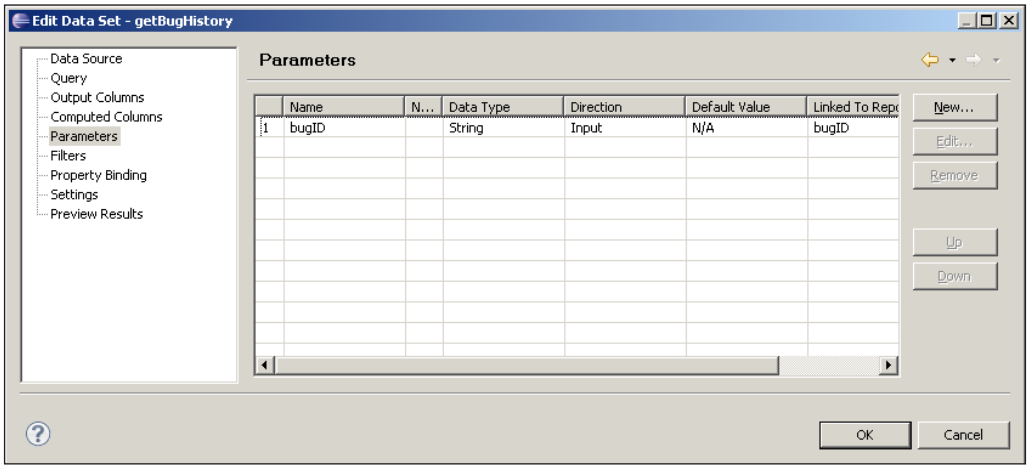
So we have created the header for the report. Now, we want to see the details of this bug. The details are basically the bug history of who changed fields, added fields, changed the bugs status, and a resolution.

1. Create a new dataset called `getBugHistory` using the following query:

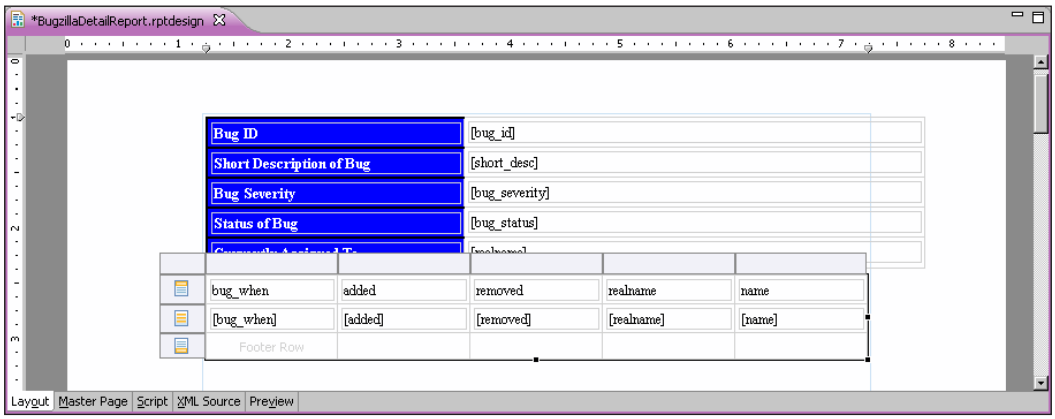
```
select
        bugs_activity.bug_when,
        bugs_activity.added,
```

```
        bugs_activity.removed,
        profiles.realname,
        fielddefs.name
from
        bugs_activity,
        profiles,
        fielddefs
where
        bugs_activity.who = profiles.userid
        and bugs_activity.fieldid = fielddefs.fieldid
        and bug_id = ?
```

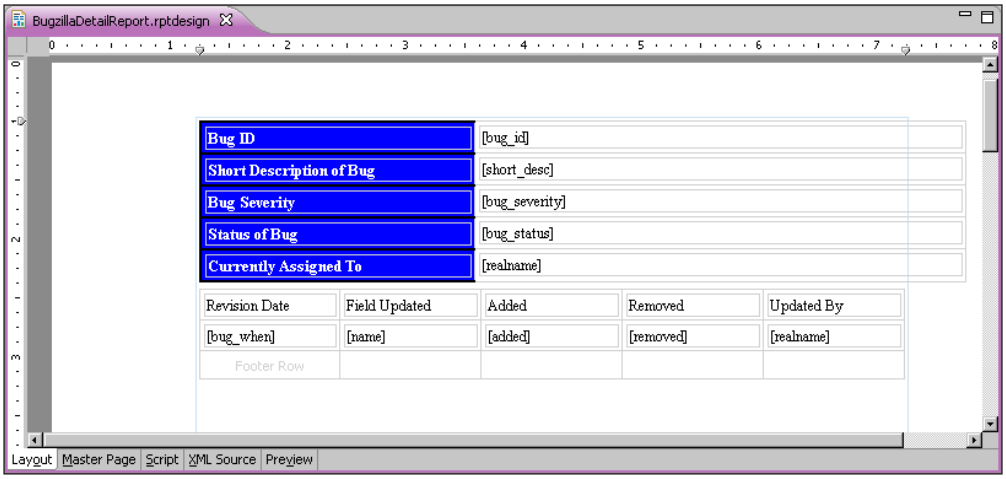
2. Add in a new parameter and bind it to the report parameter `bugID`.



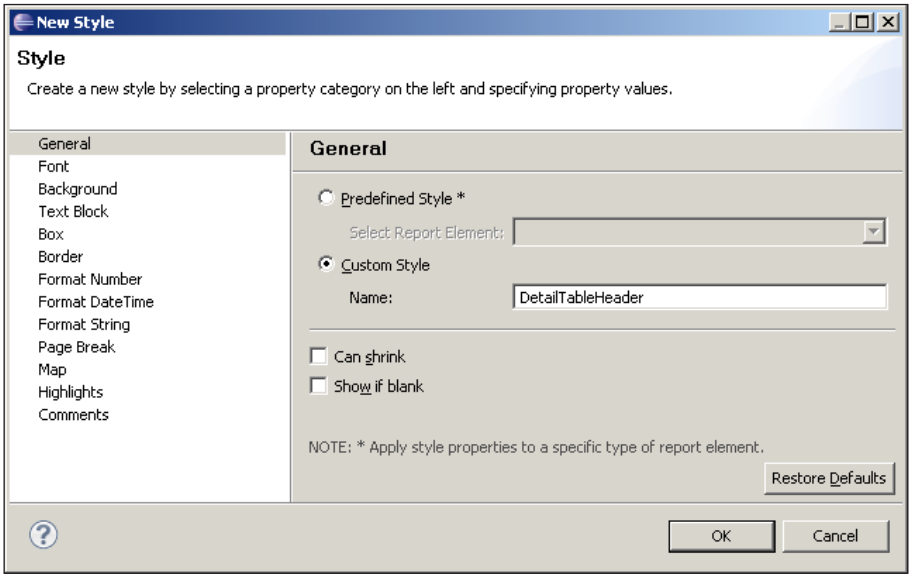
3. Drag the new dataset over to the Report Designer.



4. Update the header labels as illustrated in the following screenshot. In the screenshot, we have also moved the column containing the `Field Updated` field to the second column.



5. Now, we need to create styles for the detail row. We want to create these in the library also so that they are reusable in our other reports. Open the `BugzillaReportsLibrary.rptLibrary`.
6. Create a new style called `DetailTableHeader`.



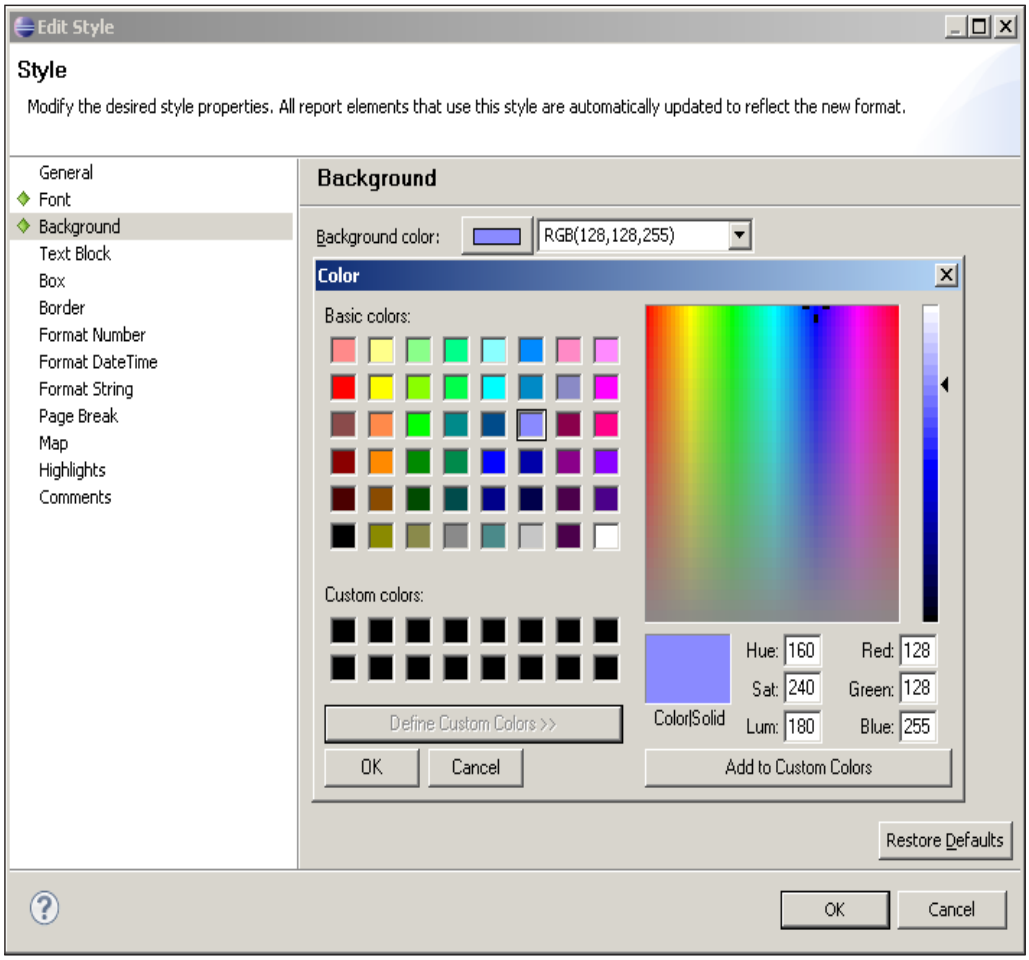
7. Use the following settings for the style:

   Under the **Font** tab:

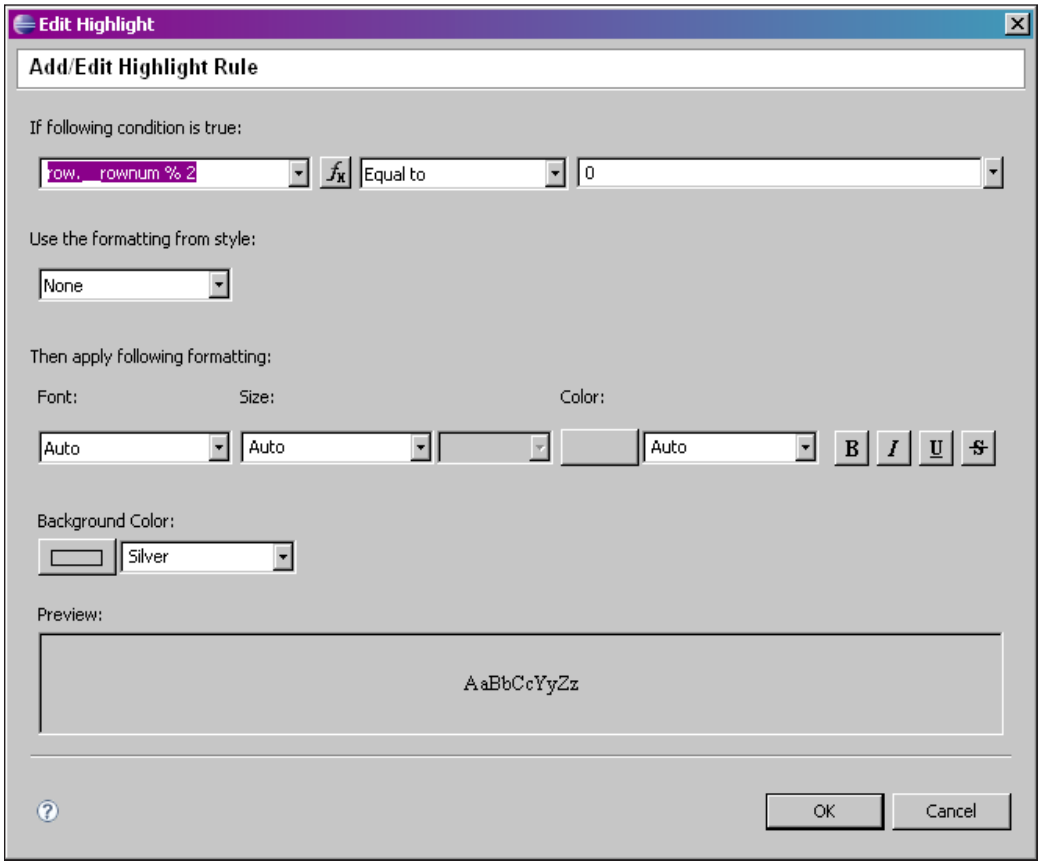
   - **Weight**: **Bold**

   Under the **Background** tab:

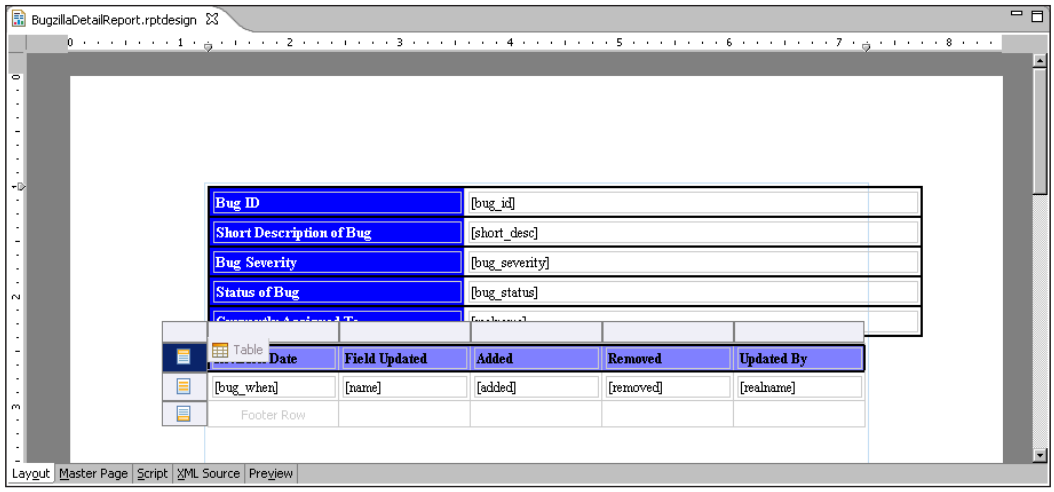
   - **Background color**: **RGB(128,128,255)**



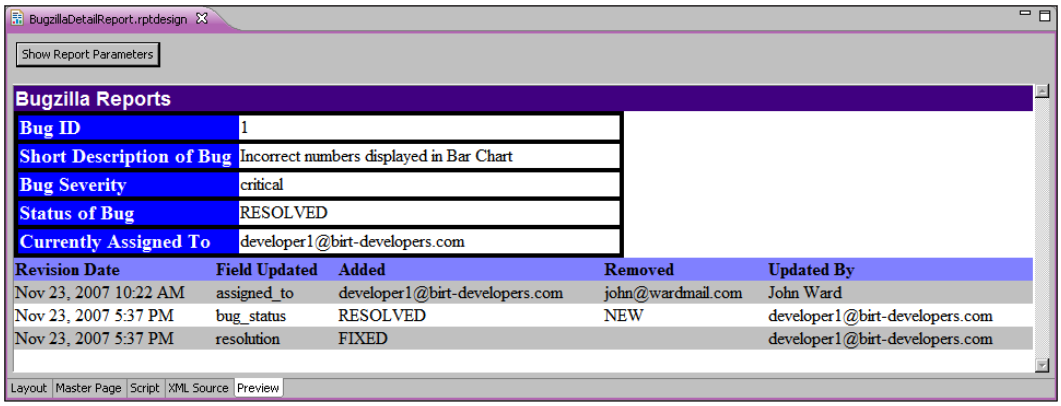
8. Create a second style called `DetailTableRow`. Under the **Edit Highlight** tab, enter `row_rownum % 2` as the expression and set the **Background color** to **Silver**.



9. Save the library. Again, close `BugzillaDetailReport.rptDesign` and reopen.
10. In the header row for `getBugHistory` table, apply the DetailTableHeader style.
11. In the detail row, apply the `DetailTableRow` style.



12. Save the report and preview it.

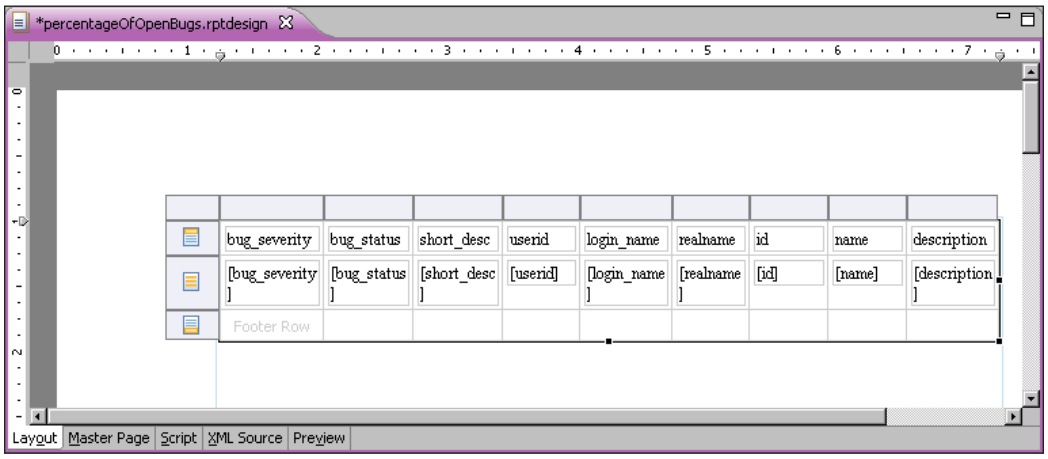


Bug status

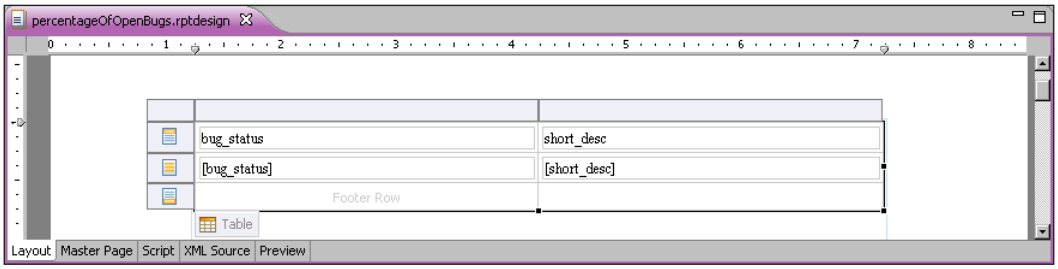
The next report we are going to undertake is the report to show open issues compared to all bugs. What we are trying to build here is a graph that shows, by category, the bugs that are open in relation to the total number of bugs for that category. No other grouping is required. Under the graph, we also want to show a list of open bugs that we can click on and have them drill down to the detail for that bug so that we can see the history of changes to that bug.

What this means is we need a bar graph showing all the status.

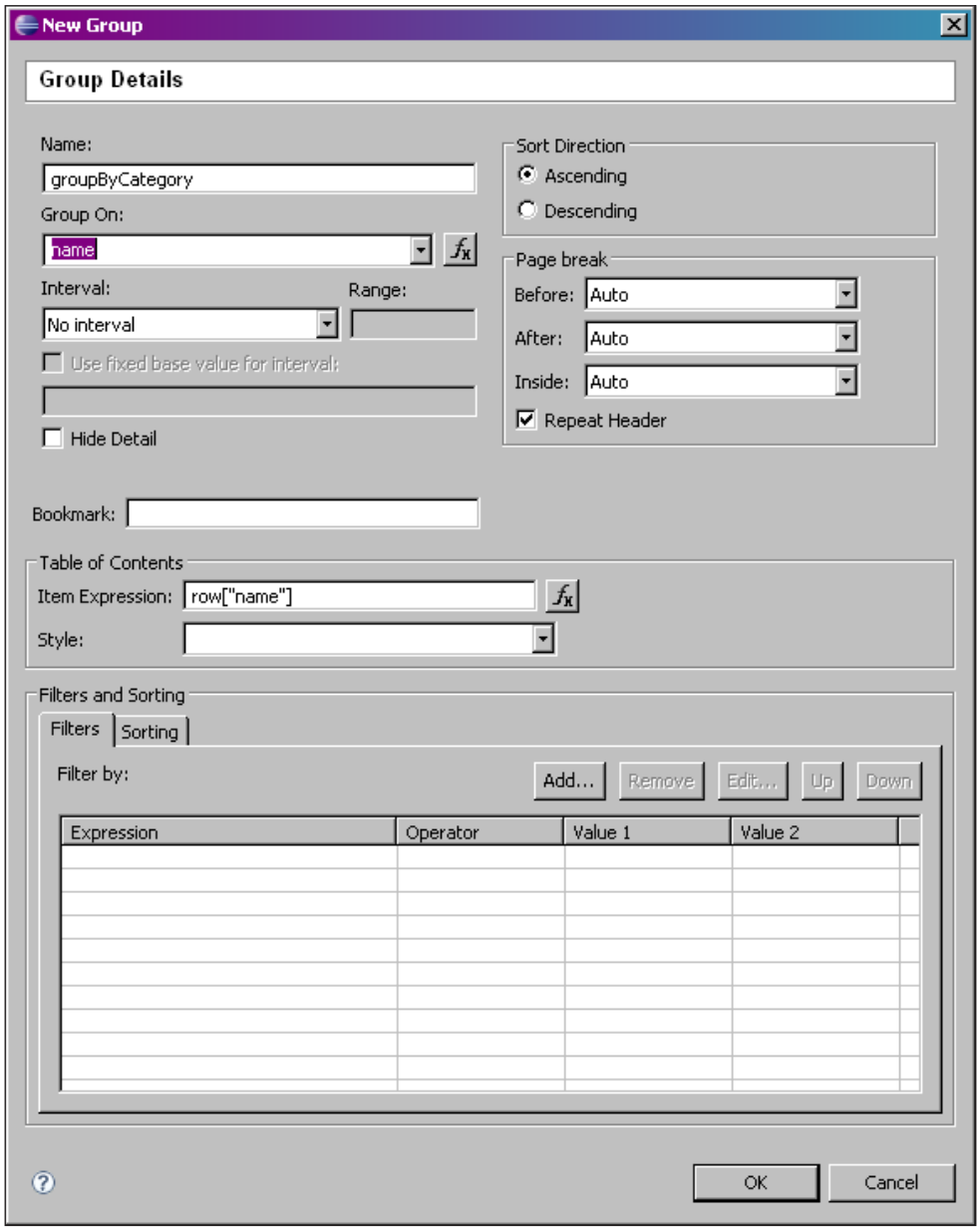
1. Open `BugzillaReportsTemplate.rptTemplate` and save as `bugStatusReport.rptDesign`.
2. Drag `getAllBugs` over to the Report Designer.



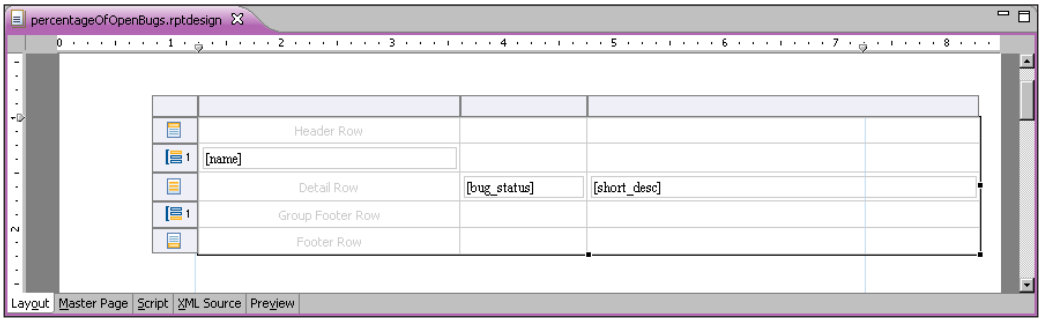
3. Delete all columns except the `bug_status` field and `short_desc` field.



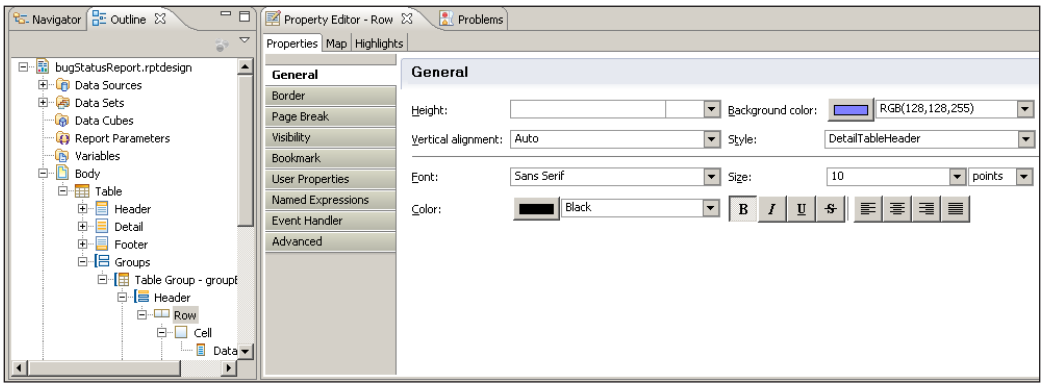
4. In the table, group data by the components' name.



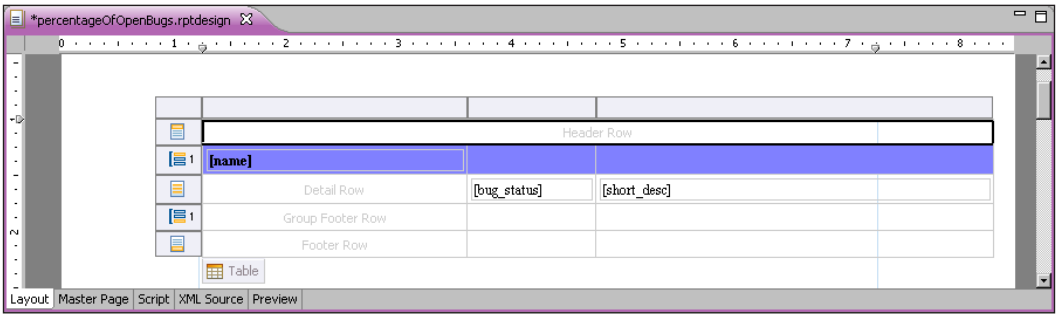
5. With the new category created in the Table, insert a new column on the right. Delete the header labels. Move the data fields to look like the following screenshot:



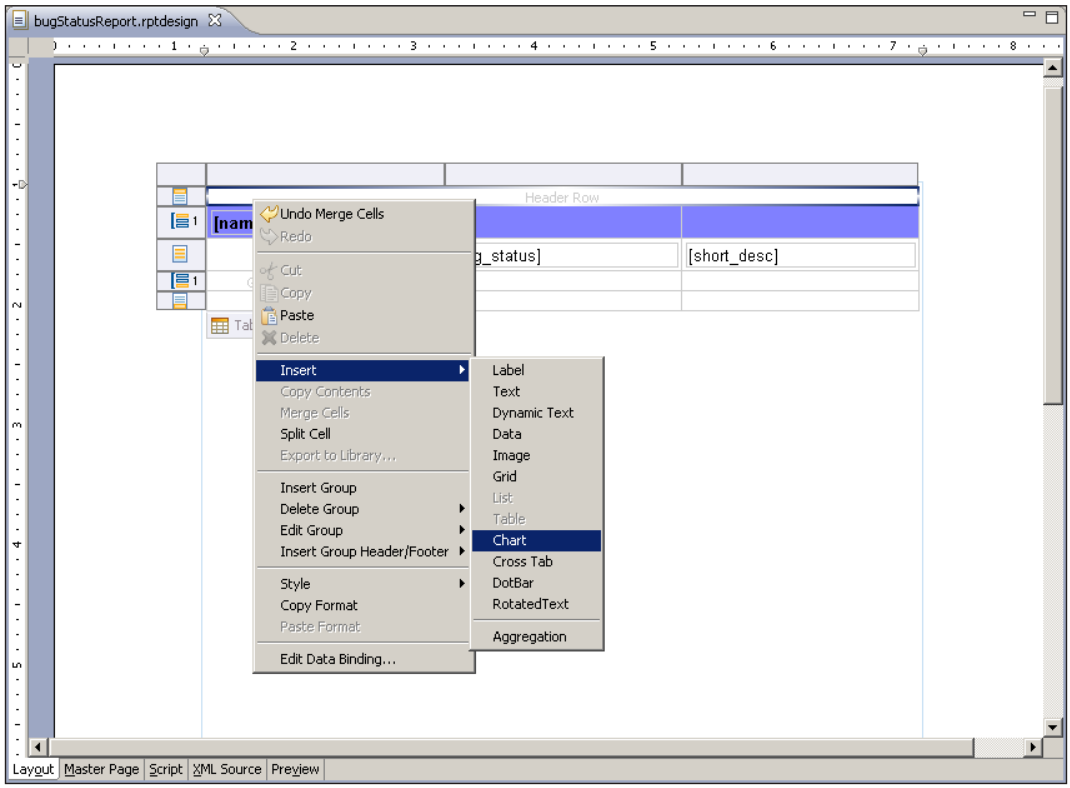
6. In the **Outline** tab, select the root element. Apply `bugZillaTheme`.
7. In the group header row with the name, apply the `DetailTableHeader` style.



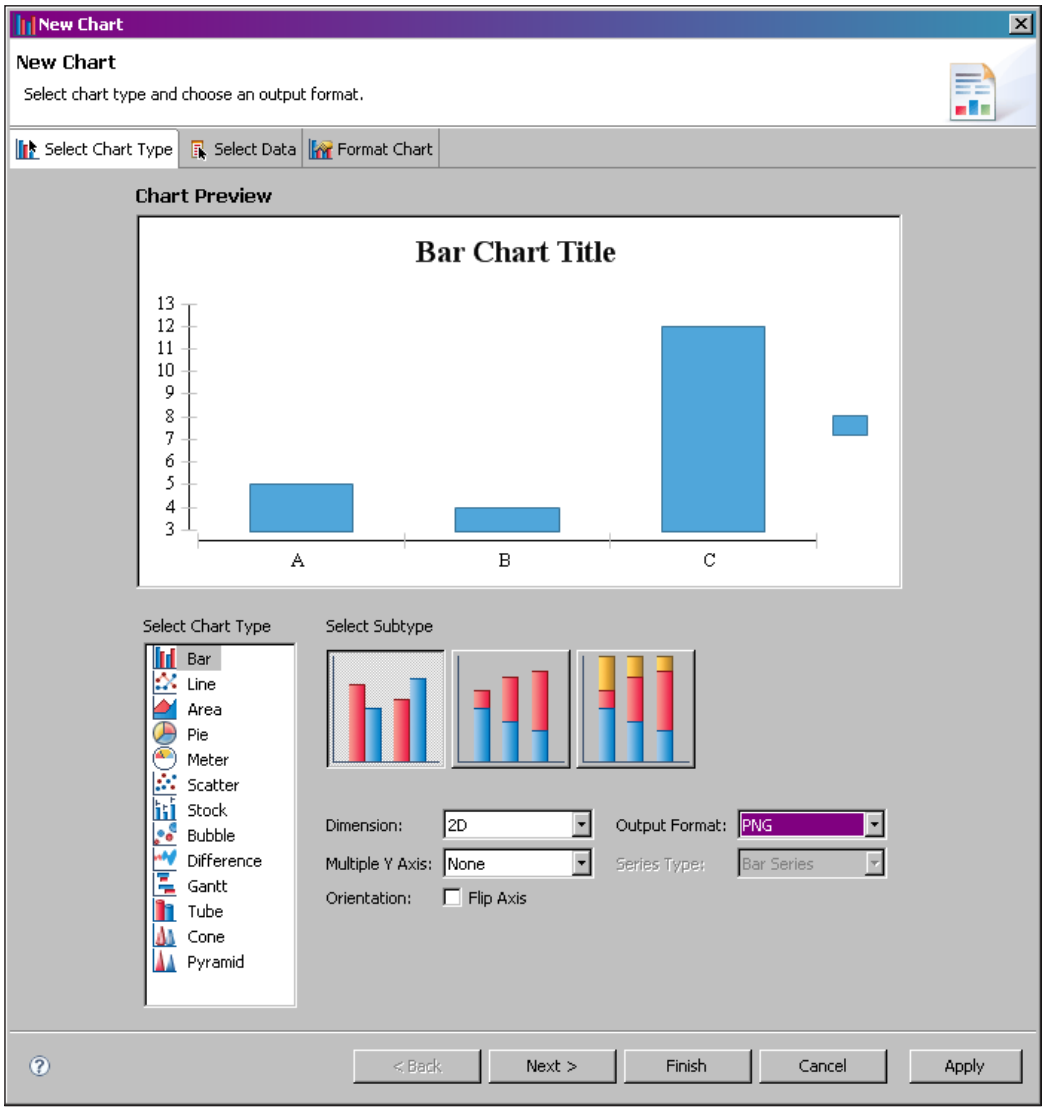
8. In the **Detail** row, apply the `DetailTableRow` style.
9. In the header row, select all the cells and merge them.



10. In to the new merged cell, insert a chart.

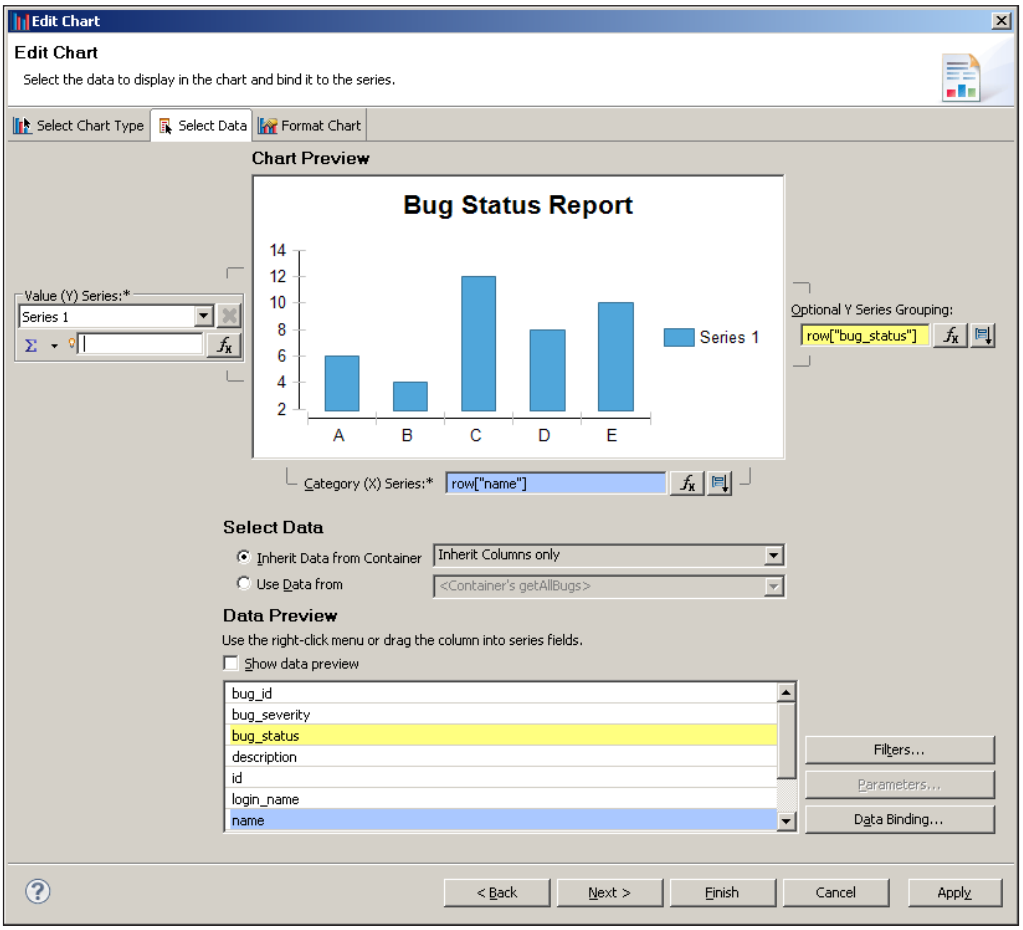


11. Select a Bar Chart and change the **Output Format** to **PNG**.



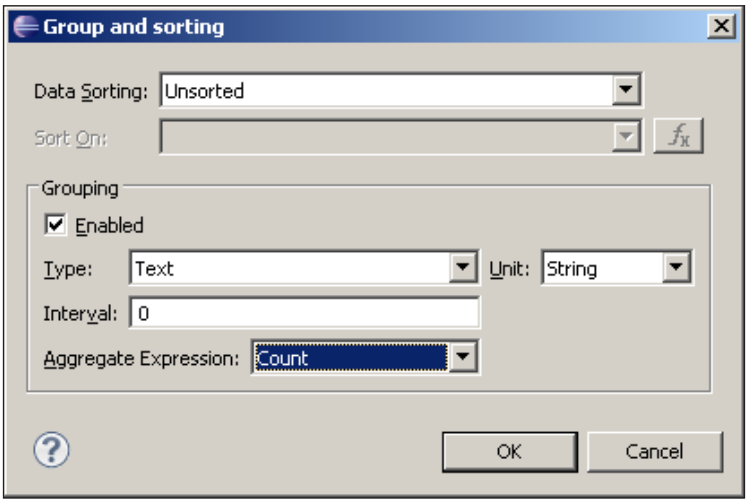
12. Open the **Select Data** tab.
13. Set the **Inherit Data from Container** drop-down list to **Inherit Columns Only**.
14. Drag the `bug_status` field to the **Optional Y Series Grouping** slot.

15. Drag the `name` field to the **Category (X) Series** slot.

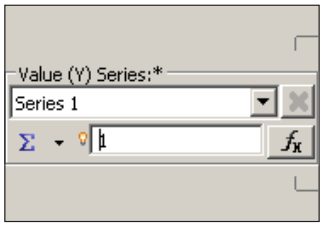


16. Click on the Edit group and sorting button.
17. In the **Group and sorting** dialog, check the **Enabled** checkbox.
18. Set the **Type** to **Text**.
19. Set the **Interval** to **0**.

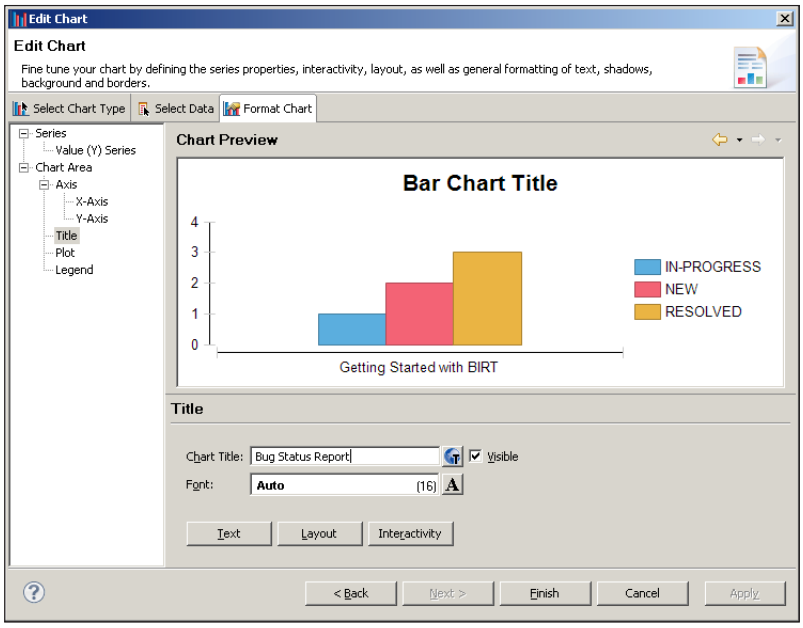
20. Set the **Aggregate Expression** to **Count**.



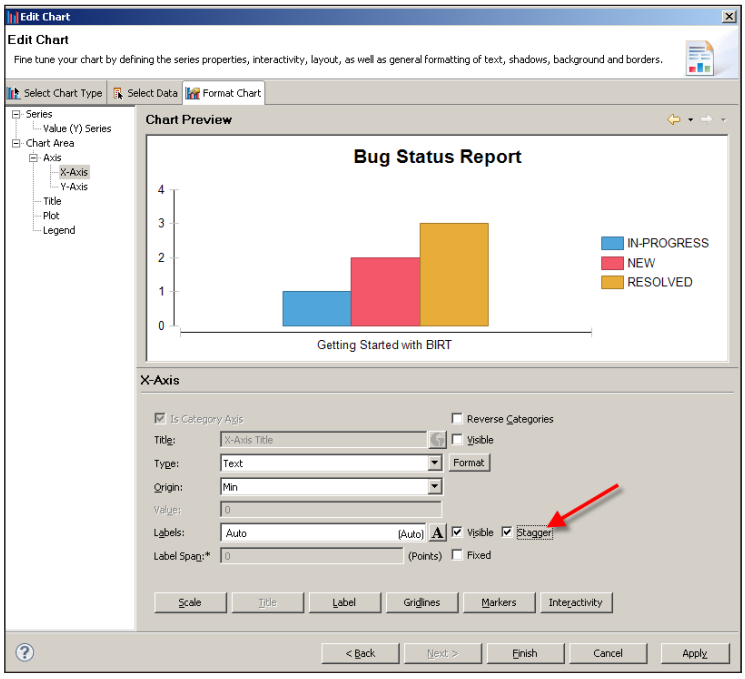
21. As the **Value (Y) Series**, enter **1**.



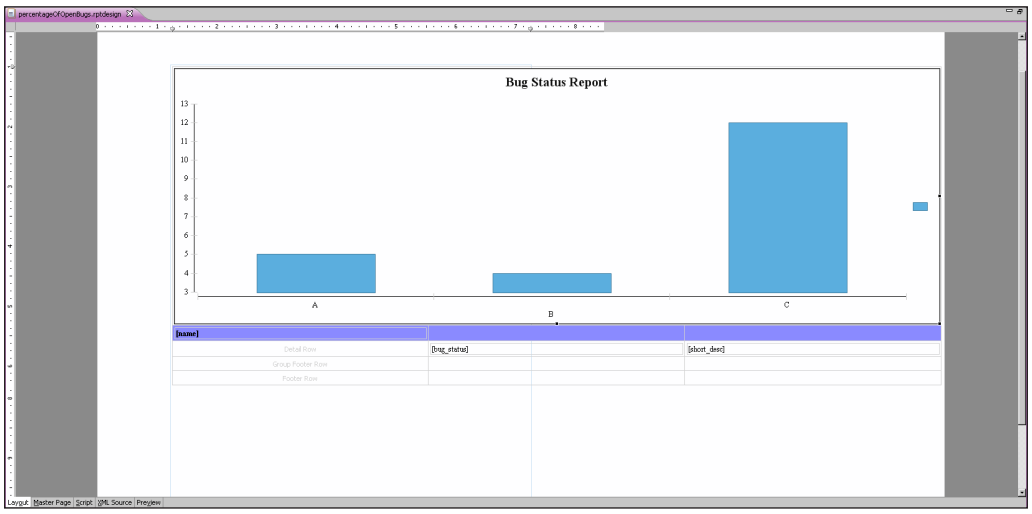
22. Under the **Format Chart** tab, go to **Title**. Enter the title as **Bug Status Report**.



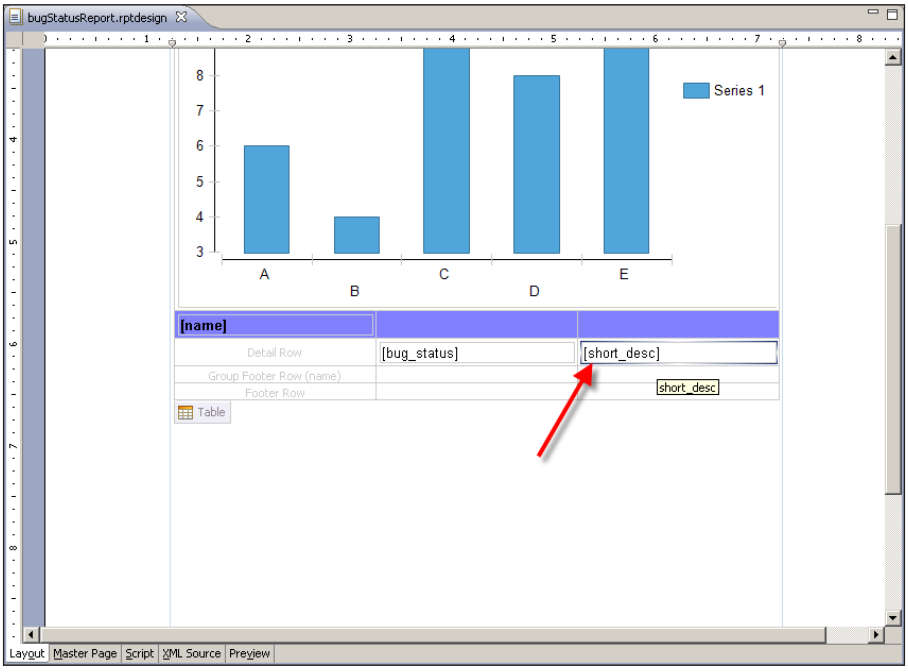
23. Select the **Axis** option.
24. Under **X-Axis**, check the **Stagger** checkbox.



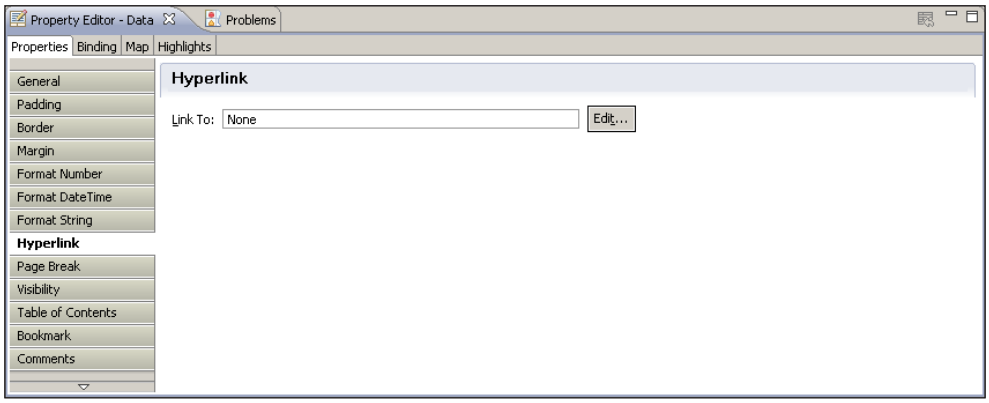
25. Click **Finish**.
26. Resize the chart to fit the number of categories.



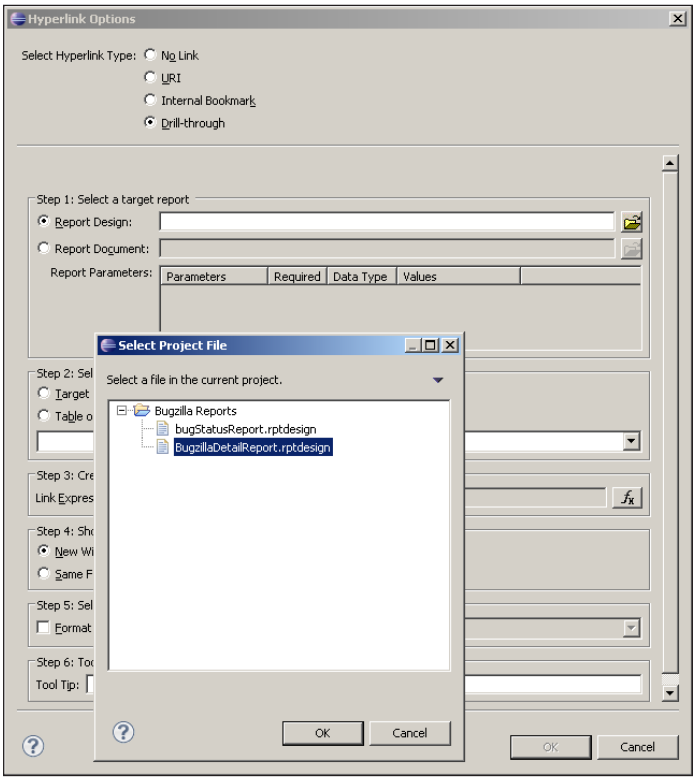
27. The last thing we need to do is add the drill-through from the descriptions to the bug detail. Select the `short_desc` data item in the report designer.



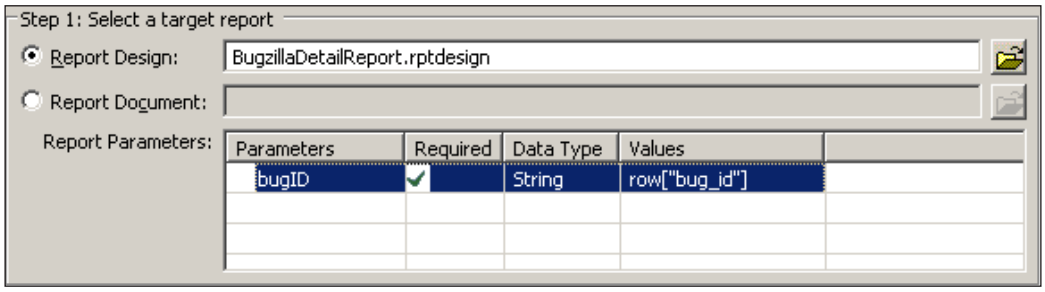
28. Under the **Property Editor**, select the **Hyperlink** tab.



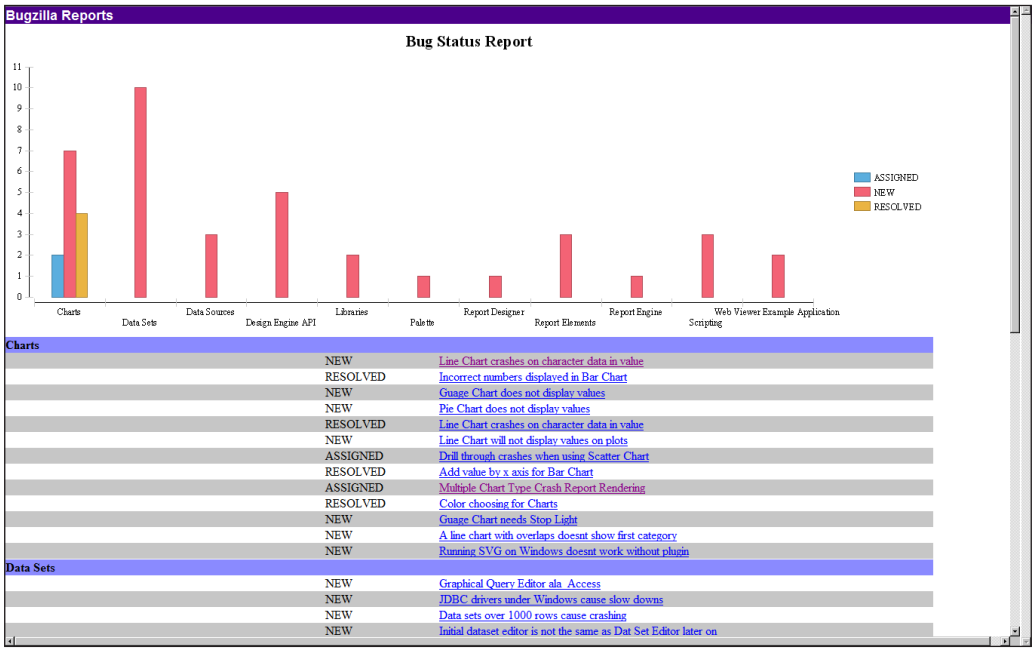
29. Click on the **Edit...** button next to **Link To**.
30. From the **Hyperlink** dialog, select the **Drill-through** as Hyperlink type.
31. Select `BugzillaDetailReport.rptDesign` as the target report.



32. Set up the target report parameter `bugID` to be linked to `row["bug_id"]`.



33. Click **OK** and save the report.



Developer issues reports

The next report is a combination of the last two reports in our list. Given a developer ID, this report will accomplish two things—shows us a Pie Chart showing fixed versus non-fixed bugs and gives us a list of bugs in an open status assigned to that developer. Let's now build the report:

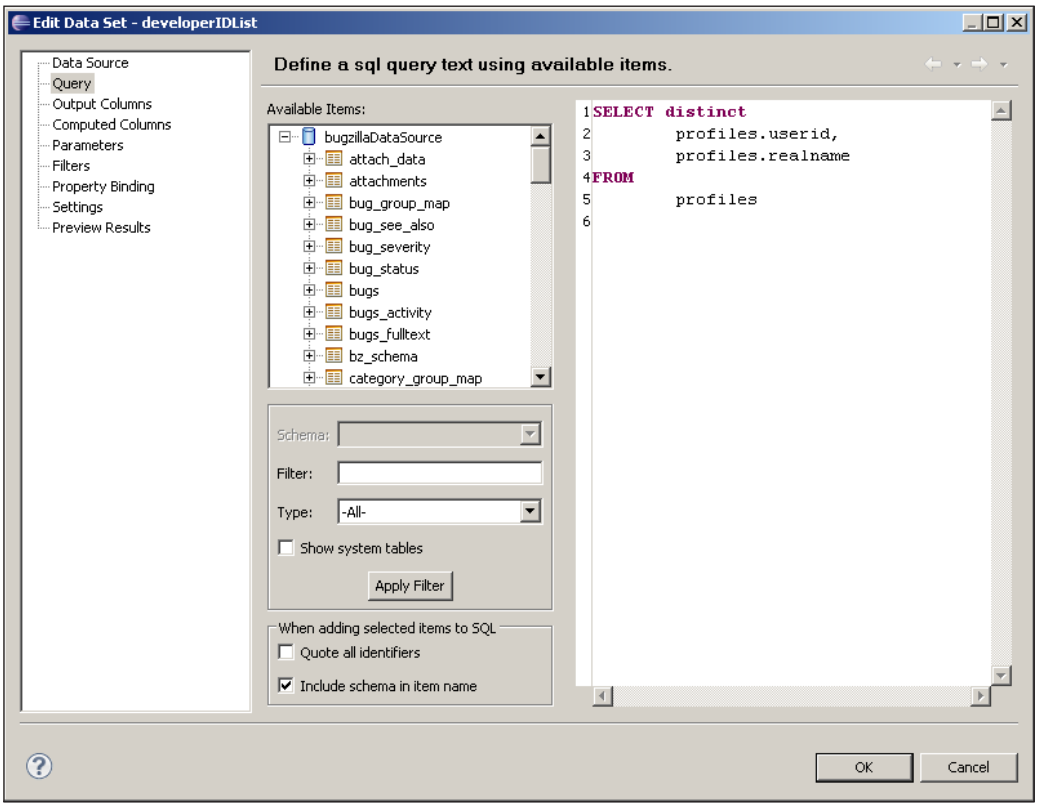
1. Open `BugzillaReportsTemplate.rptTemplate` and save as `DeveloperPerformanceReport.rptDesign`.
2. Modify the getAllBugs query to look like the following:

```
SELECT
        bugs.bug_id,
        bugs.bug_severity,
        bugs.bug_status,
        bugs.short_desc,
        profiles.userid,
        profiles.login_name,
        profiles.realname,
        components.id,
        components.name,
        components.description
FROM
        bugs,
        profiles,
        components
WHERE
        bugs.component_id    = components.id
        AND bugs.assigned_to = profiles.userid
        and profiles.userid = ?
```

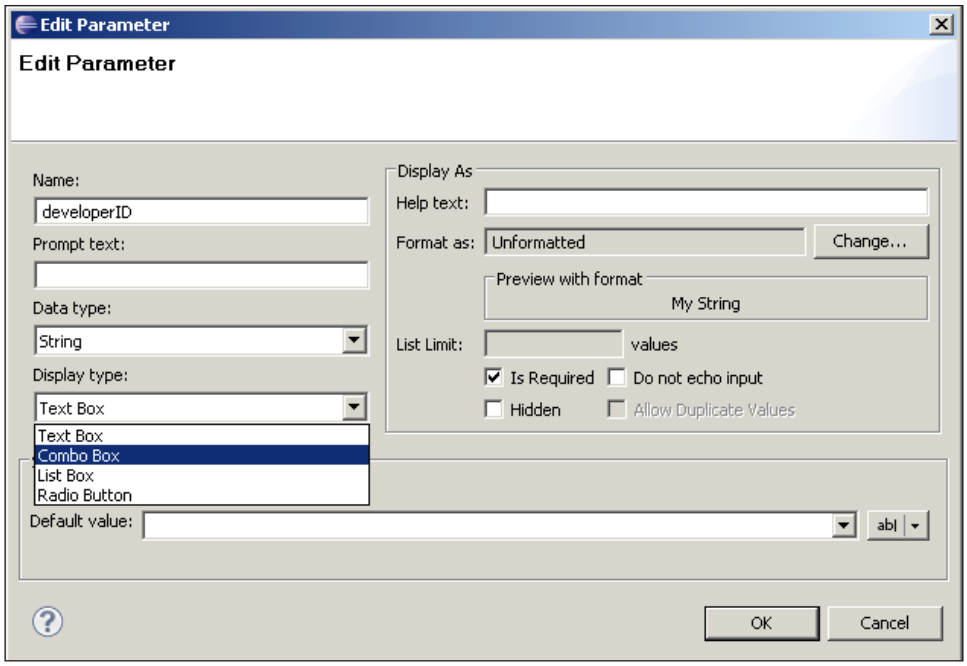
3. Bind the dataset parameter to a report parameter called `developerID`.

4. We want to use a drop-down list for the developer ID parameter. Create a new dataset to display the unique developer IDs using the following query and call it `developerIDList`:

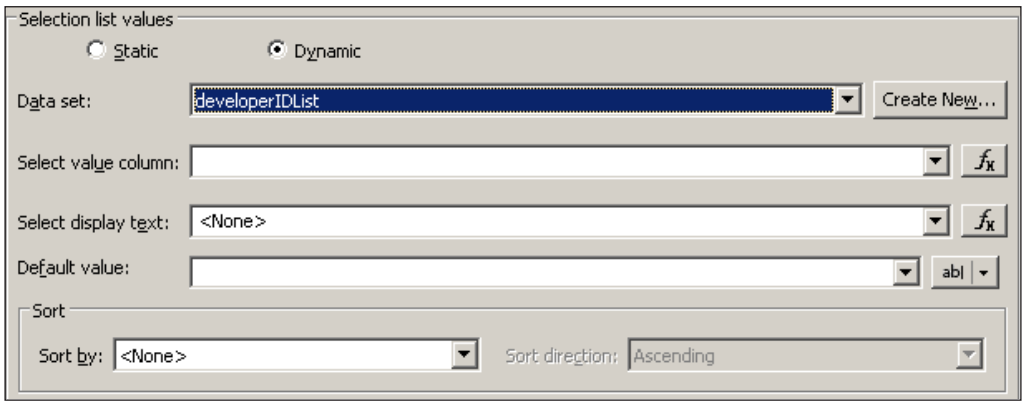
```
SELECT distinct
        profiles.userid,
        profiles.realname
FROM
        profiles
```



5. Open the report parameter `developerID` for editing.
6. Change the **Display type** to **Combo Box**.



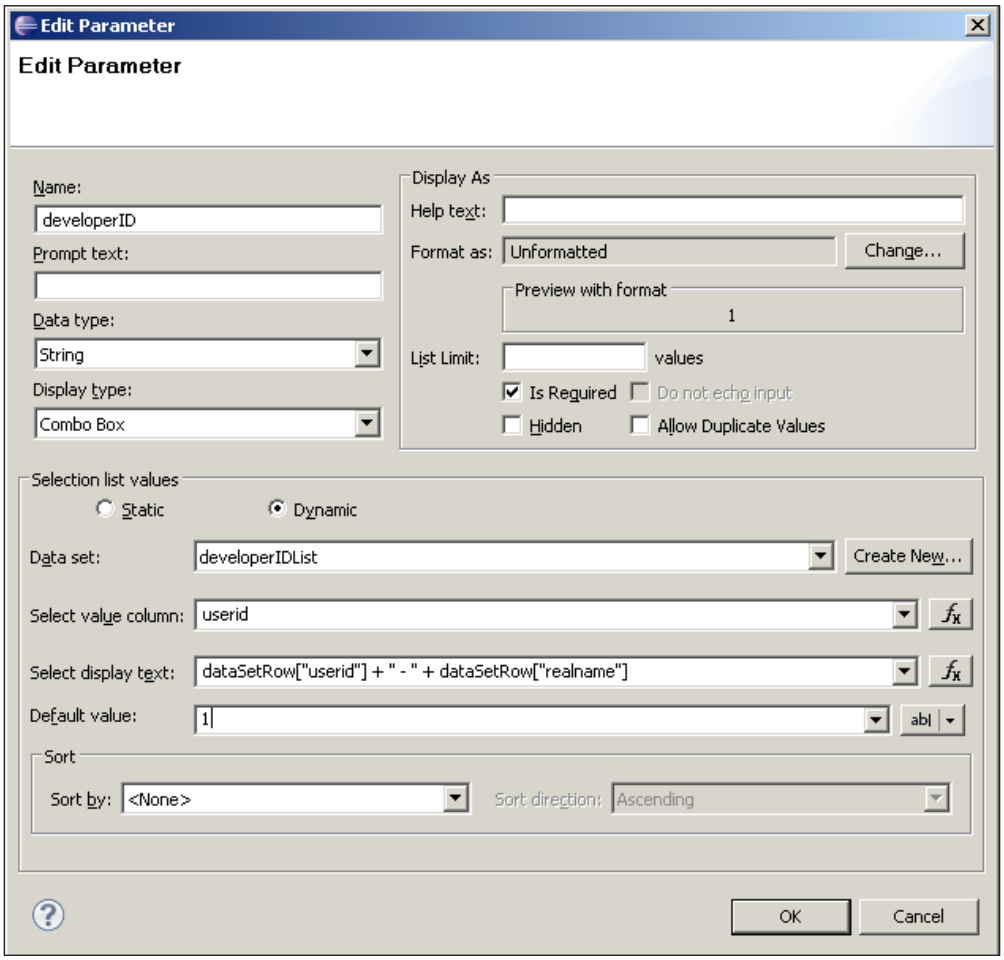
7. For the **Selection list values** option, change the radio buttons to **Dynamic**. Under the list of datasets, select `developerIDList`.



8. For the value column, select `userid`.
9. As the display text, use the following expression:

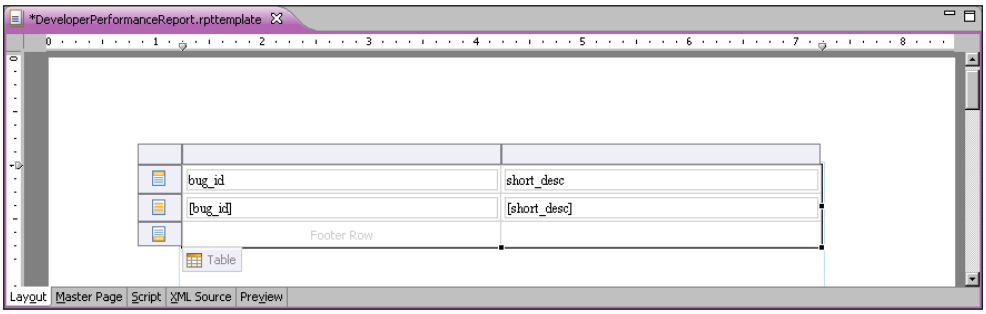
```
dataSetRow["userid"] + " - " + dataSetRow["realname"]
```

10. As the **Default value**, enter **1**. The dialog should look like the following:

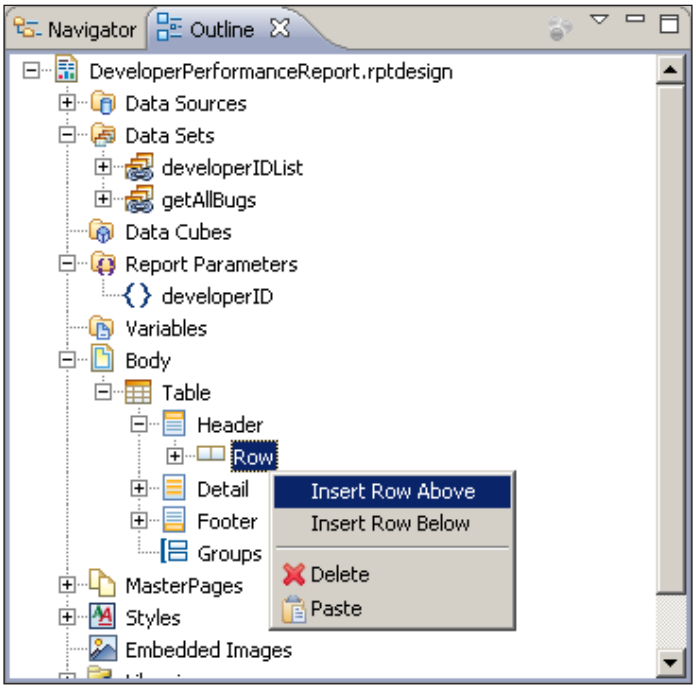


11. Click on **OK** to save our changes.
12. Drag `getAllBugs` dataset over to the Report Designer.

13. Delete all columns except `bug_id` and `short_desc`.

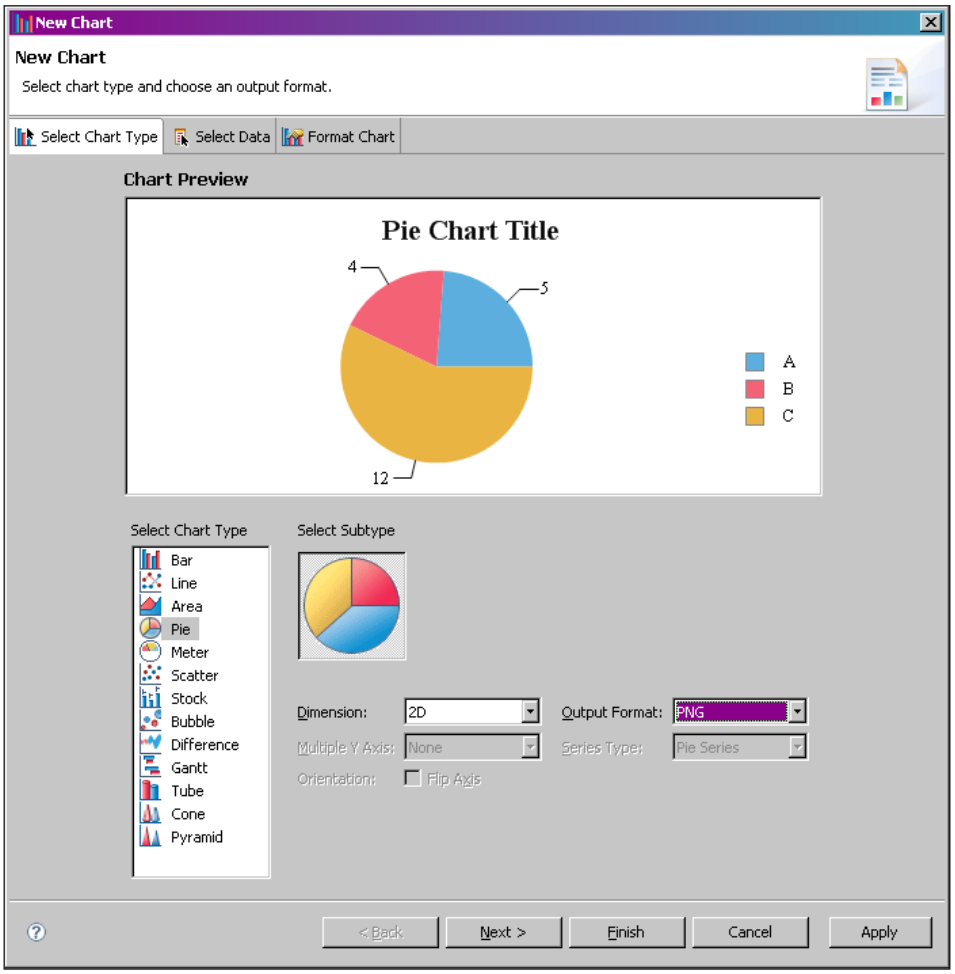


14. Insert a new row in the header above the column labels.



15. Merge all the cells in the new row.
16. Insert a chart into the new large cell.

17. Select a Pie Chart and set the **Output Format** to **PNG**.

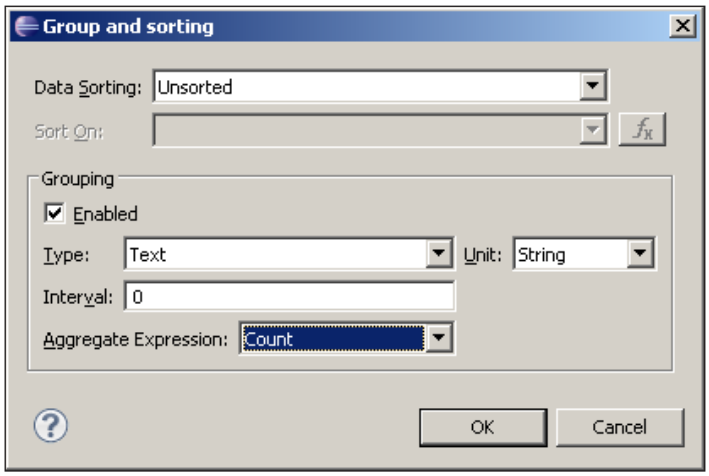


18. Select the **Select Data** tab.
19. Set the **Inherit Data from Container** drop down item to **Inherit Columns only**.

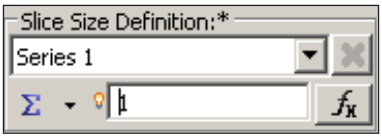
20. Under the Category Definition, use the following expression:

```
if (row["bug_status"].toUpperCase() == "RESOLVED")
      "Fixed";
else
      "Open";
```

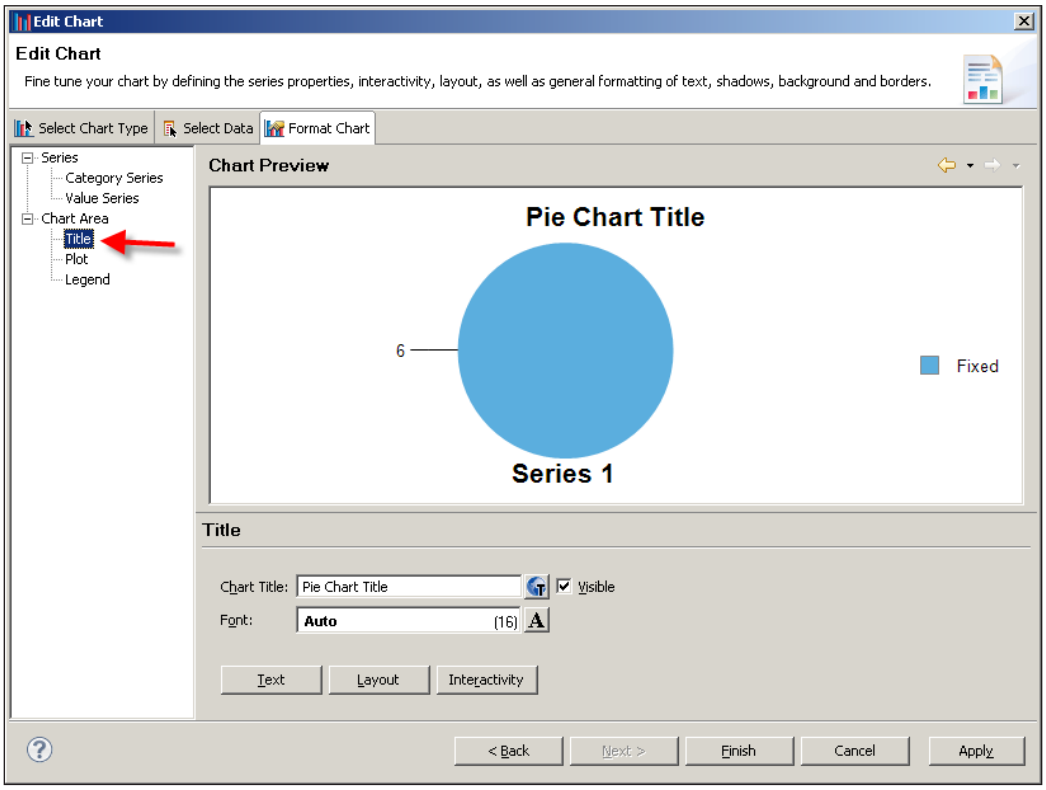
21. Click on the Edit Group and Sorting button.
22. Check the **Enabled** option under **Grouping** tab.
23. Set the **Type** to **Text**.
24. Set the **Interval** to **0**.
25. Set the **Aggregate Expression** to **Count**.



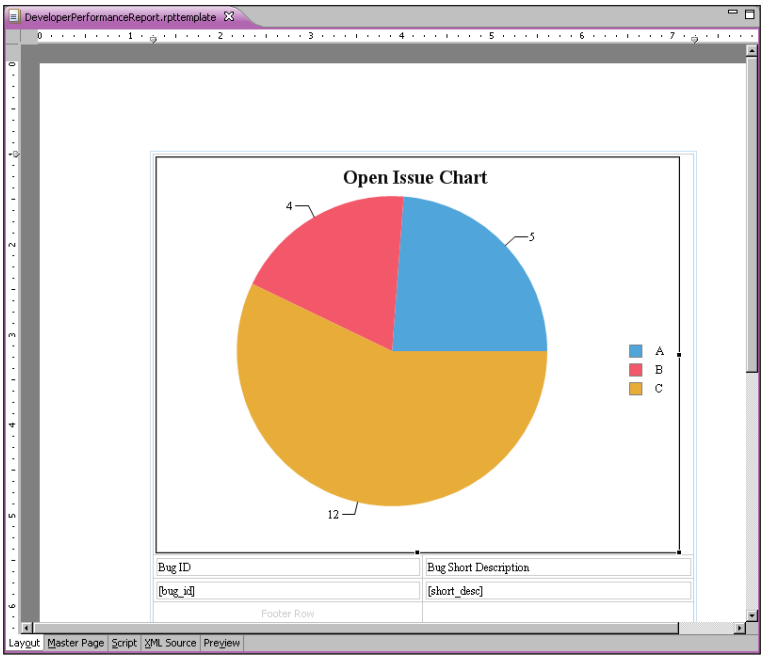
26. Click **OK**.
27. Under the **Slice Size Definition**, enter **1** for the value.



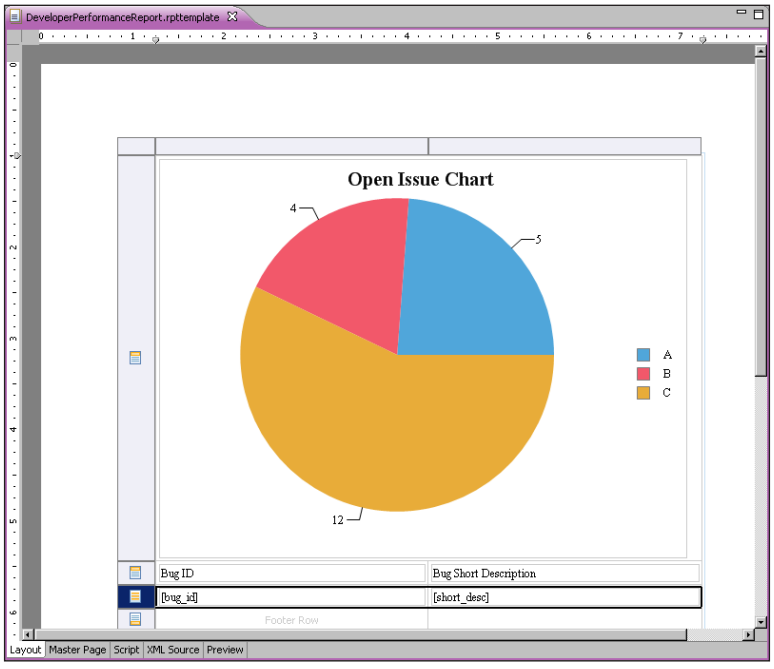
28. Under the **Format Chart** tab, go to **Title**.



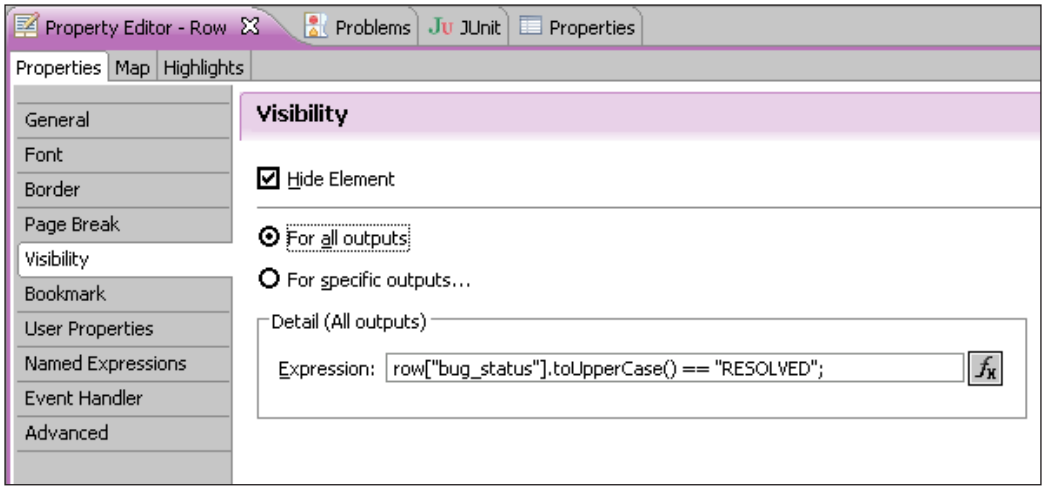
29. Change the **Title** to **Open Issue Chart**.
30. Click **Finish**.
31. Resize the chart.
32. Change the labels for **Bug ID** and **Bug Short Description**.



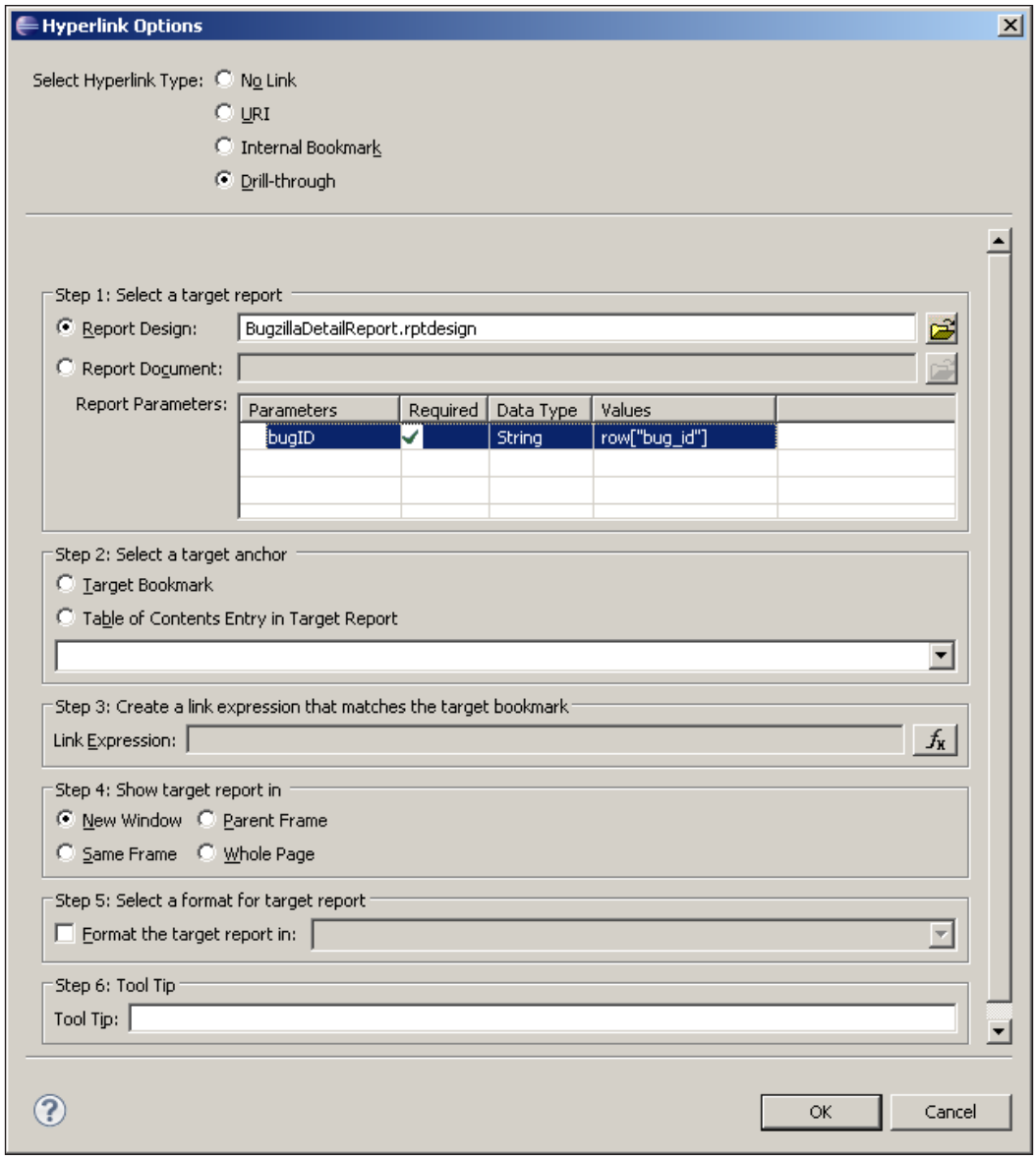
33. Select the **Detail** row of the chart.



34. Under the **Property Editor**, select the **Visibility** tab.
35. Check **Hide Element**.
36. Use the expression, illustrated in the following screenshot:

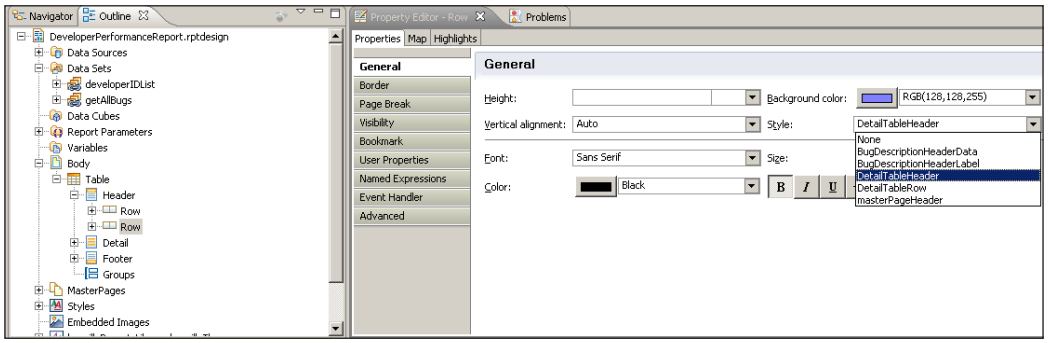


37. Select the `bug_id` data element.
38. Under the **Property Editor**, select the **Hyperlink** tab.
39. Click the **Edit...** button.
40. Create a drill-through to the detail report using the column binding's `bug_id` as the parameter value.

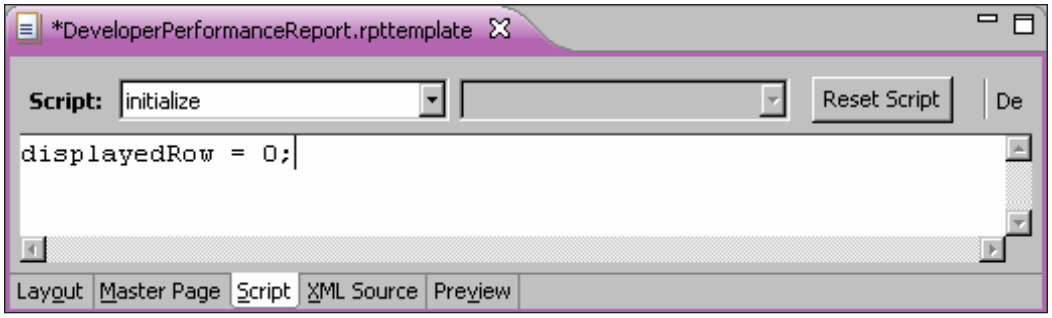


41. Select the root of the report in the **Outline** tab, and apply `bugzillaTheme`.

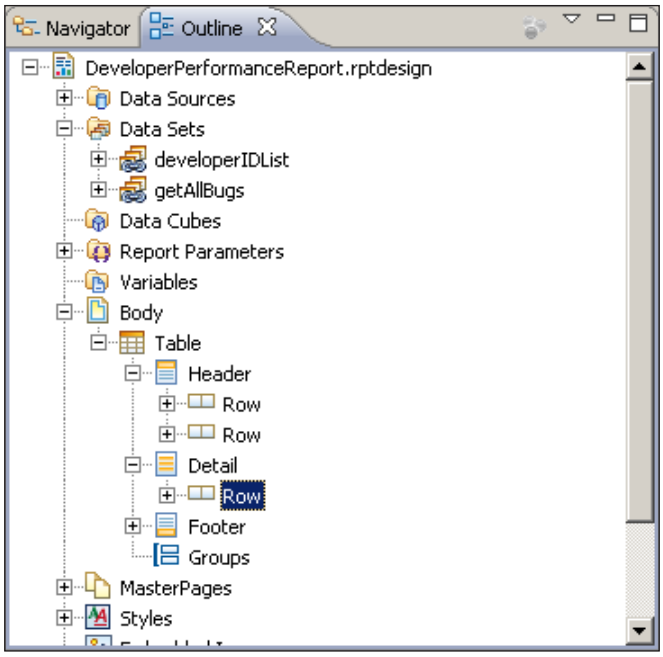
42. Apply `TableHeaderStyle` to the header row.



43. As we used the visibility expression to hide rows that are resolved, this will throw the Highlight used in the `DetailRow` style off. So, we are going to use a little bit of scripting to apply our highlight. Open up the **Outline** view.
44. In the Report Designer, open the **Script** tab.
45. Select the Reports Root element and, in the Script editor, choose the `initialize` method.
46. Type the code shown in the following screenshot in to the Script editor:

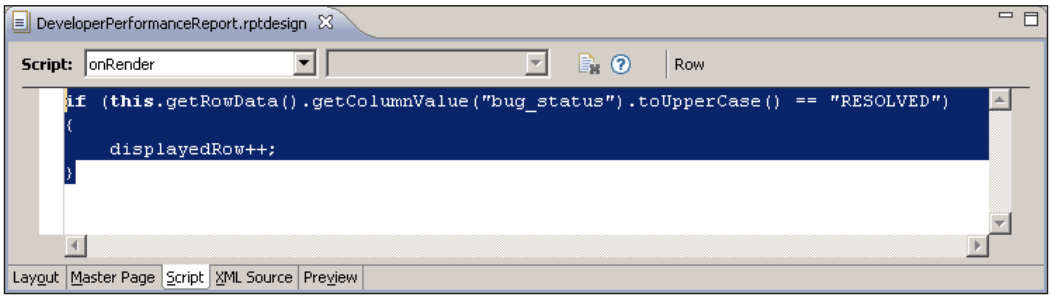


47. Select the **Detail** row in the **Outline** tab.



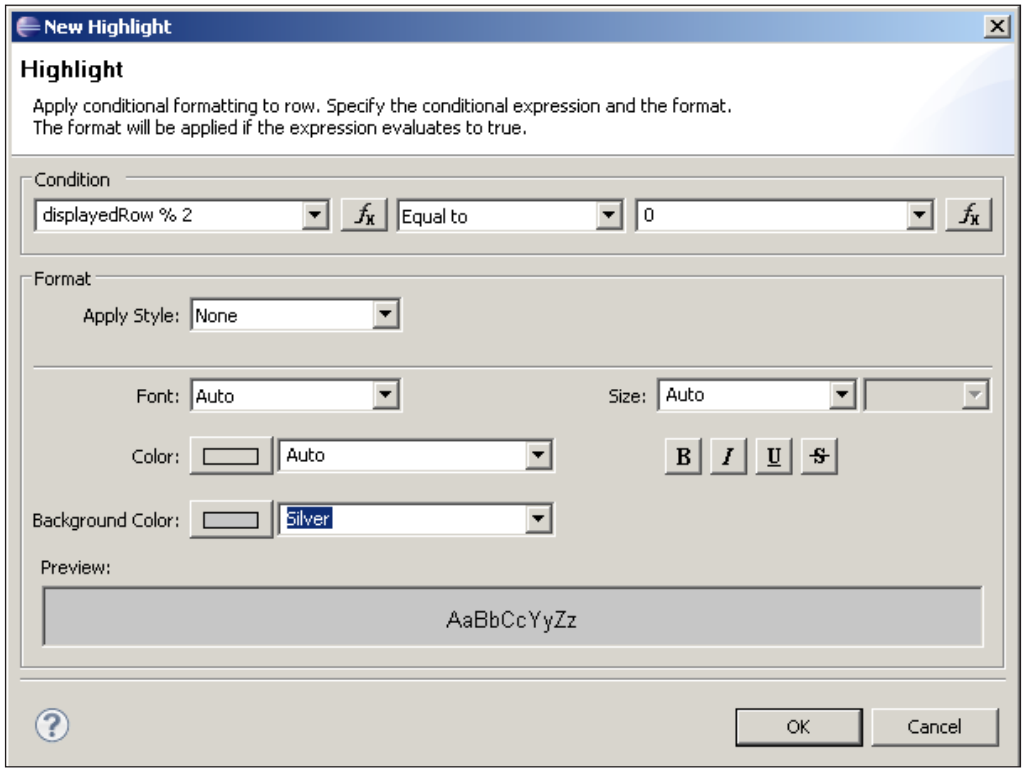
48. In the `onRender` method, type the following code:

```
if (this.getRowData().getColumnValue("bug_status").toUpperCase()
== "RESOLVED")
{
        displayedRow++;
}
```



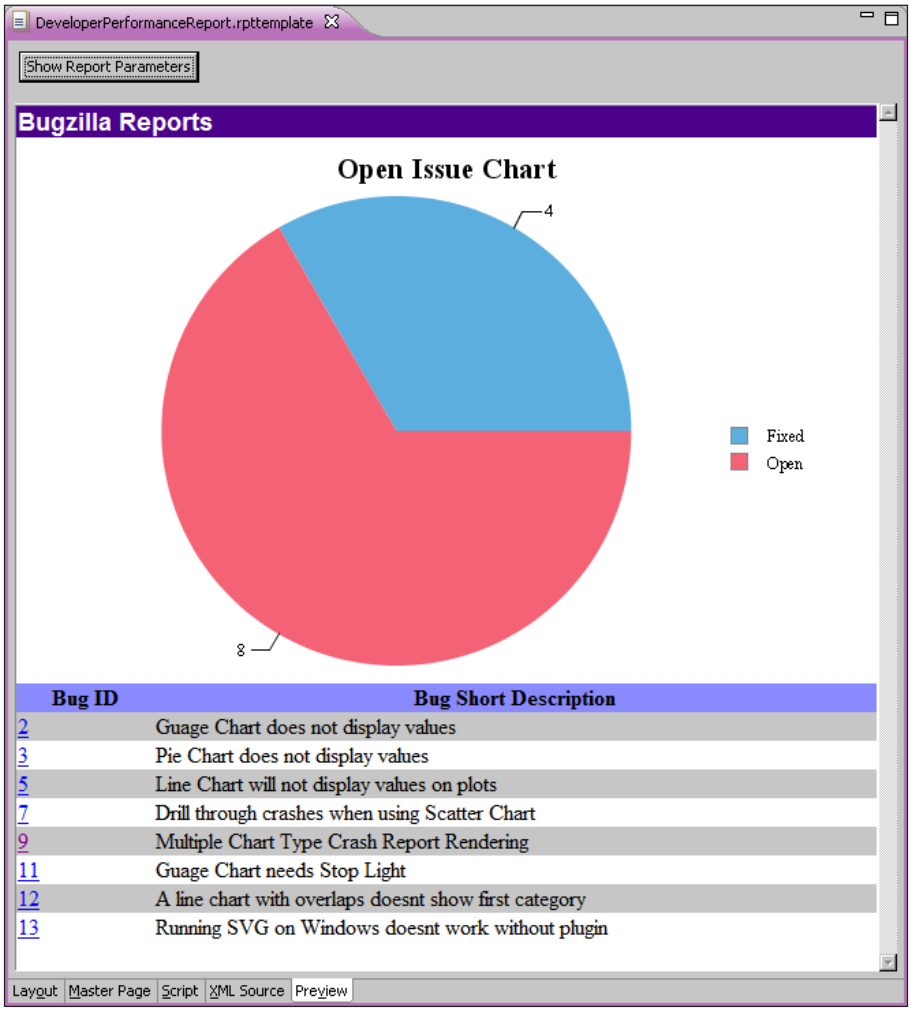
49. In the **Property Editor**, select the **Highlight** tab.
50. Create a Highlight using the following code:

```
displayedRow % 2 instead of row.__rownum
```



51. Save and preview the report.

The reason this worked is because we needed to get the values in the data rows before the visibility rule takes effect. Once we have those components created, we check in the Render phase to see if the value of `bug_status` is set to the `RESOLVED` value. If not, then we advance the counter that would normally be advanced by `ROWNUM`. This keeps it nice and uniform.



Summary

In this chapter we looked at a realistic example of a series of reports that could be used in a real life project. In fact, these requirements were based on some example of report requirements I have been given in the past.

We have illustrated just about every major element of reporting with BIRT, from Report Projects, Report Components, Charts, Scripting, and Formatting. We even looked at how to look at requirements and find reports with similar specs and data, and how to combine reports. If a single report can take the place of multiple reports and tell a more complete story, it is always beneficial to do so. In addition, we also introduced the Visibility rule, which is very similar to the Highlight rule. These reports can now be deployed to a report platform and be used for production reports.

Keep in mind that these reports were created in a controlled environment. Many times, data won't be as clean. For example, I based my assumption for the Developer Performance Report on the fact that there would be only two classes of statuses—open and closed. In reality, Bugzilla keeps this open ended, and there are Open, Resolved, Assigned, a few others out of the box. This is configurable, and there could be any number of other statuses. While in an ideal world this is the responsibility of the Data Managers to maintain, in the real world, report developers' efforts to push back usually result in nothing.

Conclusion

From the beginning we familiarized ourselves with the Eclipse BIRT Design Perspective. We familiarized ourselves with the various BIRT report components. We built some simple reports and some complex reports. We demonstrated how reusing is possible through libraries and templates. We also saw some basic report scripting.

Hopefully by reading this book, you have a good fundamental understanding of how BIRT works and are able to build BIRT reports. The BIRT community is a growing one, and there are number of resources for BIRT. There is a growing user community on the Eclipse website. There are also a few blogs out there. There are a few developers of extensions for BIRT, such as Tribix that makes output extensions.

BIRT is a large and difficult animal to define and comprehend. While on the surface it seems that it is a report development platform; it is also a set of APIs that can be consumed and extended to meet your needs. If the out of the box output formats aren't enough, then custom Emitters can be built. All sorts of things are possible.

Hopefully, now that you have a good foundation in BIRT, you can begin to extend your knowledge of BIRT, depending on your needs.