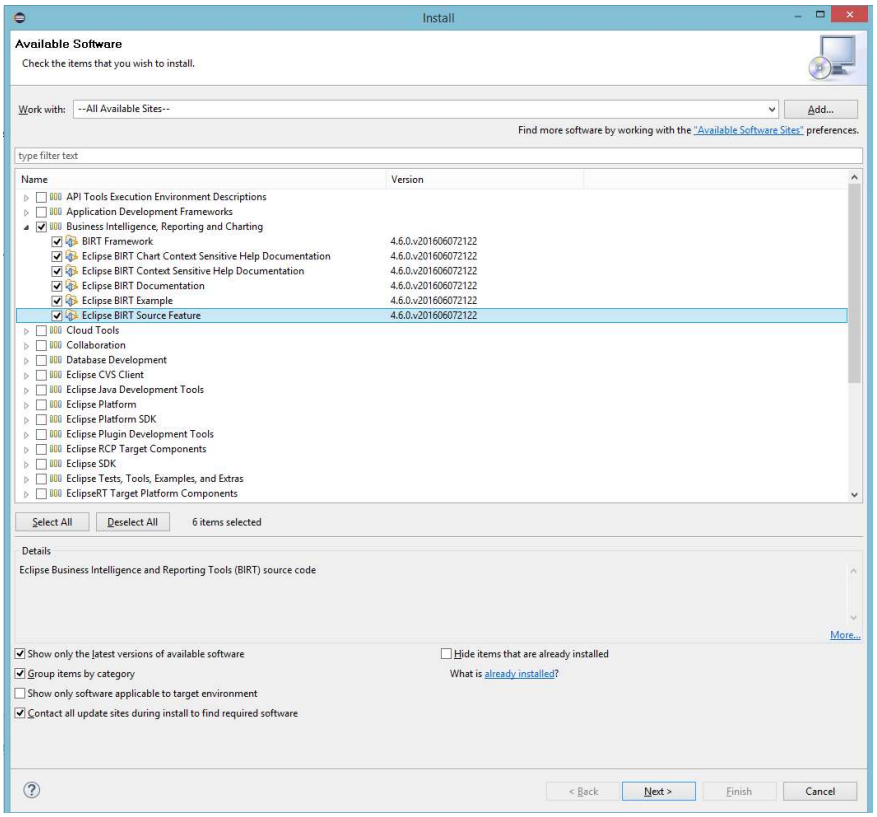
TABLE OF CONTENTS

BIRT

BIRT web resources: http://www.eclipse.org/birt/ for demos, documentation and more. There are probably a lot of other web resources with examples and more on BIRT. Vogella is typically a trustworthy reference for all things Eclipse, although I have not read this one in particular: http://www.vogella.com/tutorials/EclipseBIRT/article.html

INSTALLATION

- Select "Help->Install New Software…"
- Perform the following selection:



- Hit "Next", accept terms and Finish.

AVIX EXAMPLE

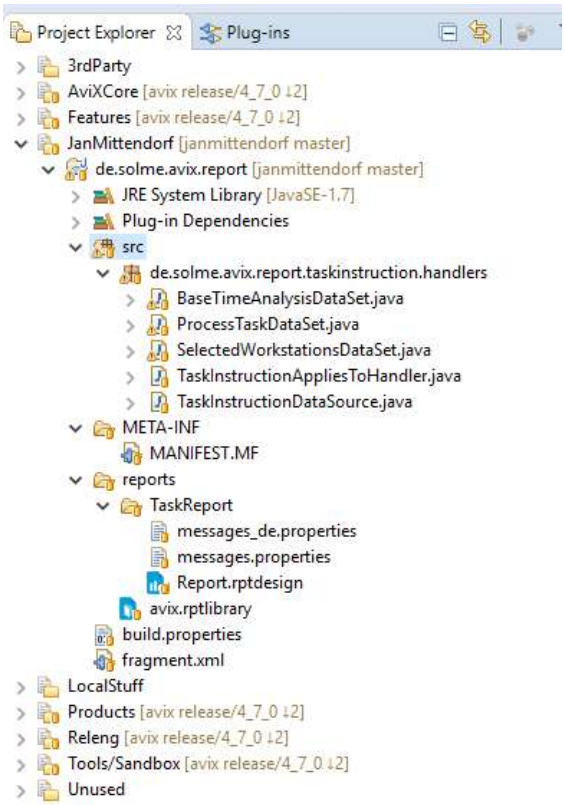
This guide does not intend to describe or explain the fundamentals of BIRT itself. Rather, it will try to explain the way we “hook” a report into AviX. What is needed, essentially, are the following components:

- A plugin (or “plugin fragment”) with your report (could be several reports in the same plugin)
- A BIRT report design residing in this plugin/fragment
- Various java components residing in this plugin/fragment
- A (standard Eclipse) extension onto an AviX API extension point called “se.solme.avix.reporting.birt.birtReports”. This can be seen as the registration of your report, making it available in AviX when running/debugging.

The subsections below will make use of an example called “TaskReport”. It is a copy of a standard report existing in AviX.

TASK REPORT – PLUGIN OVERVIEW

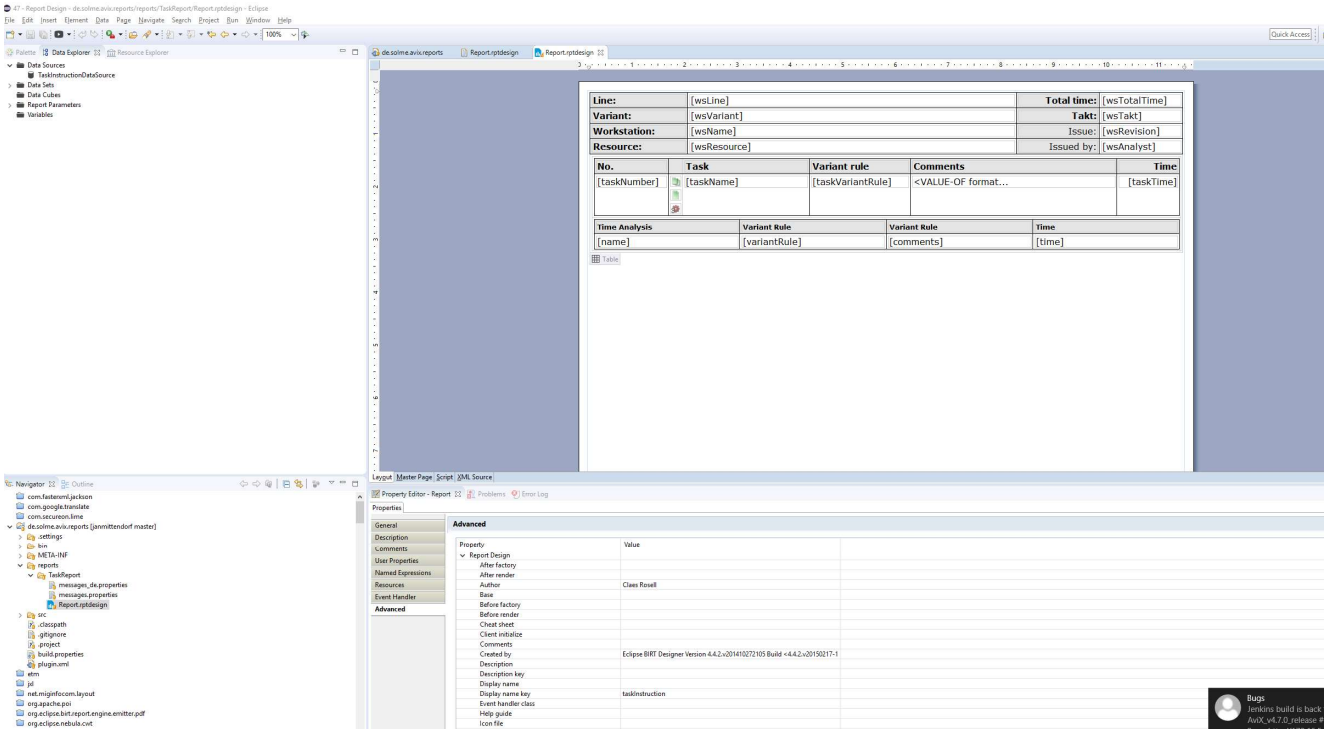
In the image below, there is one plugin fragment called “de.solme.avix.report”.



This fragment actually contains all of the components described above, and we will go through them step by step.

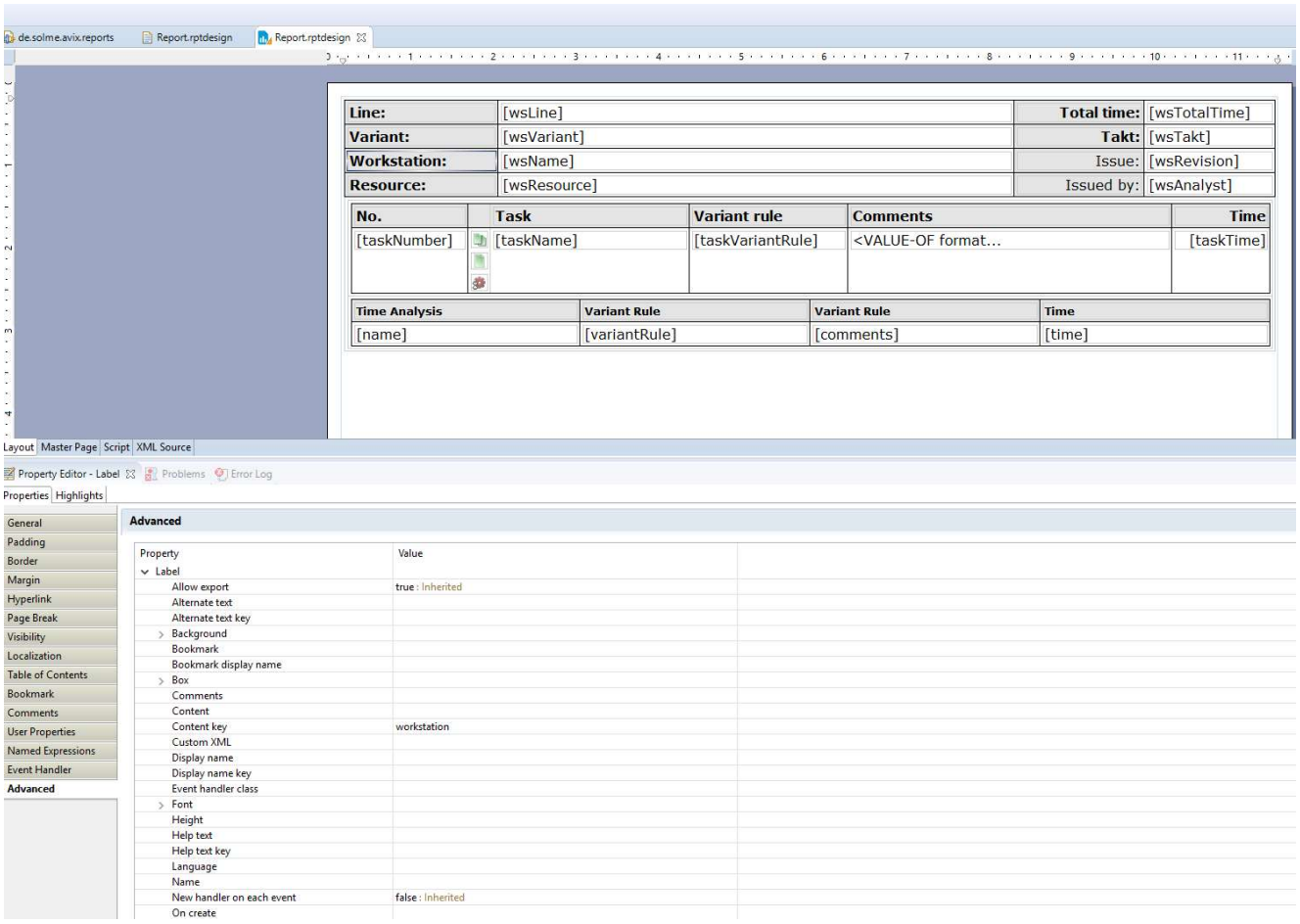
Report design – how to open

Once one has BIRT installed, the “Report Design” Eclipse perspective is available. Open it, and find the “Report.design” in the Navigator as seen in the image below. Open it (if it will not open the Report Editor by default on double-click: right-click and open with the “Report Editor”).



Localized strings in Report design

A first note regarding the “*localized strings*” is worth a mention. Immediately on the “Layout” page, a bunch of localized strings are present. “Line:”, “Variant:”, “Workstation:” etc. (You may see them already in your locale language, for instance German). If I select the cell containing “Line:” and then look in the Properties view at the bottom, I see that the so-called NLS-key for this is called “workstation:”



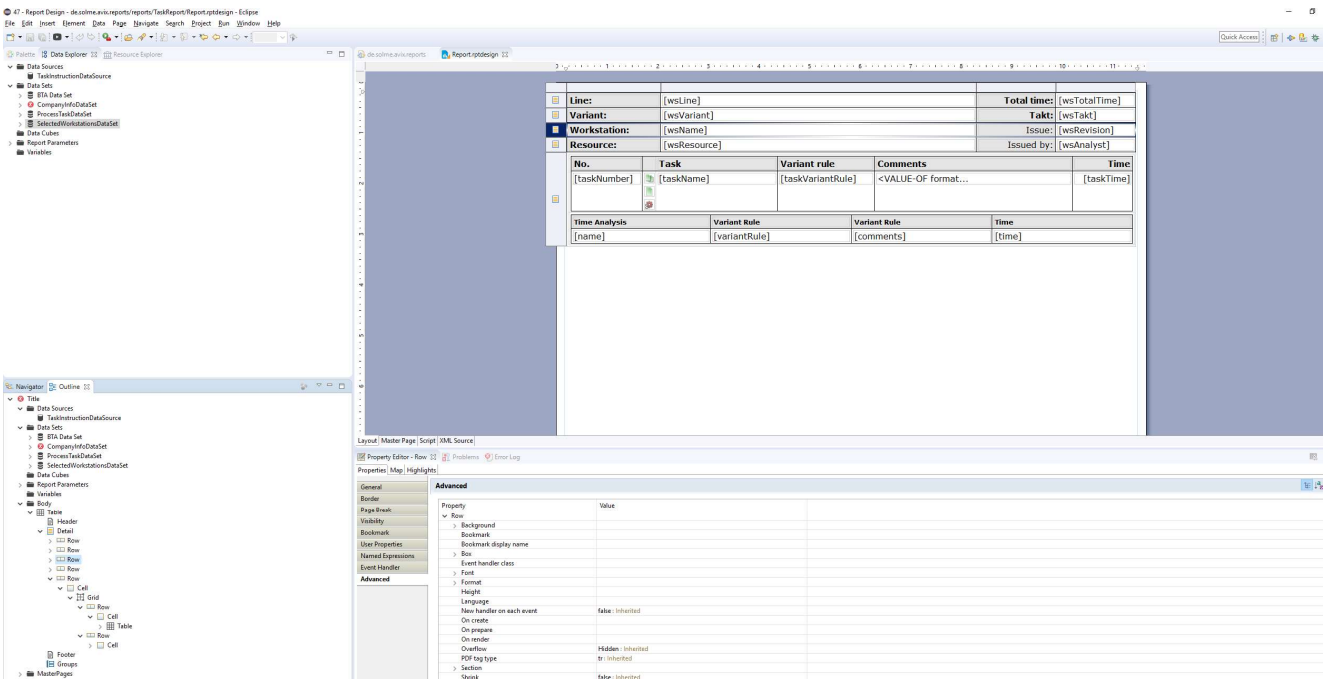
Ok, so what is this telling me? Well, adjacent to the Report.design, there are two standard java property files: “messages.properties” (English strings) and “messages_de.properties” (German strings). At least the English properties file should contain a key/value pair with the key “workstation” and the value “Workstation:”. This value is inserted both in the the Report design editor, and of course also in the printed report itself.

Also see the “Master page”. Similarly, the name of the report itself can be given via a localized string key. (Later on, try changing the values for the key “taskInstruction”, and you will see that the report name will change.)

Final word on localization: there is also a dedicated tab in the “Properties” called “Localization”. That is probably where one should manage string localization, rather than on the “Advanced” tab as I did in the screenshot above.

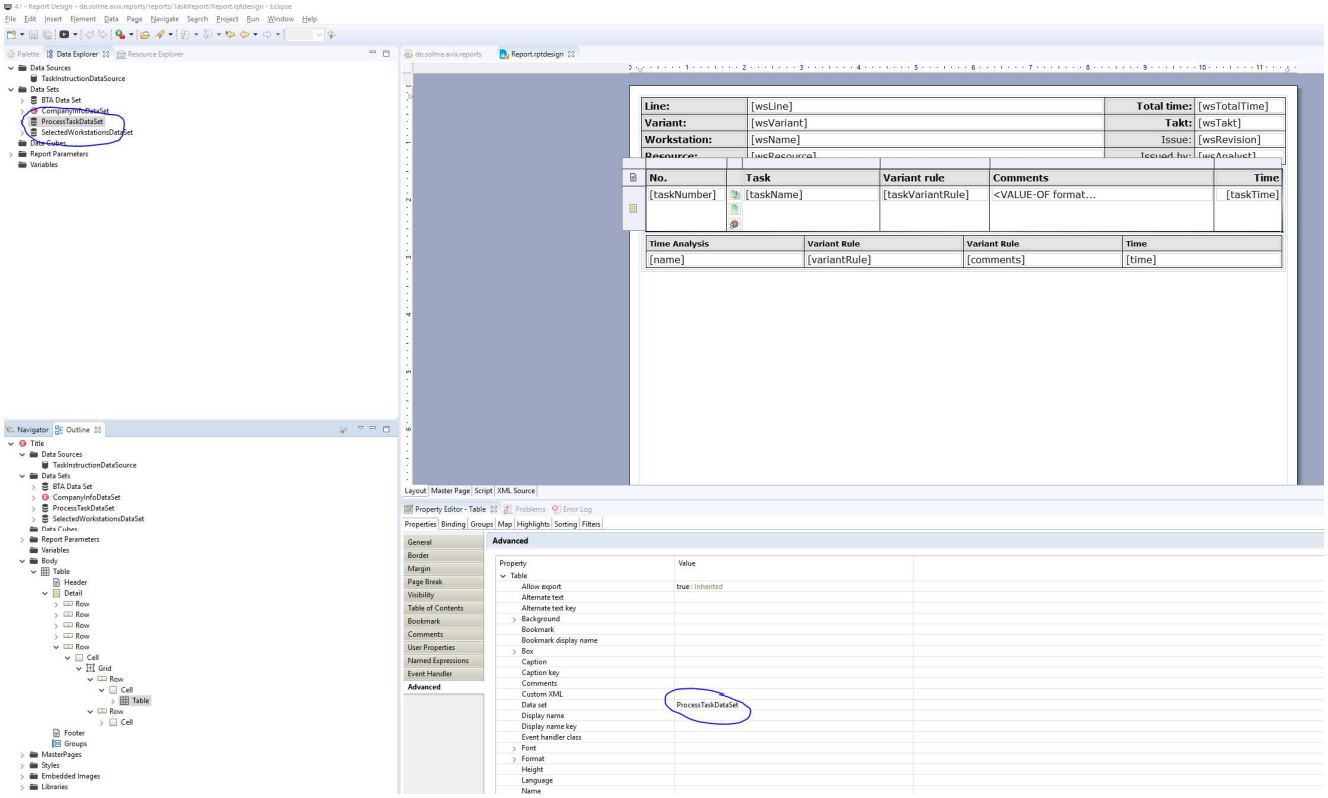
Report design anatomy, hooks to java handler components

Head over to the “Outline” view of the BIRT report design perspective. This is really great, since it enables me to “drill-down” in the layout elements. For instance, when I have expanded the “Body”, the outer “Table” and selected a certain “Row” in the “Detail” of that Table, it will also select that very row in the design editor:



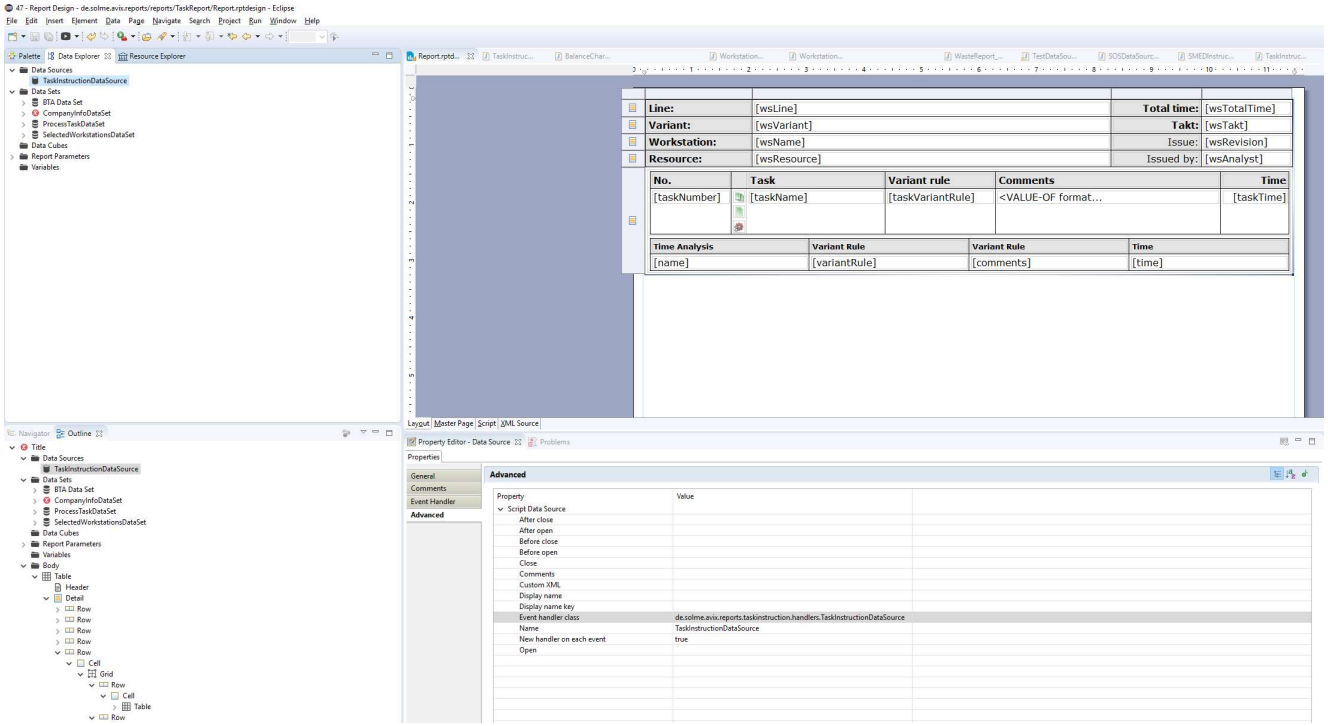
In general, I think that the views “Data Explorer”, “Outline” and the “Properties” are essential to have open at all times. They react on selection and update accordingly. You may select a Row in the visual editor, and the views will display relevant information.

Table elements are crucial: here one connect the so-called **DataSet** elements. In this screenshot, I have selected the innermost Table. My blue “circles” highlight the things of importance for the next section.



Data Source

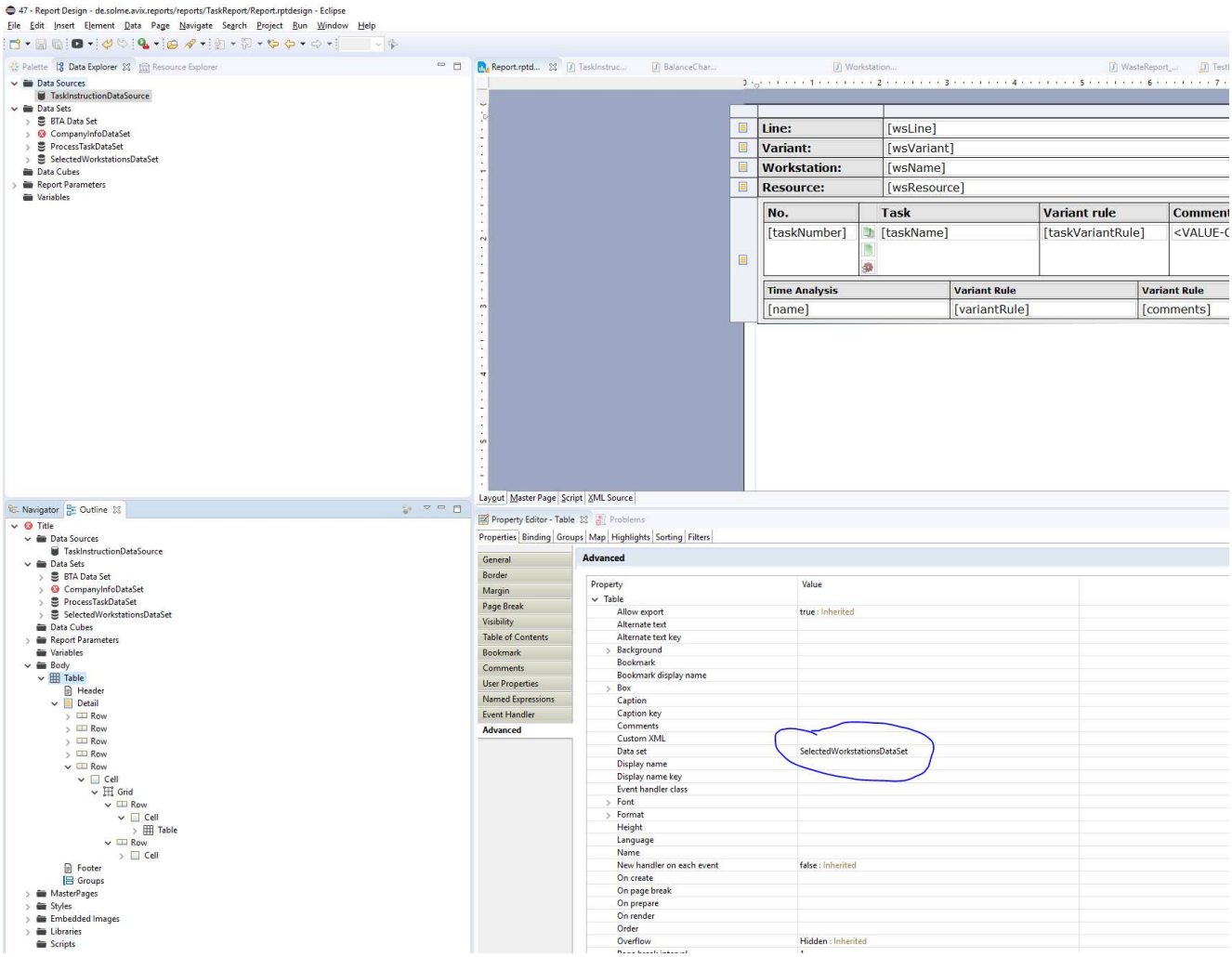
One always needs a data source. All the data sets refer a data source. In our case, it is the “TaskInstructionDataSource”:



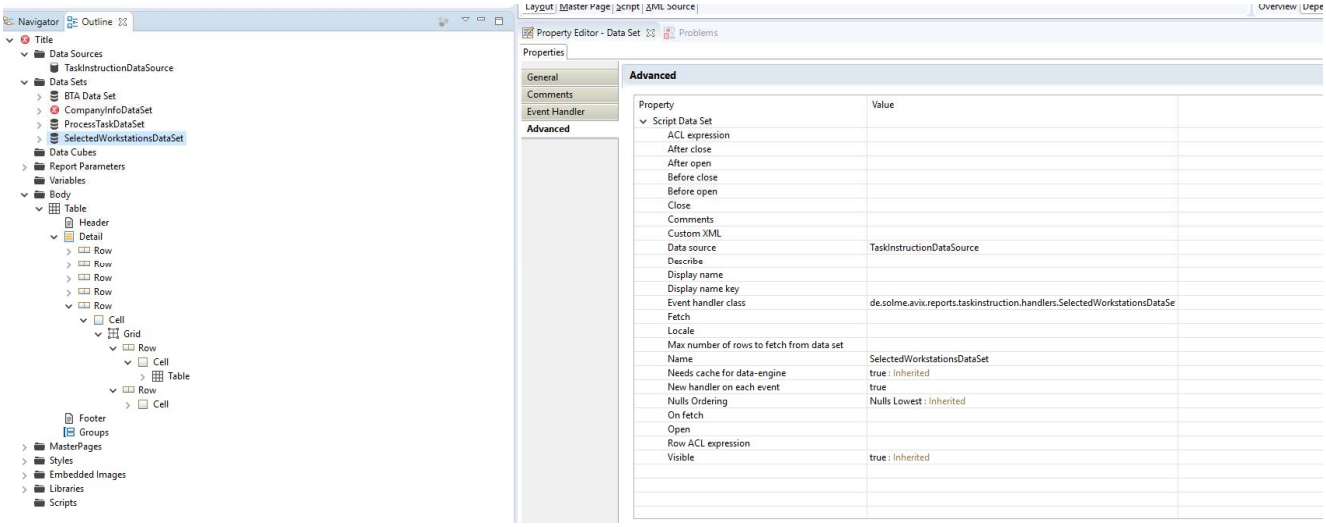
See the “Event handler class” in the Properties (you can also look in the “Event handler” tab there). Here is our first java component being referred: the class “de.solme.avix.reports.taskinstruction.handlers.TaskInstructionDataSource”. If you open that class, there is not much there. But it needs exist and be referred!

Data Sources

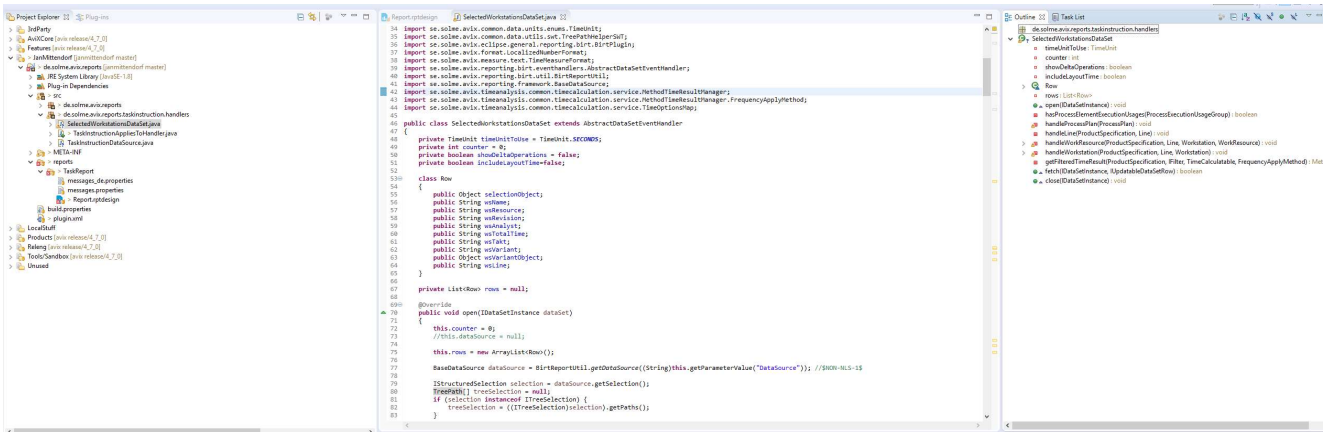
Typically, the Table elements make use of data sources. Let’s inspect the outer Table element here. It refers to a Data set named “SelectedWorkstationsDataSet”:



So what's this "SelectedWorkstationsDataSet"? Well, have a look under "Data Sets" in either the Data Explorer or Outline. Open it:



As you can see, once again we have a java component reference here. Let's open the source code for it:



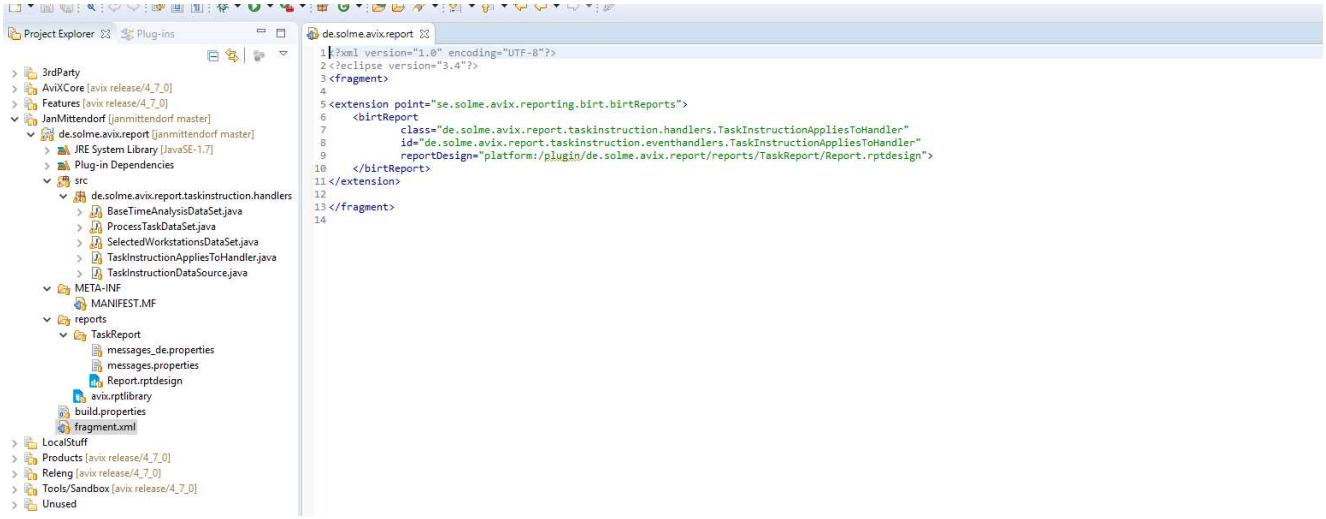
This code for sure is more fleshed out. Also, the methods of this class will be invoked by the BIRT engine when actually running the report. *When debugging, make sure to insert debug breakpoints to see what the engine is really doing*! Also, **this is where core functions of AviX the AviX API is being called**, to get objects and/or structures needed in order to fill the report.

The other AviX-related Data source of this report are "ProcessTaskDataSet" and "BTA Data Set". Please see the connection to the java components here too.

Register your report

Assuming that the report is at least partially ready to be tested in debug mode, there is one final thing we need to do.

Open up the "fragment.xml" (could also be a "plugin.xml", if this was a plugin rather than a plugin fragment):

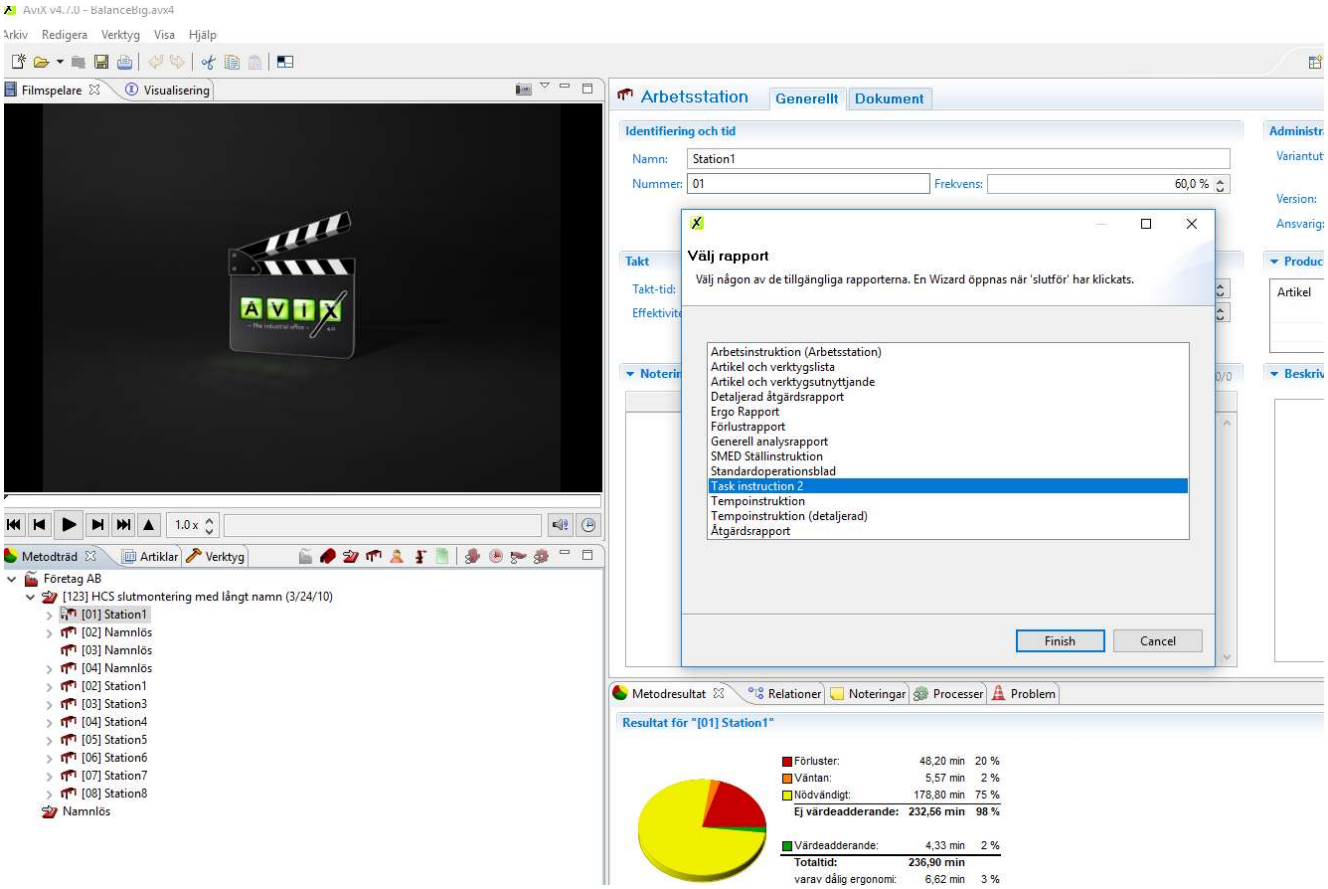


This represents the actual registration of the report, thus making it available within a running/debugged AviX. A few comments:

- The “reportDesign” element needs to refer the RPT design element
- The “class” needs to refer the final java component we need – the so-called “applies to”-handler. This component will be called for “testing” whenever an object is selected within AviX. It is simply there to test for which selections the report should be printable! (Of course, this requires some basic knowledge about the AviX data model). Suffice to say in this example, this report is printable on a multi selection, AND that any selected element must be of one of the types: Line, Workstation, WorkResource and ProcessPlan.
- The “id” needs to be unique. Typically, use the same as for the “class” element.

Test your report

Start AviX in debug. Make sure to have some data, and select say a Workstation (if the report is printable on Workstation level). Click the print button. Your report should be among the selectable reports:



WHERE TO GO FROM HERE?

First, I suggest tampering and playing around with the example report and test things out. Change simple strings, add things, and try out the “binding” things. (See a “Table” element, open its “Binding” tab in the Properties view and have a look. See the different things there, for instance “wsName”. Note that this is being used in a certain Cell. Also note that in the data set “SelectedWorkstationsDataSet”, and in the java component related to it, there are things going on).

Once you feel that you are somewhat getting the hang of things – create a new Report from scratch. Remember, you always have the existing report as a reference in case you wonder how things work.