An introduction to the

Pentaho C*Tools

Cees van Kemenade

pentaho@vinzi.nl

August 2011

Contents

1. Introduction

This document is a quick introduction to the CTools. It covers the basic concepts of the suite and shows you the potential pitfalls when using this suite. This guide is not meant to be a comprehensive tutorial or a reference guide. Therefore it won't cover many interesting features of the CTools. However, when you finished this guide:

- You understand the background of CTools,
- You are able to install the CTools, and
- you have seen some of the basic features of CTools in action.

In order to make this manual easier to read we introduce the two markers. If you notice the caution-sign, shown here in the left margin of this paragraph, then it is worth reading that paragraph slightly more careful. Each product has its own special features (sometimes also called quirks) of the CDE that you need to be aware of as a beginner, and that feel normal when you are used to it.

The second sign is used to mark paragraphs containing important hints to make usage of the CTools more convenient or the dashboards easier to maintain. Use these hints to your own benefit if you like them.

After laying this foundation you will be ready to dive into the more complex aspects of the suite yourself. We also advise you to consider the digital CDF-tutorial, which covers ten showcases (from basic to advanced) on using the CDF components to create dashboards. Basically this tutorial focuses on code examples of CDF. However, understanding how CDF operates under the hood gives you a much better understanding of how to use CDE, as CDE is the editor that discloses CDF and the other CTools. The CDF-tutorial is run in the bi-server and source code of the examples is provided. More information on this tutorial can be obtained from www.webdetails.pt.

If you want a good introduction into business intelligence and the Pentaho-tools I recommend the book:

> Pentaho Solutions: Business Intelligence and datawarehousing with pentaho and mysql
> Roland Bouman & Jos van Dongen
> Wiley, 2009

This book provides you with a good introduction to the field and it also gives a detailed description on how to install the pentaho-tools and how to configure them for use with mysql[1]. However, this book does not cover the CTools.

The pre-release of this document was distributed as 'An introduction to the Pentaho C** tools' in February 2011. If you have suggestions for the authors you can contact me at pentaho@vinzi.nl. If you have questions or suggestions regarding the application you can contact the core development-team of the CTools at cdf@webdetails.pt.

[1] Mysql is used as an example. However, these instructions can be extended to several other database systems.

2. Goals and background of the CTools

The CTools aim to deliver an easy framework to create and maintain dashboards for a community edition Pentaho business intelligence server (bi-server). It delivers a set of tools to design a set of interactive dashboards that are closely integrated with the bi-server: All data available within the bi-server are available for display in a dashboard and the security roles of the bi-server are applied when displaying the dashboards. The CTools, or actually tool-sets, are:

1. Community Dashboards Framework (CDF)
2. Community Chart Components (CCC)
3. Community Data Access (CDA)
4. Community Dashboard Editor (CDE)

Futhermore the CDE provides access to other component libraries of CDF, such as Open Flash Charts, JFreeChart, xaction components etc.

Next we introduce the four core libraries of the CTools briefly.

2a.Community Dashboard Framework (CDF)

The CDF is the engine that generates the interactive dashboards and makes these dashboards available through the user console of the bi-server. Dashboards are basically constructed by creating a set of (visual and non-visual) components and connecting them. A component is just a simple JavaScript object that encapsulate all the object properties and behaviors. These components are very flexible in as:

- Component interact with other dashboard components, for example it is a trivial step to make a component listen to a variable (like reporting period) and redraw its contents when the value of this variable changes.
- Components have many parameters that can be used to customize the behavior of this component. In fact many parameters expect a javaScript function, which allows you to insert javaScript code directly into the component.
- Components can easily morph to other types. For example you can change a simple bar chart but dynamically in a dial chart or even a data-table.

The CDF provides a rich set of components such as:

- Graphs such as bar-charts, line charts, dial charts.
- Data tables
- Selectors such as radio buttons and selection-lists
- Textboxes with auto-completion
- Connections to different data sources, such as JDBC-SQL, JNDI-SQL, Mondrian-mdx

The client-side of the CDF consists of html and javaScript code to provide a dynamic and interactive interface. Advanced developers can insert their own sniplets of javaScript and jQuery-code to provide for a high level of customization of the dashboards and its user interaction.

2b. Community Chart Components (CCC)

Good chart components are the basic visuals for most dashboards. Several open-source charting frameworks are available. However, most of the chart frameworks that have been around have been primarily built for integration in traditional applications. CCC present a new and attractive set of

components aimed for web-based deployment of graphs that allow a neat user-interaction. These components have all the components to make them highly interactive. Graphs can show pop-up windows with data values when you hover your mouse over a data point and it is possible to couple events to these data points[2] of chart elements. Some more background on and examples of CCC-components are given in this CCC-presentation (http://tinyurl.com/wd-ccc).

2c. Community Data Access (CDA)

The dashboards need to be connected to a data source (relational database, multidimensional database, XML, etc…) to provide the data that needs to be shown/visualized. CDA is a component developed to gather, merge and deliver data from different sources in a uniform manner. It allows the definition of the data sources and the data-access queries at the server-side such that the client-side of CDF can use these data sources. CDF is able to access MDX sources and SQL sources directly, however, data-access via CDA brings a several important advantages, i.e.:

- Provides an interface to a much broader set of data-sources
- To allow joining data from different sources;
- Allows secure SQL-acces by avoiding SQL injection problems of CDF[3];
- To allow an easier way to export data from queries.

Most tools available can perform queries over a single data source and there's no easy way to join data coming from different databases or in different languages (joining data from a SQL query with data from a MDX query). These tasks usually require an independent ETL job to synchronize different data sources and build a new database. CDA allows the definition of a data source that joining data from multiple sources.

The client-side of CDF consists of javaScript-code that runs in the users web-browser. This code could be compromised by a skilled end-user. For this reason the default setting of CDF is to accept no SQL-queries from the client-side code, to prevent the risk of SQL injection. The CDA allows to define SQL data sources and SQL queries at the server side. End users can get access to this data provided they have the appropriate credentials. This way CDA mitigates the risk that relational data sources are compromised by a malicious end user via SQL injection.

On the webdetails website you find a presentation of the concepts of the CDA (http://www.webdetails.pt/ficheiros/CDA.pdf). For those who want to learn more about CDA take a look at the documentation (http://www.webdetails.pt/cda/cda-documentation.html). The documentation is interesting to flip through, however, when using CDE to edit your CDA resources you don't need to grab the full details of hand-coding CDA data-sources.

[2] For example. If you click a slice of a pie-chart a javaScript-event is generated. These events can be used to trigger a drill-down on that part of the graph or to generate/modify another CCC-component.

[3] In CDF sql injection is possible because the sql-query is stored at the client side. This is an important security risk as it allows an (advanced) user to insert arbitrary sql-statements and thereby to compromise your database. MDX has a more advanced security model that can prevent these problems (if you properly configure the security roles of course).

2d. Community Dashboard Editor (CDE)

CDE is an advanced editor that allows the creation of dashboards using four-step process:

1. Laying out a (multipage) grid for your dashboard,
2. Specifying a set of data source components to connect to the data to be visualized,
3. Creating the visual components, connect them to data and assign them to the lay-out grid, and
4. Introduce more flexibility by parameterizing components and letting components interact with one another.

The CDE supports an agile development process, so you can jump back-wards and forwards in this process. In fact I advise you to do so as an incremental process where you create a simple dashboard and then start to stack additional components and interactivity works best.

The steps 1-3 can basically be performed without doing any coding. This allows the dashboard developers to focus (first) on the business requirements. However, to generate a high level of interactivity (step 4) the CTools have many parameters that actually expect a javaScript function instead of a fixed value. These functions allow for a very high level of flexibility when designing dashboards.

3. Installing the CTools

On December 2nd 2010 the CTools (CDE version 1.0 RC3) have been released. The code can be obtained from googlecode (http://cdf-de.googlecode.com/files/CDE-bundle-1.0-RC3.tar.bz2). This is a Release Candidate (RC) and not yet the General Availability (GA) release. The CTools are constantly under development, so if you want to have access to the latest features and fixes you need to use the CI-version (Continuous Integration). The preferred installation method of CTools is via the automatic installer as described in paragraph 3a. If for some reason to use the last Release Candidate instead of the CI-version use the procedure in paragraph 3b. Your last option is to install the complete set of CTools from source-code, with is very briefly described in paragraph 3c.

3a. Using the automatic installer to get the latest version of the CTools

The CI installation is easily done via the ctools installer, which can be obtained from:

https://github.com/pmalves/ctools-installer

A short description of the requirements and usage can be obtained from a blog post by Pedro Alves:

http://pedroalves-bi.blogspot.com/2011/06/ctools-installer-making-things-fast.html

As an extra this installation procedure also installs the latest Saiku, a modular open-source analysis suite offering lightweight OLAP which remains easily embeddable, extendable and configurable. version of Saiku.

3b. Manual installation of the Release candidate of CTools

The CDE version 1.0 RC3 has been tested with the Pentaho bi-server version 3.6. and 3.7. For the client-side Mozilla Firefox is the preferred web-browser, although the CTools should work fine with Google chrome and Microsoft internet explorer.

The installation of the CTools is very simple.

1. Shut down the bi-server,
2. Remove older version of the CDE-files from the pentaho solution folder[4],
3. Unpack the new bundle to the pentaho solution directory while retaining paths,
4. Restart the bi-server.

When you log in to the user console you should see an additional icon for the CDE-editor. You can use the gold-brown button under the CDE logo to start the CDE. You can also start the CDE by selecting file/new/CDE in the user console. Figure 1 shows the start-up screen of the pentaho user console; the yellow annotation boxes show the two options to start the CDE.

[4] The pentaho-solution folder contains datafiles used to generate content for your Pentaho user console. If you bi-server is installed in “c:\program files\pentaho\biserver-ce” then the pentaho solution folder is usually the subfolder “c:\program files\pentaho\biserver-ce\pentaho-solutions”.

Figure 1: Start-up screen of pentaho user console after installing the CTools.

When the bi-server is running under Linux be careful that the ownership of all CDE files is such that the bi-server is able to read them. If you run the bi-server as root-user this should not be a problem. However, if for example you run the bi-server as a user pentaho with group pentaho its best to change ownership of all the installed files to this user. This following command will change ownership for a folder named CDE and for all files and folders in this folder and for all sub-folders:

```
sudo chown -R pentaho:pentaho  CDE
```

The sudo prefix is not needed if you already have root-permissions.

3c. Installing CTools from source-code

Of course it is possible to install the CTools from the source-files (we're talking about open-source, don't we?). The sources can be obtained from:

- http://code.google.com/p/pentaho-cdf/
- http://code.google.com/p/pentaho-cda/
- https://github.com/pmalves/ccc
- https://github.com/pmalves/cde

Installation from source files is only used by advanced users, so this installation procedure is left as an exercise to the reader[5].

[5] You need to add some 'override.properties' files to introduce the appropriate settings before you perform the build process.

4. A small tour around the CDE editor

In this chapter we will give you a small tour in the CDE editor. During this tour you will also meet the other components of the CTools.

Basically the CDE has three main areas, i.e.:

1. File and settings
2. Workspace selector
3. Workspace

Figure 2 shows the start-screen of the CDE with yellow highlights added to each of the three regions.

4a. Files and settings

The "file and settings" region contains the following functions:

- New: create a new dashboard a pentaho solution folder
- Save: save changes such that all modification you made become permanent[6].
- Save as: save a dashboard under a new name.
- Reload: discard all edits and return to the last saved dashboard.
- Settings: here you can set your name and a brief description and choose the template to use for the dashboard.

The CDE does not allow you to save dashboards directly in the pentaho solution directory. Instead you have to save dashboards in a sub-directory.

HINT

It is best to use the browser window of the pentaho user console to create sub-directories. When you create the sub-folder yourself you need to take care that an index.xml file is generated for the folder.

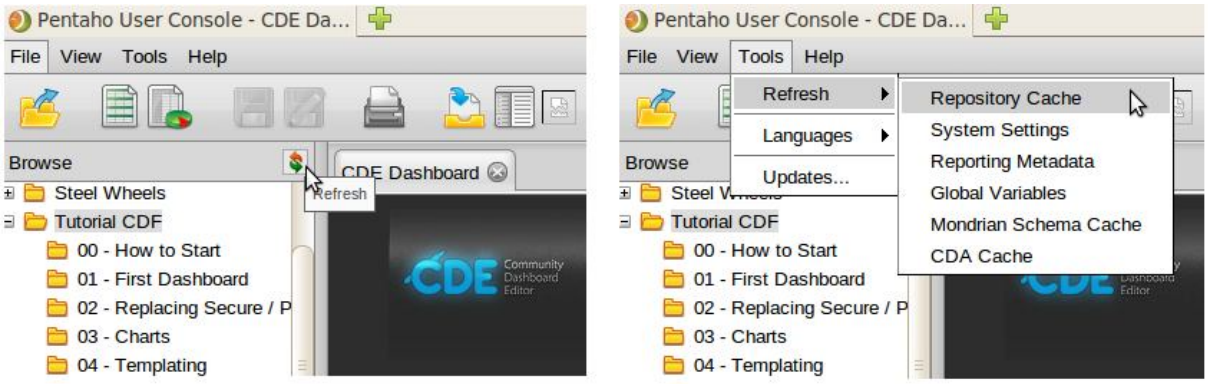


Figure 2: Two option to refresh the pentaho browser pane.

[6] CDE saves a temporary file when doing a preview, but this file should not be considered as a backup. Changes are volatile until the being saved.

It is wise to save your dashboard often. If the connection to the pentaho server is lost. This can simple happen as the server drops connections if it does not see any activity of the client. When the saving of a dashboard is successful you get a notification "Dashboard saved successfully" sliding smoothly from the top of the CDE editor (and disappearing again after a second or two)[7]. If you don't get this message the server might be slow or the server connection might be lost. Do not panic in that case and especially do not kill or reload your CDE tab, otherwise all unsaved changes might get lost. The best solution is open an arbitrary other pentaho component (for example the CDA file of you dashboard). The server will pop-up a login window and after you provide the correct credentials your server connection is active again and you can save your dashboard. However, if you think your CDE crashed and you restart the CDE editor the non-saved edits might be lost.

After you save a file to the pentaho solution folder (via the CDE) it will not be visible immediately in the file browser of Pentaho. In order to make the files visible you have to click the refresh browser button (see left side of figure 2). If you refresh the pentaho repository cache (right hand side of figure 2) the browser will also be refreshed, but this is a bit slower. A busy-icon will appear during the time that the (remote) server is refreshing the information. When the busy icon disappears the new information is received from the server. However, it will still take a number of seconds for the client-side browser to refresh the folder listing and the file list.

4b. Workspace selector

The workspace selector is used to detect the workspace that will be shown. The workspaces are:

- Layout: the lay-out workspace is used to define the lay-out of your dashboard by dividing the screen in a series of labeled regions. You can style the regions and add html-elements (text or images) to regions.
- Components: In this workspace you define and customize the different components (graphs, tables, images, etc.) and you assign each of these components to one of the labeled regions that you've defined in the lay-out workspace.
- Data sources: Components like graphs and tables will need a data source to provide their content. In this workspace you can define and edit data sources. You need to give each data source a name (label) such that you will be able to refer to these data sources in the component definition.
- Preview: When you push this button a preview of your dashboard is shown on top of the CDE.

If the preview button does not work anymore (it highlights upon being clicked but nothing happens) than please don't panic and click kill the CDE-tab or reload the CDE-tab because will cancel all your unsaved changes. See the previous chapter (save does not react) for the solution.

[7] A "Dashboard saved successfully" indicates that the bi-server has received your dashboard files. After receiving the files the server still has to save these to disk. If a problem occurs during the write-operation the files (for example because you create a folder by hand that the server does not have write access on) then the CDE will not be able to notify you of the fact that your dashboard never arrived on disk.

A more elaborate description of the workspaces will follow next.

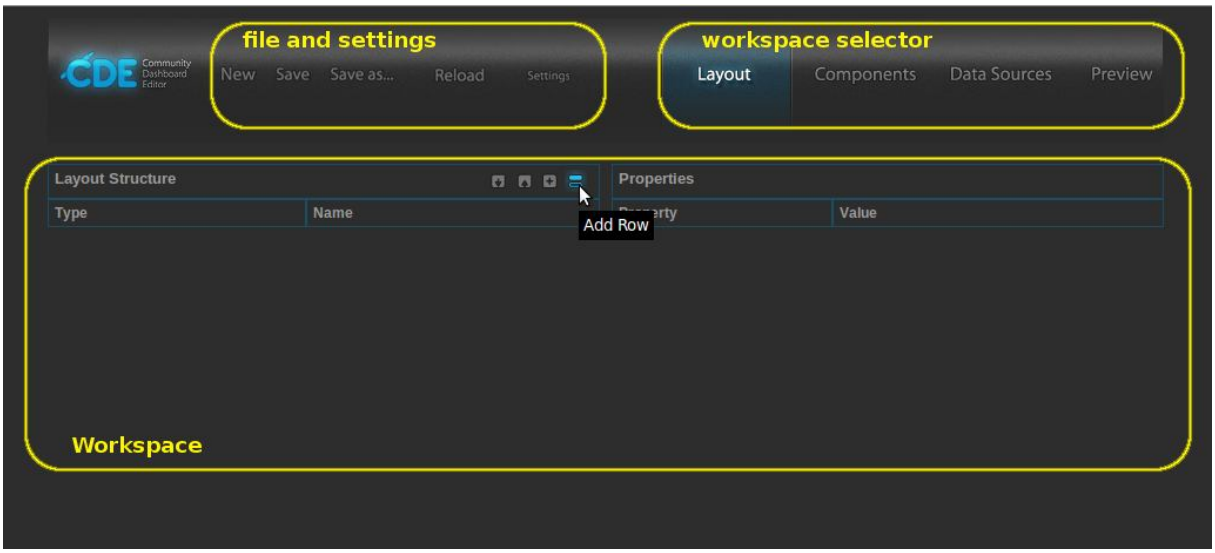


Figure 3: the three main area's of the CDE.

4c. The Workspaces

The main area of the CDE is the workspace editor. Using the workspace selector you can switch to the other workspaces. Here we will give a brief introduction to each of the workspaces.

The Layout workspace

When you start the CDE it displays the layout area with an empty layout (as shown in figure 2). The layout workspace consists of two panels. The left pane, labeled “Layout structure”, shows the current layout design. The right pane, labeled “Properties” shows the properties of the element that currently is selected in the left-pane.

In this section you will find:

1. Left pane of the layout workspace
2. Right pane of the layout workspace
3. Control of widths using Blueprint
4. Using the color-editor
5. Some suggestions for using the CDE

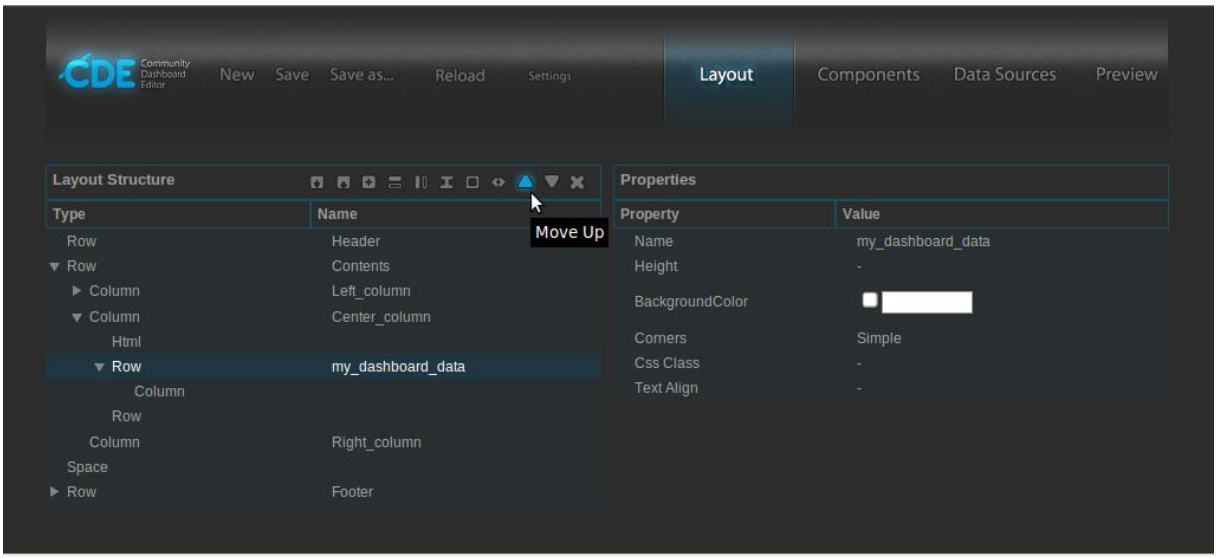


Figure 4: A simple layout design

The left hand pane of the layout workspace

Figure 4 shows the Layout workspace after we added a simple page-design. The left pane shows that the page is divided in three rows labeled “Header”, “Contents” and “Footer”. The “Header” row is a simple row containing no sub elements. The “Contents row does have a number of sub elements. These elements have been expanded. If you want to collapse this expanded element to a single line you need to click at the triangle left of the row-element. In front of the row “Footer” you also notice a small triangle, however, this triangle is pointing to the right, indicating that this row contains collapsed sub elements[8].

At the top of the “Layout structure” pane a toolbar is provided to modify the layout.

An enlarged version of this toolbar is shown above. We will explain this toolbar starting at the left and moving to the right:

1. Save as template: Save the current dashboard as a template. This template will contain the complete layout. You can select whether the template also includes the component and data sources that have been defined in the other workspaces. After saving the template the CDE will issue a new-command and start with an empty layout.
2. Apply template: When you press this button a scrollable list with a number of standard layout-templates (two-column, three-column, etc...)[9].
3. Add resource: Add a cascading style sheet (.css) or a javaScript (.js) resource to the current dashboard.

[8] For the experts: Under the hood the layout of rows and column is mapped to a structure of nested in a html file that is generated dynamically by the pentaho-CDE-plugin.

[9] The apply template button only shows the default-template. If you save a template using the save-template button then you need to open it as a normal dashboard file via the browser.

4. Add row: A row is added to the layout. If the current selected element within the layout is a column then the row will be added as a sub element of this column (one level deeper). In all other cases the row is added at the same level as the selected element.
5. Add column: A column is added to the layout. If the current element is a row then the column is inserted as a sub element of this row. In all other cases the column row that contains the selected element.
6. Spacer: add some space between subsequent elements. If a row is selected a vertical spacer is added, if the current selection is a column then a horizontal spacer is added.
7. Add image: Add an image to the current select row or column element.
8. Add html: Add an html block to the current select row or column element.
9. Move up: Move the current element one step up in the list (while remaining at the same level).
10. Move down: Move the current element one step down.
11. Delete: Delete the selected element.

The toolbar is dynamically generated based on the current element being selected. For example, if you take a look at the initial layout in figure 2 you see a very short toolbar containing only four elements.

The right hand pane of the layout workspace

The first property in figure 4 is the "name" property[10]. This name is used to set the name of the element. This name is also shown in the name column of the Layout structure. These names are also used within the CDE to attach components to the different cells of the layout that you have defined. The only special characters allowed in a name is '_'.[11]

Control of widths using 'Blueprint.css'

The lay-out of CDE is based on 'blueprint.css' stylesheets. To minimize the need for manual control of widths the 'blueprint.css' divides a screen in 24 columns [12].
If you create a column in the layout workspace it has three parameters that are defined in terms of the Blueprint-columns:

1. Span size: the width of this column,
2. Prepend size: the amount of empty space to be inserted before this column, and
3. Append size: the amount of empty space to be inserted after this column.

[10] The "name" property corresponds to the html-id tag of an element.

[11] According to the CDE a '.' is also allowed in a name. However, in practice name's containing a '.' give unpredictable results.

[12] In the layout workspace the width is set in terms of blueprint-columns. However, it is possible to force the width of a component in pixels in the component view. However, this is not a real issue it is best to leave the width of a component empty such that the width of the component is derived from its contents and the size of the layout cell ( ) this component resides in.

If the combined width of the columns on a line exceed the page-width then the columns that do not fit will be wrapped to the next line. The blueprint.css does not state a metric for the heights. The height of a column is defined by the height of its contents. An empty column therefore becomes invisible as it has a height of 0 pixels (unless you explicitly did set the height of the column). For more information on the blueprint framework you can visit http://www.blueprintcss.org.

Using the color-editor

The CDE includes an option to set the BackgroundColor property of Rows and Columns. In order to set the color you have to click the small square box next to the BackgroundColor property (see figure 5).

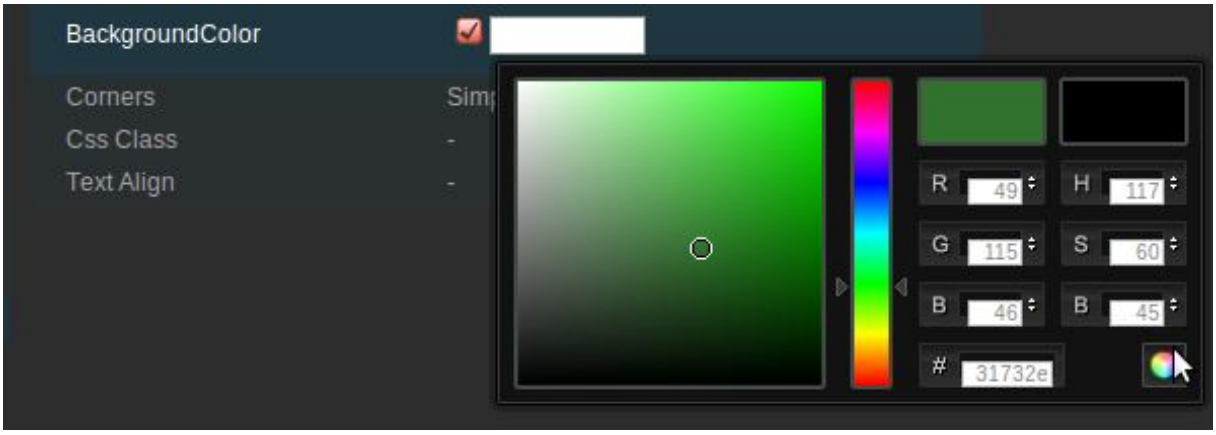


Figure 5: Color selector

When you click this box a check-mark is set and the color-editor pops up. To select a color you need to apply the next three steps:

1. Select the Hue (color-range) by drawing the tiny rectangles on the rainbow-selector in the middle to the appropriate color range,
2. Select the Saturation and Brightness by drawing the white circle in the square box to the exact color you want (initially this circle is locate in the left bottom of the box), and
3. Click the color-circle in the right bottom to confirm the selection.

It is also to enter the color-code directly in terms of RGB, HSB or hexadecimal color code (field #) by entering a number in the appropriate boxes of figure 5.

Some suggestions for using the CDE

When entering a property in the right hand pane it is important to hit <ENTER> afterwards. If you do not hit enter and select another element from the left hand pane it will be assumed that you entered <ESC> and the value you entered will be discarded without notice. Due to the fact that you switched to another element you will are likely not to notice that your data-entry was discarded. If you use the mouse to move to another property in the right hand pane the data-entry for the preceding property will be accepted.

Several field have auto-completion functionality, which shows up as a list below the field that you are entering. So, if you want to move outside the textbox because you don't want to see the mouse

pointer over you data-entry it is better not move it below the box. It will take the CDE a few seconds to determine the list of auto-completion options, but when this list appears the item below your mouse-pointer will surprisingly replace the contents you are editing.

Be careful when you want to add a few columns or a few rows simultaneously. As soon as you add an element the toolbar might change. For example when you start with an empty layout the add row button is the rightmost button of the toolbar (see figure 3). However, as soon as a row is added the delete-button is added at the rightmost position of the toolbar (see figure 4). So doing a double click at the rightmost button in an empty layout will add a row and delete that row due to the second click. The end result is a layout that is still empty.

The right pane of figure 4 shows the properties of the layout element that currently is selected. These properties can differ per element, so the list changes dynamically if you move to another element in the Layout structure.

The component workspace

The component workspace is shown in figure 6. This workspace consists of three panes. The left pane enumerates all components-classes that can be available in the CDE. The middle pane shows the components that actually have been added to the current dashboards. The right pane shows the details of the concrete component instance that is selected in the middle pane.

In this section you will find:

1. Left pane of the component workspace
2. Middle pane of the component workspace
3. Right pane of the component workspace
4. Customize the dataset interpretation of chart components
5. Adding interactivity via javaScript functions
6. Some suggestions for using the CDE

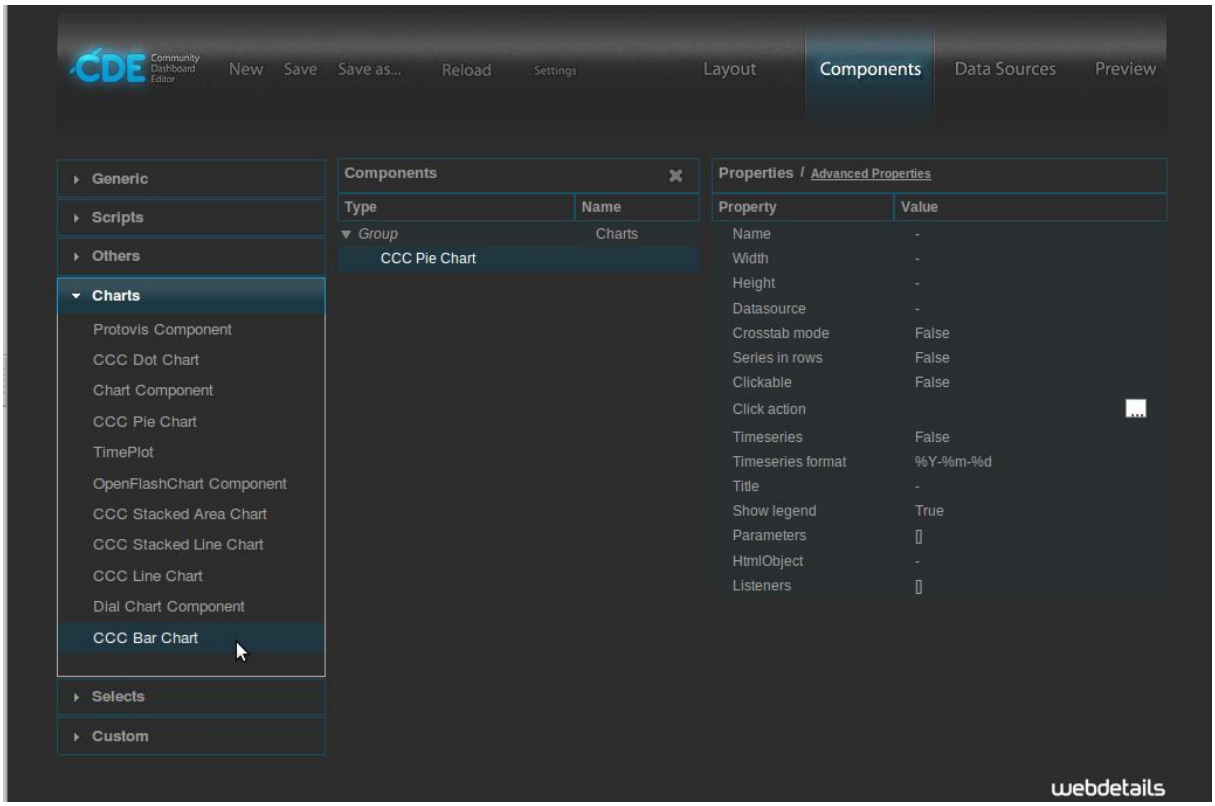


Figure 6: The Components workspace.

Left pane of components workspace

The available components are categorized in six categories. The components are:

1. Generic
    a. Simple parameter: introduce a javaScript parameter and set its default value to primitive value (integer, double, string, etc..). JavaScript parameters are used amongst other to connect components.
    b. Custom parameter: introduce a java-script parameter that with compound value default value (array, object, function…)[13], or where the value is initialized by a java function.
    c. Date parameter: A date parameter can be set to a specific date, for example today's date, or it can be coupled to a date-picker.
2. Scripts
    a. JavaScript function: define a javaScript function[14].
3. Others
    a. PRPT component: This component can execute the (new) Pentaho Reporting unified format.

[13] The purist under the javaScript programmers would say that a javaScript function is also an object and an array, however, this more verbose descriptions does not do any harm.

[14] If you have a long function, or multiple function to add then you can also put a javaScript file (.js) at the server and include this javaScript file as a resource via the lay-out workspace.

b. Pivot component: Excecutes a JPivot Action Sequence and creates an iframe where the pivot table is embedded.
c. Comments component: Adds a comments section to a page.
d. Navigation Menu component: Introduces a menu that allows you to browse the complete pentaho solution repository (non-hidden folders).
e. Table component: Shows the contents of a data-source in the form of a table. By default tables are sorted, searchable and paginated. These is an extensive list of options to customize this component.
f. Execute Prpt component: Executes a prpt and displays the result on a new pop-up window.
g. Traffic component: This component can generate a traffic light image without needing a xaction file.
h. Execute xaction component
i. Freeform component
j. Text component: Show the result of an expression, where expression is a javaScript function.
k. Related content component
l. Button component
m. Query component: executes a sql or mdx query and returns the result.[15] The CDA offers a more versatile set of query tools.
n. Pivot link component

4. Charts
   a. Protovis component: a graphical toolkit for visualization. The CCC-chart are also based on Protovis. However, this components gives you access to the full power of the protovis library, which is a powerful javaScript library for a wide range of visualizations.
   b. CCC Dot chart: the CCC charts are the preferred charts in the CDE because these charts look pleasant and have many features for interaction, like tooltips, animation and drill-down options.
   c. Chart component: Component using the Jfree chart graphics.
   d. CCC Pie chart: See CCC dot chart.
   e. TimePlot
   f. OpenFlashChart component: Component using OpenFlashChart for visualization.
   g. CCC Stacked Area chart: See CCC dot chart.
   h. CCC Line chart: See CCC dot chart.
   i. Dial Chart component
   j. CCC Bar chart: See CCC dot chart.
5. Selects
   a. TextInput component: text input without auto-completion
   b. Select component: This component presents a list-box to select a parameter. It returns the selected parameter, or its display-value.
   c. Date range input component

[15] In many bi-servers this sql-query via this component is not allowed as client-side sql introducing a security risk (sql-injection).

d. Check component: list one or more checkboxes and connect these to a javaScript variable.
e. Month picker component
f. Radiobutton component: A radio button component (enforces that exactly one item of a list is selected.)
g. Auto complete component: Enter text with auto-completion.
h. Selectmulti component: Select one or more elements from a list and put these in a javaScript variable.

6. Custom
    a. Raphael component: Raphael is a library for Scalable Vector Graphics (svg).

Middle pane of the component workspace

The middle pane shows the components that you have instantiated. These component are grouped by category. In the example in figure 6 we only have one component instantiated (a CCC Pie chart component) that belongs to the 'group' charts. The middle pane has a dynamic toolbar shown a the top.

An enlarged version of this toolbar is shown above. From left to right we see the buttons:

- Move component one line up.
- Move component one line down.
- Delete the component.

In figure 6 the toolbar only contains the delete button, as the middle pane only contains a single component, so there is nothing to reorder.

Right hand pane of the component workspace

The right-hand pane show the properties of the component that is selected in the middle pane. Next graph shows component workspace. The available list of properties is already fairly long. For the CCC Pie chart in figure 6 this list contains 15 properties that can be set. If you click the 'Advanced Properties' button at the top of the right-hand pane the list of parameters will even be extended by adding the advanced properties to this list. The most important properties that appear for many components are:

- Name: This is the name of the component. This name will also be shown in the middle pane.
- Width: The preferred width of the component in pixels. By default the width will be set based on the lay-out you have designed in the layout-workspace. So only, use this option if the lay-out engine of CDE does not provide your component with sufficient space.
- Height: Again width in pixels. Only set this parameter if the defaults used by CDE do not give a satisfactory result.
- Datasource: The CDA data source (defined in the data source workspace) that this component will be connected to.
- Parameter: the javaScript parameters that can be used by this component.

- HtmlObject: gives the name of the layout container (defined in the layout workspace) that should display this component.
- Listeners: A list of javaScript variables that this component will watch. If the value of one of these javaScript variables changes then a redraw of this component will be triggered.

Customize the dataset interpretation of chart components

Dashboards consist of visual elements and one of the most important elements is of course the chart that is used to visualize your data. The charts have two Boolean parameters that initially might be difficult to understand, i.e. *crosstab mode*[16] and *series in rows* both having a default value false.

If *crosstab mode = true* the chart component is expecting a matrix-like dataset where series and categories form the rows and columns of the matrix and the data is in the matrix (so the first data column is used for either series or category labels). If *series in rows = true* a series is a row of the matrix and the categories are set in columns. The schema's below show how the data is interpreted using the as markers:

- S = series (as shown in the legend)
- C = categories (X-axis label for a vertical graph)
- D = Data value
- … = The table can be extended in this direction

The shaded cells are the column headers.

Crosstab mode = true
Series in rows = true

	C1	C2	…
S1	D	D	…
S2	D	D	…
S3	D	D	…
…	…	…	

Crosstab mode = true
Series in rows = false

	S1	S2	…
C1	D	D	…
C2	D	D	…
C3	D	D	…
…	…	…	

Notice that the column headers will show in your chart in *crosstab mode = True*.

If *crosstab mode = False* the chart component is expecting a three column dataset where each row corresponds to exactly one data-point. Here the parameter series-in-rows determines the order of the columns. If *series in rows = false* the expected order is Serie, Category, Data. If *series in rows = true* the expected order is Category, Serie, Data. The schema's below show how the data is interpreted using

In the CCC-charts the category labels you use should be text strings or floating points values. If you use integer values a category labels the charts might get corrupted[17].

[16] As a rule of thumb: Mdx queries and sql-queries with a "GROUP BY" clause are used to generate data-sets in cross-tab mode. Other sql-queries are more likely to generate data-sets that aren't in cross-tab mode.

[17] Internally the CCC library uses the values 0,1, … n to refer to the categories along the x-axis. If you also use these numbers as a category label the datapoint will be drawn in the wrong category. If you need numerical values along the category axis make sure to use floating-point formatted numbers.

Crosstab mode = false
Series in rows = false

S	C	D
S	C	D
S	C	D
…	…	…

Crosstab mode = false
Series in rows = true

C	S	D
C	S	D
C	S	D
…	…	…

In *crosstab mode* = *False* the column headers (names) are not part of the dataset and therefore will not become visible in your chart.

Customization of CCC charts using extension points

For the CCC charts you have an extensive list of options under the label "advanced properties". On of these options is extension-points. The extensionpoints allow you to set amongst other the font-family, font-size, text-orientation of the different textcomponents of the chart. When clicking on extentsionpoints you can create a list of name-value pairs. The name has the format A_B where A is the item you want to customize and B the property to be customizes. So for example, if you want to customize the font of the title A = *titleLabel* and the property B = *font*, so the parameter to be set is called *titleLabel_font.*

Here I give you a non-exhaustive list of the available or item-types and the properties you can set as an indented list.

- Text-items (e.g. titleLabel, xAxisLabel, yAxisLabel, legend, barLabel, pieLabel):
  - font: for example "13 px serif"
  - fillStyle: for example "yellow"
  - textAngle: an angle in radians (you can to turn the horizontal axis labels at an angle to prevent them from overlapping[18] with one another).
  - textAlign: for example "left", "center", " right".
  - textBaseline: for example "top", "bottom".
  - textStyle: for example "black"
  - text: A function that is used to modify/format the text (for example function(d) {return d.substr(5) } to limit the length of the text to the first five characters.
- dot (the marker used in a Dotchart) :
  - fillStyle: for example "white"
  - shape: for example "square"
  - shapeRadius: for example 4
- line-items (line, xAxis, yAxis, dot, xAxisRule, yAxisRule):
  - lineWidth: for example 0.5
  - fillStyle: for example "green" or "#00ff00"
  - strokeStyle: for example "blue"

[18] If you change the textAngle of the xAxisLabel you probably also need to set the textAlignment to "Left" to prevent the text from crossing the xAxis.

- bar (the marker used in a BarChart)
- pie (for the pie-charts):
  - innerRadius: for example 10
  - strokeStyle: for example “white”

Adding interactivity via javaScript functions

At many locations you can insert javaScript functions. If you want to insert a function use the syntax:

function() { <body> return result; }

Do not put a ‘=’ in front of the function and do not put a ‘;’ after the closing bracket, otherwise the function is going to be interpreted as a text-string instead of a function.

HINT

If you want to insert a complex function in the dashboard it is best to put this function in a separate resource-file (.js). The javaScript that you enter in the CDE will be included in your dashboard-file without being (thoroughly) validated. If you make an error your scope (too many of too few closing brackets) this will screw up the rest of the file and lead to quite unpredictable results/errors. If you put it in a separate .js file the file will be completely discarded if it contains a scoping error, and other components of your dashboard will not be infected by this scoping error.

An more elaborate description of most of these components and their parameters can be found in the pentaho solution repository. Via the user-console go to the folder:

BI developer Examples/CDF/Documentation.

Use ‘File/Open’ or use the browser (View/browser)[19]. If the ‘BI developer Examples’ does not show in your repository then you need to edit the index.xml file in this folder and set visibility to ‘true’ (or ask your system administrator to do so).

Some suggestions for using the CDE

Most properties will have an auto-completion function. When you edit such a properties a list pops up containing the options to choose from. This list is context sensitive. For example, when you click on the HtmlObject a list of all layout containers appears. However, when you click on the parameter property it will show a list of defined javaScript variables[20]. When entering a value of a property the CDE only does a rudimentary validation, so if you include the value that does not yet exist the CDE will not complain. Also when you are previewing a dashboard the CDE will not complain a lot if there is an error. The only signal you get is that your component is not drawn at all. Therefore, I recommend you to use the auto-completion function whenever possible. Most components have a

HINT

[19] After browsing to the folder of the component in the top panel you have click on the file in the lower panel to view it.

[20] This list will be limited to the javaScript variables that you defined using the parameter components. It will not include the parameter that have been defined in your own javaScript resources that you included in the CDE.

lot of properties. If a component fails it might take you a significant amount of time to track down the property-value that you mis-spelled.

HINT

The CDE is a bit parsimonious with its warnings and suggestions when you make an error in a component. Actually the main error-message is a component that does not get rendered to the screen. I see two solutions to this issues.

Option 1: Do not ever make errors, en

Option 2: Preview your dashboard after every significant change you make.

Of course the first option is the fastest. However, for most mortals I would advise the second option. There is a little time lost in doing many previews. However, it can save you an awful lot of time as it is much easier to pin-point the error if you only made one or two small changes. If you hit the save button after each successful preview you can easily backtrack to the previous version via the reload button when you observe an error.

The data source workspace

Figure 7 shows you the data source workspace. It has the same layout as the components workspace. Again you see the list of all available data sources on the left-hand pane, the middle pane shows the data source that you have created for this dashboard and the right-hand pane shows the properties of the current selected data source.

In this section you will find:

1. Left pane of the data source workspace
2. Middle pane of the data source workspace
3. Right pane of the data source workspace
4. JDBC vs JDNI
5. Some suggestions for using the CDE

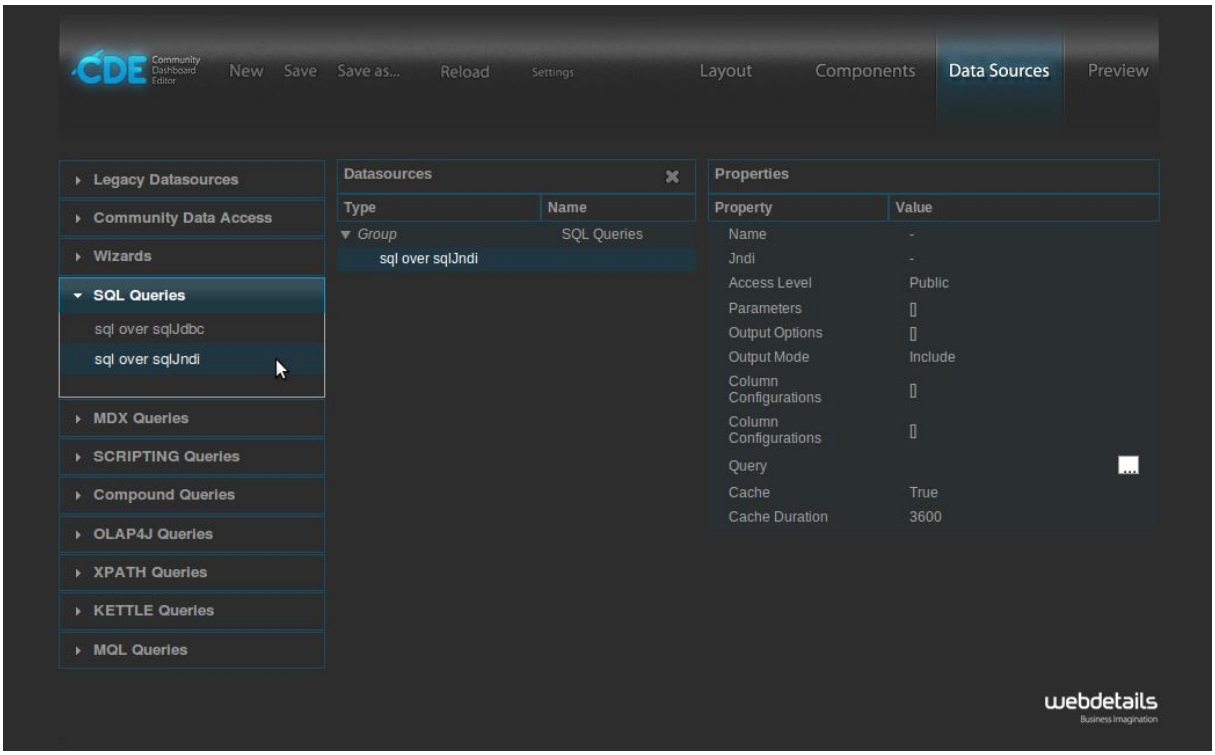


Figure 7: the Data sources workspace.

Left hand pane of the data source workspace

The available data sources in the left hand pane are:

1. Legacy data sources
    a. OLAP MDX query
    b. SQL query
    c. Xaction result set
    d. Kettle transformation
2. Community Data Access:
    a. CDA data source: include an existing .cda file in your dashboard
3. Wizards
    b. OLAP members wizard
    c. OLAP Chart wizard
4. SQL queries
    a. Sql over sqlJdbc: access a database via the Java DataBase Connectivity API (Jdbc).
    b. Sql over sqlJndi: access a database via the Java Naming and Directory Interface (Jdni).
5. MDX queries
    a. mdx over mondrianJndi
    b. denormalizedMdx over mondrianJdbc
    c. denormalizedMdx over mondrianJndi
    d. mdx over mondrianJdbc
6. Scripting queries
    a. Scriptable over scripting: currently the only supporting scripting language is BeanShell, which is a small and embeddable Java source interpreter.

7. Compound queries
    a. Join: Create a join of two arbitrary other datasources. So it is also possible to join tables that reside in different database systems.
    b. Union: Concatenate two datasources that have the same column format.
8. OLAP4J queries
    a. Olap4J over olapJdbc
    b. Oloap4J over olapJndi
    c. denormalizedOlap4j over olapJdbc
    d. denormalizedOlap4j over olapJndi
9. XPATH queries
    a. Xpath over Xpath
10. KETTLE queries
    a. Kettle over kettleTransFromFile
11. MQL Queries
    a. Mql over metadata

Middle pane of the data source workspace

The middle pane shows the current data sources defined for this dashboard. On top of the middle pane you have the same dynamic toolbar as described for the components workspace.

Right hand pane of the data source workspace

The right-hand pane show the properties of the data source that is selected in the middle pane. Next graph shows component workspace. The most important properties that appear for many data sources are:

- Name: This is the name of the data source. This name is also be shown in the middle pane and this name is to make a connection to this data source for a component in the component workspace
- Width: The preferred width of the component in pixels. By default the width will be set based on the lay-out you have designed in the layout-workspace. So only, use this option if the lay-out engine of CDE does not provide your component with sufficient space.
- Height: Again width in pixels. Only set this parameter if the defaults used by CDE do not give a satisfactory result.
- Datasource: The CDA data source (defined in the data source workspace) that this component will be connected to.
- Parameter: the javaScript parameters that can be used to create parameterized datasources[21].
- Query: the query used to extract data from data source.

[21] The parameterization is limited by the underlying data-engine. In a sql query you can use the parameter to limited the result set (WHERE statement) or to changing grouping or ordering. However, in SQL it is not possible to parameterize which columns you want to retrieve because this would involve a security risk.

JDBC vs JDNI

As you can see the CDE/CDA provides a rich set of datasources. Many datasources are available in a Jdbc and a Jdni variant. When using the Jndi datasources you need to define the datasource in the administration console of the pentaho server. I prefer using Jndi over Jdbc when your need access several tables from a database. A Jdni datasource is referenced using a single parameter, its name. A Jdbc datasource requires you to provide four parameters instead of one:

HINT

a. Name of the database
b. Connection-URL (a fairly complex connection string)
c. User name used to connect to the database server
d. Password that belongs to this username.

Some suggestions for using the CDE

CDE does not do any error-checking on the queries that you enter. If there is an error in a query then this query usually fails. Errors returned by the database engine will not be shown to you. So the only feedback you get is that the component that uses the data is not rendered. We therefore recommend that:

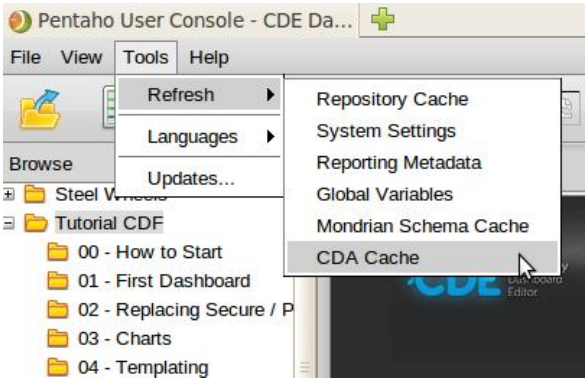
HINT

a. You enter copy-past the query to your database/mdx client to test whether this query is valid and returns the correct result set.
b. Check the data source via de CDA previewer to see whether you get the proper results. An error in the data-source settings might prevent the data source from returning the right result even when the query is correct.

The CDA definitions are stored in a separate file (the .cda file). This file can be opened by clicking on it in the file-browser of the pentaho user console. However, there are two caveats:

a. You need to hit the save button of the CDE otherwise the newly created definition will not be stored in the .cda file (instead it is stored in a temporary file that is not visible in the file-browser).
b. If the data-source is executed, either in the CDA previewer, or because you used the CDE preview then the result have been stored in the CDA cache. The result are <u>also cached if the query return zero records due to an error in your query</u>!. I advise to explicit save the dashboard and to clear the CDA cache after repairing/modifying a data source, because some versions of the CDA do not use the temporary CDA file[22].

[22] Clearing the CDA cache is needed in some versions of the CDE because the CDA does not detect correctly that the .cda file is more recent than the cached results.



The snapshot above shows you how to refresh the CDA cache. Beware, if you refresh the repository cache, the CDA cache will not be refreshed as this is a separate cache.

5. The CDF component event model

The CDF component allow you to generate interactive dashboards with a lot of complex interactions between the different components. Figure 8 shows the eventmodel used to connect the components to each other and to javaScript variables. I will first explain the different components of this model. Next I will show you how parts of this event-model correspond to parameters of you components.

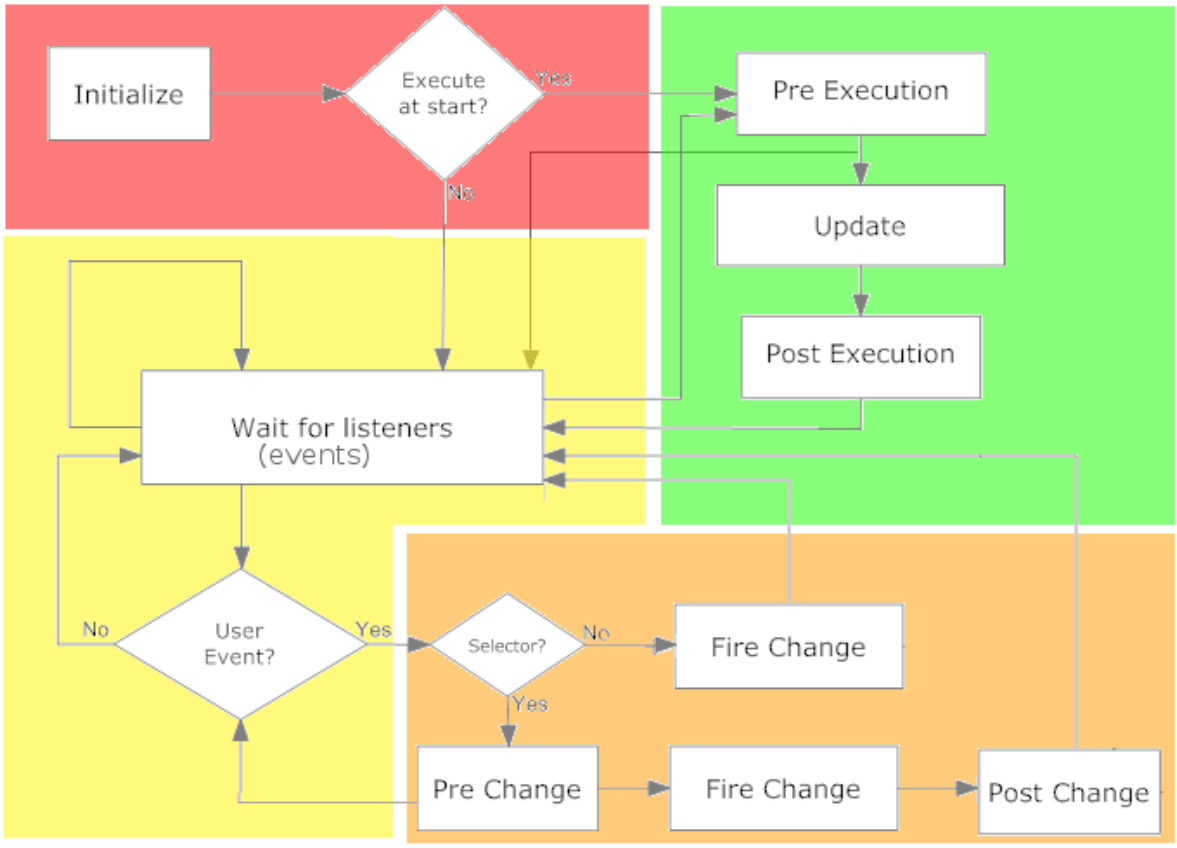


Figure 8: Schematic representation of the CDF-interaction model

5a. The event model

We will explain the event model of CDF via a walk through figure 8, starting at the red shaded area (left-top of figure). When the dashboard is loaded in a web-browser all components (javaScript objects) will be instantiated. For each of the components it will be checked whether it should be executed at immediately (go to the green shaded area at the right top) or the component will wait for events before being executed (yellow shaded area at the left bottom of the figure).

The green shaded area (right top of the figure) describes the execution process (rendering) of the components. The execution of a component consists of three steps:

1. preExecution step: data can be prepared before actually updating (rendering the component). Furthermore it is possible to break the execution and move back to 'Wait for listeners' (step 2 and 3 will be skipped and a transition is made to the yellow area).

2. Update step: The component executes all actions needed to render itself to its designated part of the dashboards.
3. postExecution step: This function can be used for example to signal that the component has finished or to perform some post-processing on the result of the component. After a component is executed it will wait for event (yellow shaded area).

The yellow shaded area (left bottom of the figure) represents all component that are waiting for user interaction. If a user interacts with a component (mouse or keyboard input) then the component will move to the orange shaded area (right-bottom of the figure). If the user input is accepted a fireChange is executed, which sets a new value for a one parameter. As a result of the fireChange event all component that listen to that parameter will be executed such that they can update their contents and render the results (green shaded area at the right top).

The actual path taken to achieve a fireChange event is different for selector and other components. For a select component the preChange function is executed. This function can be used to validate the user input. If the input is not valid the event will not be propagated and the dashboard return to the waiting area for new input, otherwise the fireChange event is used to modify a change of the parameter. The postChange event can be used by the selector to do its post-processing.

5b. How component parameters relate to the event model

After discussing the event-model it is time to translate this to the parameters of the components.
The relevant parameters are:
Advanced properties of components.

1. executeAtStart: A boolean (or function returning a true/false value) that tells whether the component be executed when the dashboard is loaded?
2. Parameters: An array of (input-)parameters that are passed to the component. For example in a sql-query these parameters can be used to customize the query based on user input.
3. Parameter: The (output-)parameter of this function.
4. Listeners: This is an array of parameters that need to trigger the execution of the component and thus the rendering of this component.[23]
5. PreExecution: This function is executed before the component is initialized or updated. If the preExecution returns false, the component is not updated.
6. PostExecution: This function is computed after performing the update and can be used to signal that the update is ready.
7. PreChange: The prototype is *function(v)*, where *v* is the candidate new value for the parameter (see item 3 of this list) of this component. This function is only relevant for selector components and can be used to validate the input. If this function returns false the fireChange event will not be executed[24].
8. PostChange: This JavaScript function will be executed after the change was fired.
9. Click action: The CCC charts can be made clickable (set Clickable = true). If you do so than clicking a mark (point, bar, wedge, …) will trigger a function with prototype:

[23] Objects from Parameter and Parameters are not automatically added to the listener array. So if the selector needs to be updated if its parameter is changed by some other component you need to add it to listeners too.

[24] So the PreExecution should not call the fireChange function directly for parameter of the component. As this call will already be performed by the Dashboard engine if PreExecution returns the value *true*.

function(s, c, v) { <body> }, where

s = label of the series

c = category label (value at the x-axis)

v = value (value at the y-axis)

The most basic version of the function only does a Dashboards.*fireChange(param, value)*. However, it is also possible to create a more complex interaction by popping up a menu and let the user select the desired action.