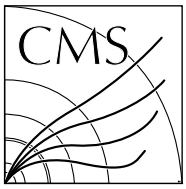
Available on CMS information server

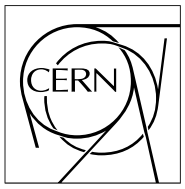
CMS DN -2010/013



The Compact Muon Solenoid Experiment

Detector Note

The content of this note is intended for CMS internal use and distribution only



22 July 2010

Display Framework Documentation for Beam and Radiation Monitoring group. CMS Detector Note.

Olga Filyushkina
CERN
Matthew Hollingsworth
University of Tennessee

Abstract

The real-time online monitoring framework used by the Beam and Radiation Monitoring group at the Compact Muon Solenoid at Large Hadron Collider, CERN provides real-time diagnostic of beam conditions. The framework provides data quality assurance, low latency, data caching, and real-time visualization. We present the user's guide for this framework.

Display Framework Documentation for Beam and Radiation Monitoring group. CMS Detector Note.

Olga Filyushkina
CERN

Matthew Hollingsworth
University of Tennessee

Geneva, July 22, 2010

CONTENTS

1. BRIEF INTRODUCTION

The real-time online monitoring framework used by the Beam and Radiation Monitoring group at the Compact Muon Solenoid at Large Hadron Collider, CERN provides real-time diagnostic of beam conditions, which defines the set of the requirements to be met by the framework. Those requirements include data quality assurance, vital safety issues, low latency, data caching, etc. The real-time visualization framework is written in the Java programming language and based on JDataViewer–a plotting package developed at CERN.

2. CONFIGURATION FILES IN GENERAL.

The on-line monitoring framework is an interactive tool for displaying real-time data in different formats. It allows multiple data representations and has a simple and flexible configuration technique. To run a display with the help of a browser within the .CMS Network, it is necessary to have:

- XML configuration file is responsible for the display set up and data structure.
- HTML configuration file connects the XML config file with Java Applet code.
- CSS file is optional and responsible for special display formatting.

Java web start display requires following configuration files:

- XML configuration file is responsible for the display set up and data structure.
- Java Web Start launch file with .jnlp extension.
- XML layout configuration class (optional).

2.1 XML Configuration file

The XML file represents the structure of the data to be displayed and the display itself at the same time, so it must be consistent and follow the predefined hierarchy. Main tag is the container fram <Viewer> ... < /Viewer>, which contains different tabs with charts and a toolbar. It may have any number of second level element, but only one Viewer tag is allowed per XML file. Second level tag is called <DVView> .. < /DVView>. It is responsible for one tab in the display. It is possible to have multiple tags within one display instance, but it is not recommended because it is only possible to observe one tab per display at a time, while transmission of data would increase significantly. Third level element is <DataView> ... < /DataView>, which

represents a single chart within the display. Several DataView tags form a DVView element. Final level element is called Plot. A single <Plot/ > is responsible for one plot on a chart and there no limitations for number of plots within one chart.

2.1.1 Attributes and Attribute's specifications

Each tag contains a certain number of attributes, that describe the display and data to be monitored.

Viewer's attributes are:

- "Name" defines the name of the display.
- "dipCacheUrl" specifies URL to dipCache server, which contains cached dip data.
- "showExplorer" toggles data explorer on or off.
- "capture " sets capturing function if true. Possible values: true, false, instant. "Instant" value is applied to histograms only ans allows to make a png file every time new data is added to a histogram.
- "path" is auxiliary attribute used when the capture is set true. It specifies the directory where png files produced by capture function are saved.
- "Time" is another auxiliary attribute of capture function. It sets the time period when one png file is made. By default it is set to 100000 milliseconds.
- "fileName" specifies the file name for the capture functions.

An example of <Viewer> ... < /Viewer> tag:

```
<Viewer
        name="BRM Monitoring"
        dipCacheUrl="http://10.176.29.237:8080/brm-wbm/"
        showExplorer = true >
... </Viewer>
```

DVView has only one attribute called "name", which defines the name of a tab in the display. An example:

```
<DVView name="BCM1L">
.. </DVView>
```

The DataView element contains:

- "Name" is the name of a chart within the display.
- "axisType" defines the axis type. Default values can be set as a number or timestamp. If it is left blank, then the value of axisType will be set to number automatically.
- "multiAxis" is a boolean value which is set to be true when multiple Y axes are needed.

An example of <DataView> ... < /DataView> tag:

```
<DataView name="Current" axisType="time"
  multiAxis="false">...</DataView>
```

The plot tag defines a single plot within the display and it has complicated attribute structure.

- "Name" is the name of the plot to be displayed.
- "dipName" defines a dip subscription that contains the data to be displayed.
- "xField" is the name of the dip field to be plotted on the X axis. When xField is not used, dip time stamp is plotted on the X axis.
- "yField" is the name of the dip field to be plotted as Y values. It is possible to specify a dip field that contains a scaler, array or an array element. It could look like:

  ```
  yField="OverallMonitor",
  yField="Maxima",
  yField="regB_inputs[1]".
  ```

- "type" is an attribute that defines the type of the plot to be displayed. It has 5 default values that it can be set to, which are described in detail in a table below. It could be set to "history" type, which defines the time based plots. The other types of this attribute are "histogram" and "instant". The specification of those types can be found in Table 2.1
- "reset" is an attribute that resets the histogram any given period of time.

- “axisName” sets the axis name. All axes with the same name will be forced onto the same yaxis in multi-axis view.
- “renderer” sets the renderer to an axis. Allowed you to select from BAR,SCATTER and AREA.
- “ strokeColor” sets the color of a strike. The standard list of colors is available (RED, BLACK, etc.).
- “markerType” sets the marker type from the standard list of marker types.
- “fillColor” sets fill color for the plot area from the standard list of colors.
- “transparency” sets the transparency to the stroke. Values permitted” BRIGHTER, DARKER, LIGHTER (LIGHTER means more transparent).
- “markerSize” sets the size of the marker. Value: Integer (default is 5).
- “YMin” sets minimum Y value. Disables automatic axis calculation.
- “ YMax” sets maximum Y value. Disables automatic axis calculation.
- “logarithmic” sets the scale logarithmic. The base of the logarithm (10 typically).
- “strokeWidth” sets the stroke width. Value: Double (default is 5).

An example of the <Plot> ... < /Plot> tag:

```
<DataView name="Current" axisType="time"
  multiAxis="false">...</DataView>
```

2.2 HTML Configuration file

HTML configuration file is a basic HTML file that contains <applet>< /applet> tags. Its purpose is simply to pass the path to XML file to the DipCacheVIsualizer applet, and then to embed it into a web page. Also it is useful for combining few applets in one web page and other simple HTML operations.

A typical HMLT document should start with the <html> ... < /html> tag and all the following content should go comprised by it. To add an

Syntax	Purpose	Functionality	Usage	Example	Dataset Naming
history:Number of data points to be displayed in a chart	gets the number of data points defined from dipChace server and plots it on a time scale	Time scaler with X axis set to timestamp	It can be used for any single value	history:1000	The name of dataset is the name of the subscription
histogram:Xmin:xMax:Number of Bins	creates histogram within specified range	Histogram that will add new data points to existing data set	It can be use for arrays and a single value	histogram:-3:4:8	The name of dataset: histogram_<the name of the subscription>
histogram:-1	creates histogram with bins from 0 to N	Histogram that adds new data points to existing data set	It can be used for array values ONLY	histogram:-1	The name of dataset: histogram_<the name of the subscription>
instant:xMin:xMax:number of bins	creates histogram within specified range	Histogram that sets new data to dataset per unit of time	It can be use for arrays and a single value	instant:0:100:10	The name of dataset is the name of the subscription
instant:-1	creates histogram with bins from 0 to N	Histogram that sets new data to dataset per unit of time	It can be used for array values ONLY	instant:-1	The name of dataset is the name of the subscription

Fig. 2.1: Different types of histograms.

display to a HTML page <applet>< /applet> tag is used. All applet tags should be placed inside a <body>< /body> element. It is necessary to have a <param/ > tag inside an each applet element. It holds the link to the XML file. All tags used in HTML document are standard and the information about their meaning and usage can be found on official HTML web pages at the w3c.org site

2.2.1 Attributes

In order to run an applet properly it is essential to pay close attention to the attributes settings in HTML configuration file. Detailed description of each attribute is presented below. Applet tag:

- "code" attribute sets the name of java class that creates an applet instance. It should be set to cod="cern.brm.DipCacheVisualizer.class".
- "archive" attribute provides the link to all jar files necessary to run the display.
- "width" and "height" attributes define the size of the display. They can be set to any reasonable number.

Param tag

- "name" attribute set the name of the parameter passed to java applet class. It has to be set to name="config"
- "value" attribute passes the name of XML document to java applet class. For example: value="bsc.xml"

2.3 CSS Configuration file

As it is already mentioned above it is possible to use CSS document within the framework. The way to embed CSS configurations to XML file is to put it directly to XML document inside of <DataView> ... < /DataView> tag.

2.3.1 Possible configuration

The CSS styling supports at the moment 6 different elements (selectors): chart, chartArea, scale, grid, axis and legend. The scale, grid and axis elements may depend on the axis (X or Y). To specify style for selected we use a conditional selector with argument axis set to X or Y e.g.

```
scale[axis='X'] {
  title: 'X coordinates';
}

axis[axis=Y] {
  min: 0;
  max: 10;
}
```

If no axis contition is specified - these elements apply to both: X and Y axis.

```
grid {
  majorColor: #FC123F;
  majorStroke: stroke(2);
}
```

Note that properties defined for a specific axis override properties defined for both axis i.e. properties defined by selector scale[axis=$'$ X'] override these defined by scale selector.

The "dataset" selector requires a condition attribute: name of the data set which is the same with yField name in XML file:

```
dataset[name='Name of Data Set'] {
  renderingType: 'BAR';
}
```

Possible values for the element can be found at

```
http://172.18.203.110/ap/dist/jdataviewer/1.2.0/
        build/docs/api/cern/jdve/package-summary.html
```

2.4 Java Web Start launch file

Java Web Start is a framework developed by Sun Microsystems that allows users to start application software. Web Start applications do not run inside the browser, and the sandbox in which they run need not have as many restrictions, although this can be configured. Web Start has an advantage over applets in that it overcomes many compatibility problems with browsers' Java plugins and different JVM versions. On the other hand, Web Start programs cannot communicate with the browser as easily as applets.

Example of the Java web start launch file for the monitoring framework:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp
  spec="1.0+"
    codebase="http://10.176.29.237/dev/"
    href="launch.jnlp" >
    <information>
        <title>BRM Displays</title>
        <vendor>BRM Team</vendor>
    </information>
     <security>
      <all-permissions/>
  </security>
    <resources>
        <!-- Application Resources -->
        <j2se version="1.0+" />
        <jar href="brm-wbm-1.0.3-SNAPSHOT.jar" />
        <jar href="lib/jdataviewer-compat.jar"/>
        <jar href="lib/dip.jar"/>
        <jar href="lib/spring.jar"/>
        <jar href="lib/xercesImpl.jar"/>
        <jar href="lib/httpclient.jar"/>
        <jar href="lib/httpcore.jar"/>
        <jar href="lib/commons-logging.jar"/>
        <jar href="lib/log4j.jar"/>
        <jar href="lib/flute.jar"/>
        <jar href="lib/sac.jar"/>
        </resources>

    <application-desc
        name="BCM1L Display"
      main-class="cern.brm.AppletWrapper">
<argument>
        http://10.176.29.237/dev/bcm1LX.xml</argument>
</application-desc>
</jnlp>
```

The Jnlp tag is responsible for basic configurations. The "spec" defines the java version to use. "codebase" attribute specifies the path to the jnlp file, and "href" set the name of the jnlp file.

All the jar that are required by the application should be given inside

the <resourses> .. < /resourses>. The jar archives should be signed.

The <application-desc> .. < /application-desc> tag indicates the entry Java class with "main-class attribute. There are two classes that can be used to create web start display. One is called AppletWrapper class. This class is used to produce the display that contains the DipCacheVisualizer Applet only. It takes a single parameter, which specifies the name of the XML configuration file. AppletLoader is another class that can be used to produce the web start display. This class is capable of combining few header applets and the DipCacheVisualizer applet into one display. It takes a list of parameters, that indicate the names of the XML configuration files for the applet. Note, that the first parameter has to define the XML file that is responsible for the display layout. By default the AppletLoader grabs the predefined layout config file called Swix.xml.

Swix.xml file describes the layout of the display. It has a custom xml tags called "visualizer" and "header" that specify the place of the visualizer applet and header applet.

2.4.1 Layout configuration file

The default layout for the monitoring framework is defined in Swix.xml. But it is essential to understand the structure of this file in order to change the layout for a display.

An example of the layout configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<frame id="f" size="1000,800" layout="BorderLayout">
   <panel constraints="BorderLayout.PAGE_START"
                  layout="GridLayout(0,2)>
        <head id="head"  visible="true"/>
        <head id="head1"  visible="true"/>
   </panel>
        <vis id="vis" constraints="BorderLayout.CENTER"
                  visible="true"/>
</frame>
```

<frame>< /frame> tag defines a display. It has "size" and "layout" attributes. For the layout specifications, please, go to

```
http://java.sun.com/docs/books
        /tutorial/uiswing/layout/using.html
```

The header applet has to be placed inside the <panel> .. < /panel> tag. It is important to follow the naming standards. "id attributes have to

be specified as “head”, “head1”, “head2”, etc up to “head5”. The <vis/ > tag is responsible for placing the DipCacheVisualizer applet to display. No modifications are needed to this attribute. There is only one <vis/ > tag per layout configuration file!

3. HEADER DISPLAY CONFIGURATIONS

A header is an applet that is capable of displaying simple status messages such as text or numbers. To simplify the monitoring of the status messages, the NumberMonitoring applet has the functionality that applies certain color schemes to the applet's background or foreground. For example, the normal background color for the header is white, but if status message contains the word "ALARM", the background of the header switches to red.

3.1 XML configuration file

The configuration file has a clear structure that must be followed. <header>< /header> is the main container tag. All the following elements should be placed inside it. The attributes:

- "name" specifies the name of the applet.
- "baseUrl" is the path to the Dip-Cache server.

<subscription>< /subscription> defines the dpi subscription. There could be more than one subscription tags inside of the header. The attributes:

- "url" provides full path to the dip subscription.
- "state" is a flag that specifies if the status field in the dip subscription should be printed to the header. Possible values: true, false.
- "titleSize" helps to use the space inside of the header more efficient. It modifies the amount of space that is provided for the header title.

<dipField>< /dipField> tag specifies the name of the dip filed. The attributes:

- "name" is the name of the field that will appear on the screen.
- "field" is the name of the dip field in the subscription.

An example of XML configuration file for Header display:

```
<?xml version="1.0" encoding="UTF-8"?>
<header name="BEAM Summary"
          baseUrl="http://10.176.29.237:8080/brm-wbm/" >
<subscription url="dip/CMS/BRM/BRMSummary/Acquisition"
                    state="true" titleSize="300">
         <dipField field="intensityBeam1"
                           name = "Beam1_Intensity"/>
         <dipField field="intensityBeam2"
                           name = "Beam2_Intensity"/>
         <dipField field="BunchCrossingDeltaT"
                  name = "Beam1-2_DeltaT(nS)"/>
</subscription>
</header>
```

4. FRAMEWORK ARCHITECHTURE

4.1 Plotting Package

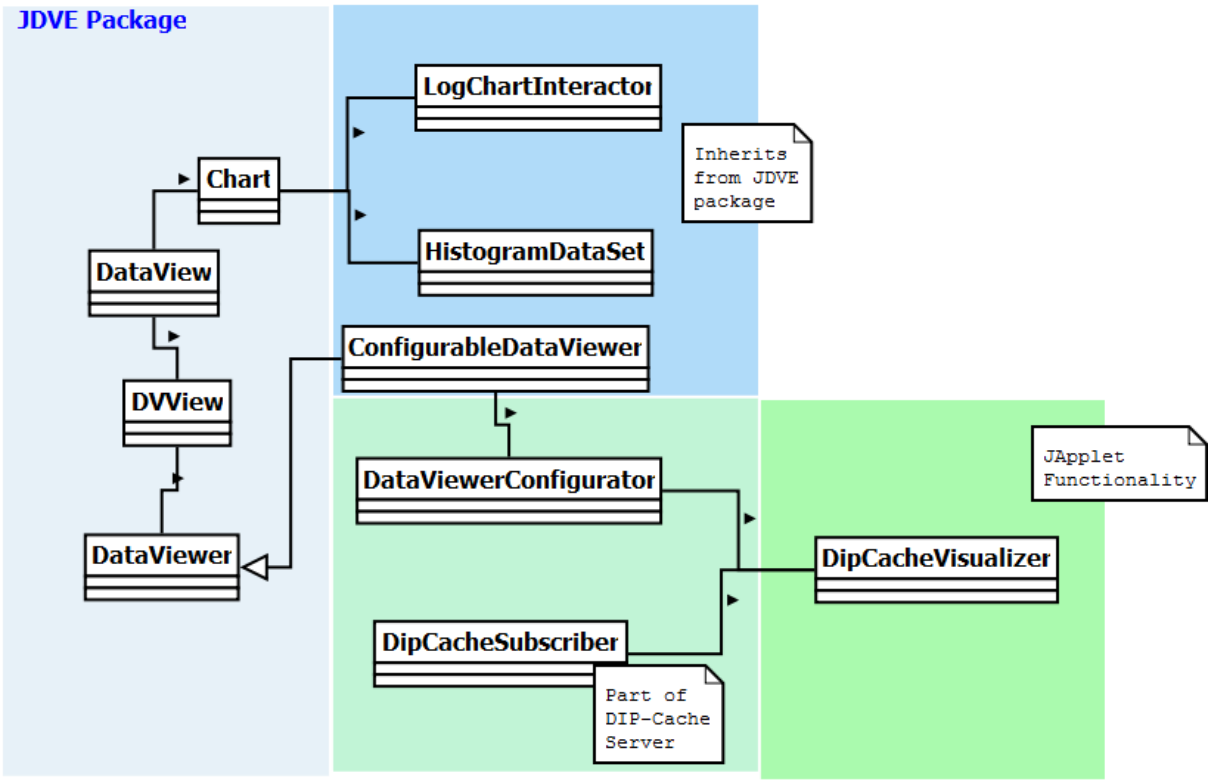


Fig. 4.1: The Class diagram that reflects the dependencies inside the framework.

The framework is developed around four main classes, but is also includes a number of auxiliary ones.

DipCacheVisualizer is an entry point class. It inherits from Java Applet class and is responsible for integration of the monitors based on JDVE

package to Java applet. The class contains the private class named DataGenerator, which implements DipSubscriptionListener interface, that is defined inside of the dip package. The enternal class handles new data coming from DIP-Cache and puts it into a proper data-set. To retrieve the data, DataGenerator class has a method called extractData(DipData data, String field, int index), which extracts DipData into a suitable format.

In order for DataGenerator class to perform its task correctly, it is critical to have a robust and easy way to match a DataSet object to a DipSubscription. To meet this goal DipCacheVisualizer class has a private Hash map that matches the keys represented in String format to object arrays that contain a DataSet object and other relevant information.

Another important step that is crucial for the visualization framework is embedding a monitor inside of a Java Applet object. Inside of the DipCacheVisualizer it is done by the createChart() method, which makes a call to DataViewerConfigurator.configureDataViewer (String XMLFile) static method. DipCacheVisualizer gets the path to XML configuration file as a parameter from the HTML page and parses it to DataViewerConfigurator class that is capable of reading XML. The path to XML configuration file should be valid and reachable in order the framework to complete the task successfully. If the configureDataViewer() method fails to return an object the program can not continue executing.

DataViewerConfigurator.configureDataViewer(String XMLFile) method returns ConfigurableDataViewer object that is assigned to the internal field inside of DipCacheVisualizer. At this point the Hash map inside of ConfigurableDataViewe gets retrieved and reassigned to the local Hash map. At the same time for each data set new DipCahceSubscriber object is created that makes a call to DIP-Cache and gets the latest data, and then passes it to DataGenerator class.

Rendering XML is done within DataViewerConfigurator class. Since XML files can get rather big and complicated, but at the same time all configuration files have the same structure, Simple API for XML(SAX) technology is used. The DataViewerConfigurator has a static method configureDataView(String xmlUrl), which play the role of entry point to the class. This method performs two important operations: first it loads and parses XML document, and also creates an instance of ConfigurableDataViewer class that gets shaped while XML execution and then returned to the DipCacheVisualizer class.

ConfigurableDataViewer is derived from DataViewer class that is the part of JDVE package. It contains a private map that lists datasets to the object array, which is very important process for data transport inside

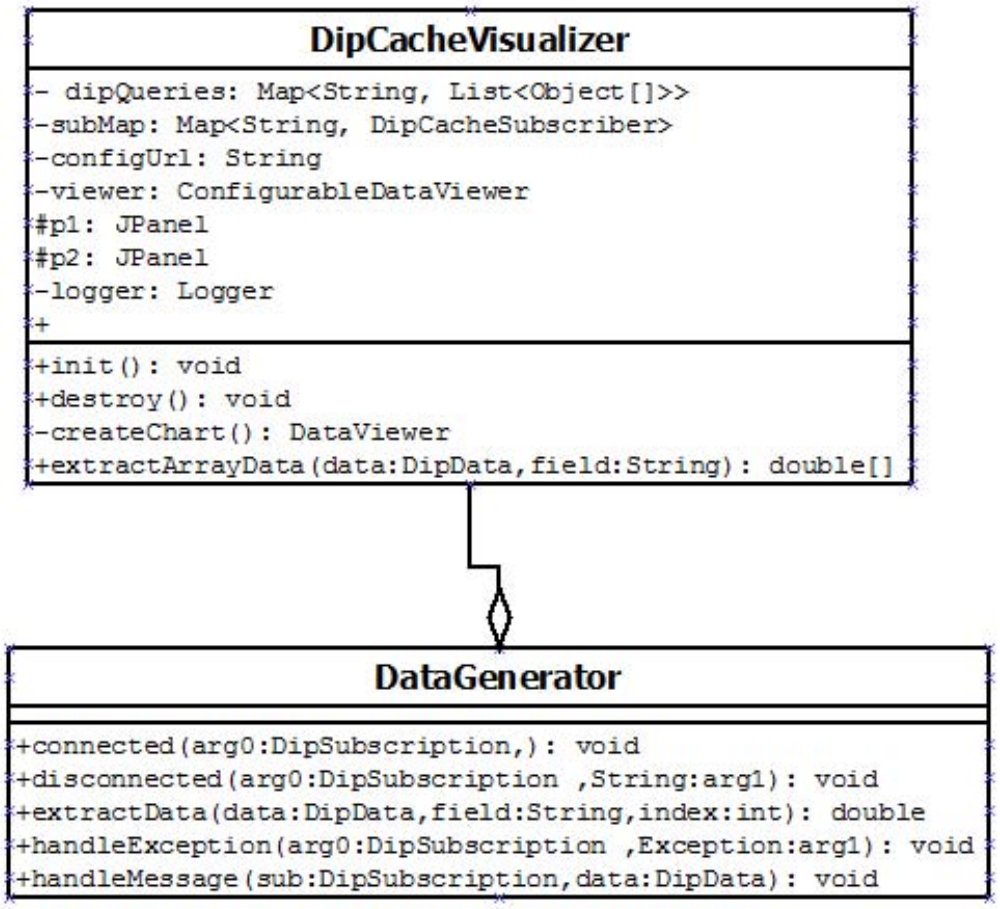


Fig. 4.2: DipCacheVisualizer.

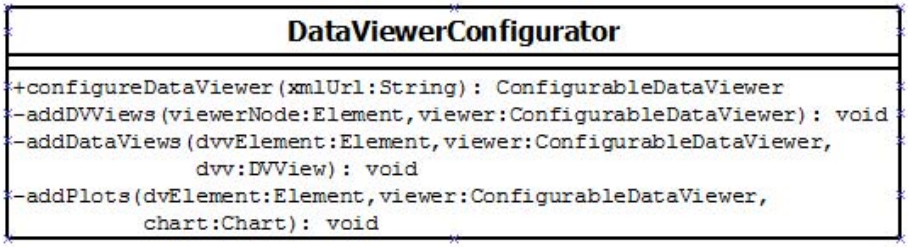


Fig. 4.3: DataViewerConfigurator class.

of the visualization framework. This map is used as a key inside of the DipCacheVisualizer class to match Dip Subscriptions to Dataset objects.

Also ConfigurableDataViewer is capable of making difference between a single field, array data or a certain field inside array. It is important to make to differentiate between data types in order to use suitable data extraction method and avoid data losses.

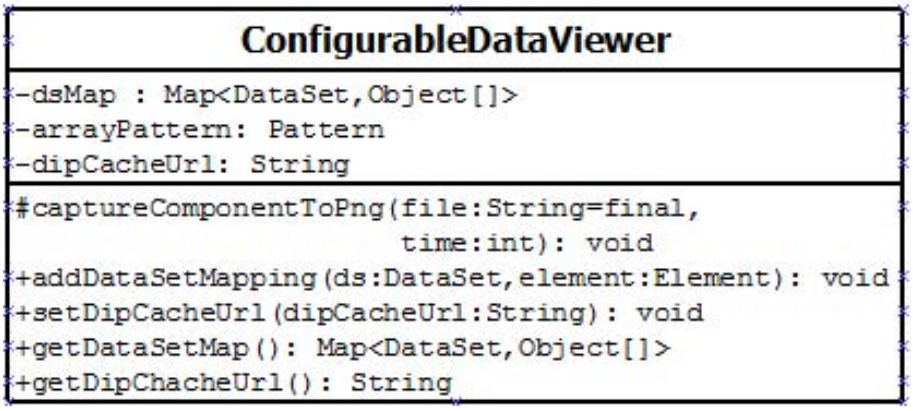


Fig. 4.4: ConfigurableDataViewer class.

4.2 Header Package

To simplify the monitoring of the status messages, the NumberMonitoring applet has the functionality that applies certain color schemes to the applet's background or foreground. For example, the normal background color for the header is white, but if status message contains the word "ALARM", the background of the header switches to red.

The design and implementation of the Header Applet is almost identical to the DipCacheVisualizer Applet. The only basic difference is that the "ReadHeaderXML" class, that parses XML configuration files, creates an object called "header" and then parses it to the header applet. Java Spring class was used to create a header object from an XML file.

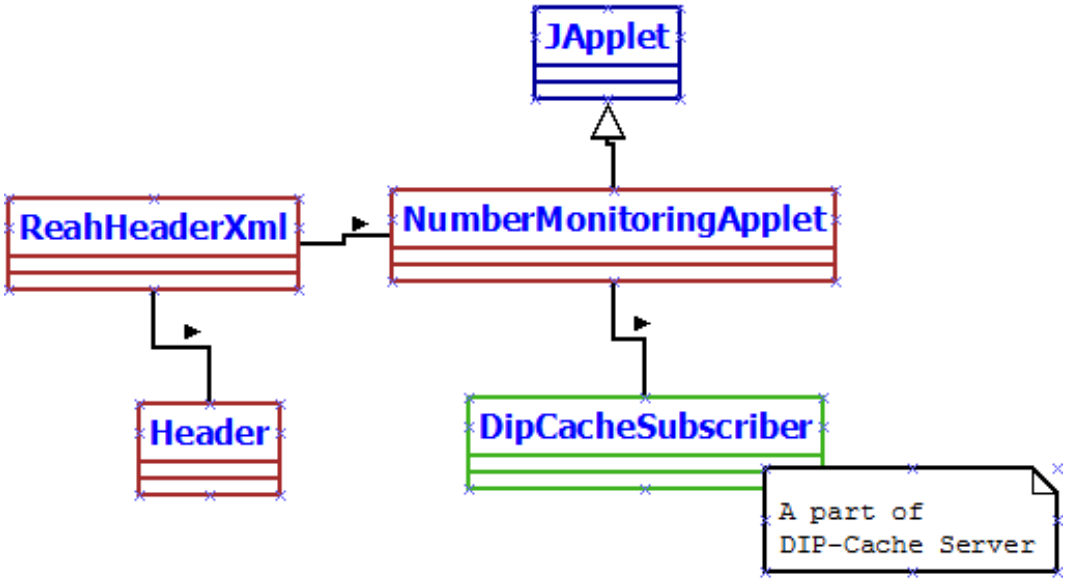


Fig. 4.5: Class Diagram that represents the relations of Header package.

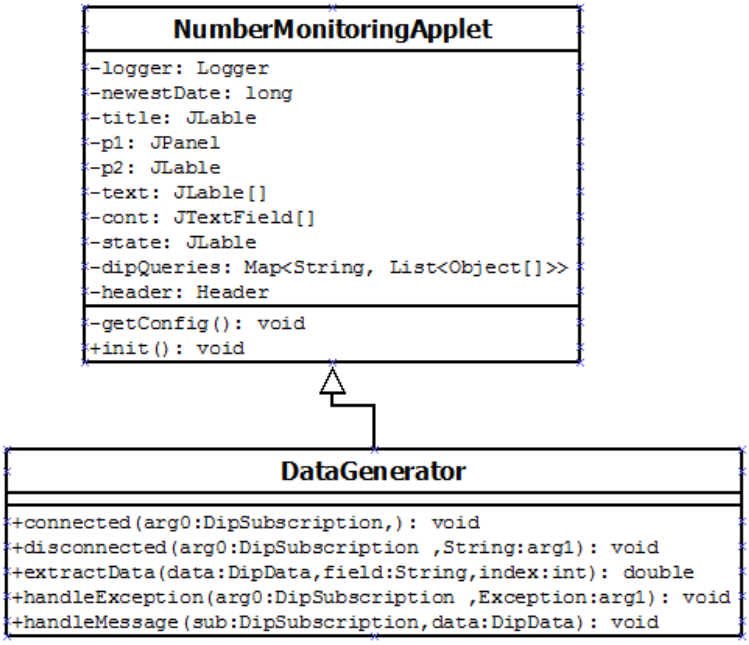


Fig. 4.6: NumberMonitoringApplet Class.

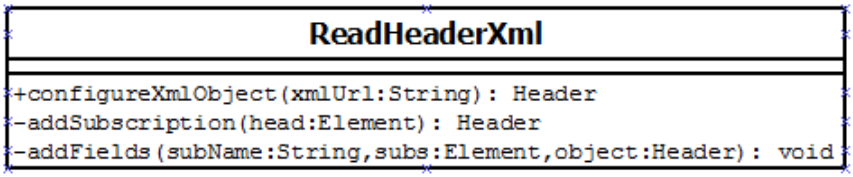


Fig. 4.7: ReadHeaderXml Class.

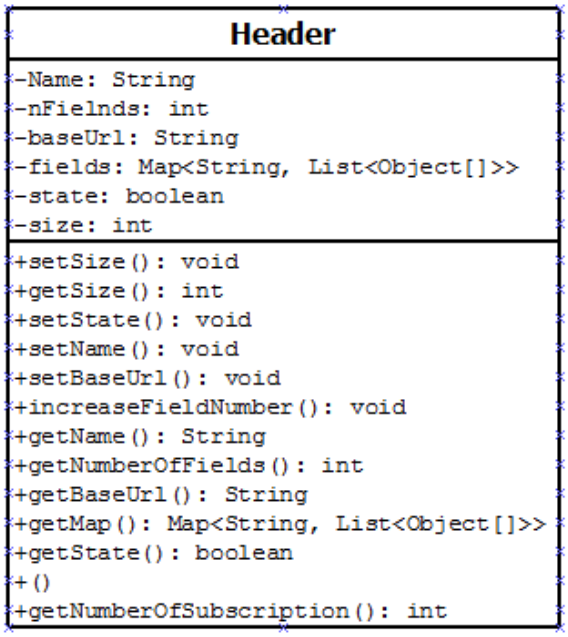


Fig. 4.8: Header Class.

5. STATUS FORM

5.0.1 Architecture

The status form is meant to be a simple Java frame that is capable of display boolean values, which represent the status of the certain equipment. It is completely separate from DipCacheApplet and HeaderAppler, even thought it inherits the basic structure from them. The main difference that it can be used only with Java web start technology, because the main class of the package (called StatusFrom) doesn't inherit from the Java Applet class, but from JFrame class, which makes it difficult to embedded to the HTML page.

The package that contains the class, that parses XML configuration files for the form, and the supporting class is called StatusXMLReader. The basic architecture idea is very simple: user creates the XML configuration file and the java web start launch file, then the framework parses XML, subscribes to the defined DIP subscription and creates a simple Java Frame. The relationships of the classes inside of the framework are represented in the class diagram 5.1

5.1 Configuration files

The same as with other XML configuration files the structure of the document has to be followed.

<status>< /status> is the main container tag. All the following elements should be placed inside it. The attributes:

- "name" specifies the name of the frame.
- "baseUrl" is the path to the Dip-Cache server.

<subscription>< /subscription> defines the dip subscription. There could be more than one subscription tags inside of the header. The attributes:

- "url" provides full path to the dip subscription.

<dipField>< /dipField> tag specifies the name of the dip filed. The attributes:

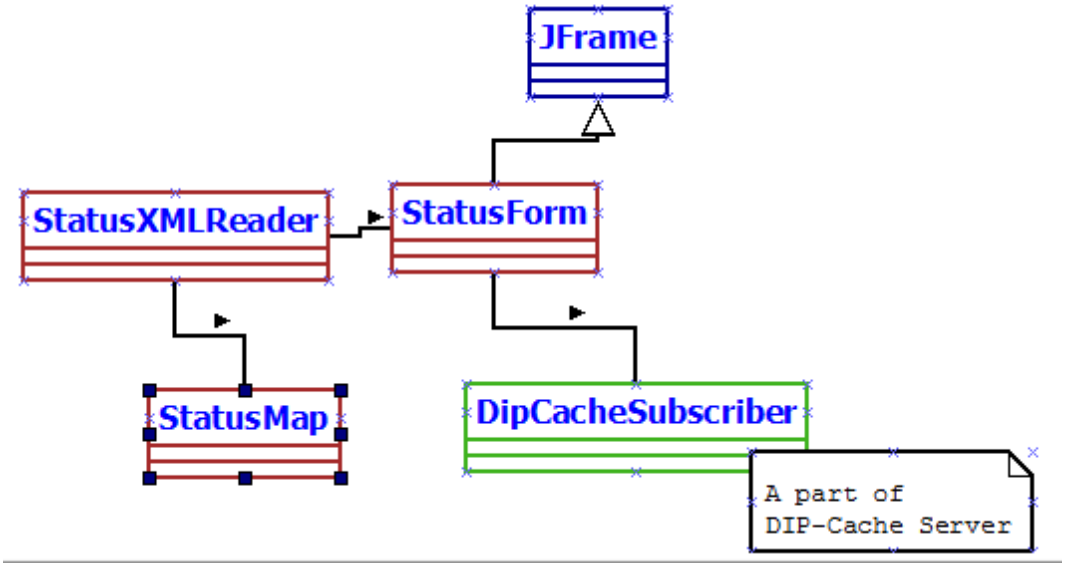


Fig. 5.1: Status Form class diagram.

- "name" is the name of the field that will appear on the screen.
- "field" is the name of the dip field in the subscription.
- "Okstate" represents the safe state of the equipment.

An Example of an XML configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<status name="PLC Crate"
        baseUrl="http://10.176.29.237:8080/brm-wbm/" >

<subscription url="dip/CMS/BRM/PLCcrate/Status" >
        <dipField field="input(inject_permit)"
                name="Injection Inhibited"
                OkState="false"/>
        <dipField field="station_A"  Okstate="true"/>
        <dipField field="station_B"  Okstate="true"/>
</subscription>
</status>
```

The Java Web Start file is a standardized example of jnlp launch file. The detailed description how to implement Java Web start for the real-time monitoring framework can be found in the section 2. The main class for creating Status forms is StatusForm, that accepts only one attribute, which is the path to the XML configuration file.

An example of Java Web Start configuration file:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp
  spec="1.0+"
    codebase="http://10.176.29.237/dev/"
    href="launchPLCCrateStatus.jnlp" >
    <information>
        <title>BRM Demo</title>
        <vendor>BRM Team</vendor>
    </information>
     <security>
      <all-permissions/>
  </security>
    <resources>
        <!-- Application Resources -->
        <j2se version="1.0+" />
        <jar href="brm-wbm-1.0.3-SNAPSHOT.jar" />
        <jar href="lib/jdataviewer-compat.jar"/>
        <jar href="lib/dip.jar"/>
        <jar href="lib/spring.jar"/>
        <jar href="lib/xercesImpl.jar"/>
        <jar href="lib/httpclient.jar"/>
        <jar href="lib/httpcore.jar"/>
        <jar href="lib/commons-logging.jar"/>
        <jar href="lib/log4j.jar"/>
        <jar href="lib/flute.jar"/>
        <jar href="lib/swixml.jar"/>
        <jar href="lib/jdom.jar"/>
        <jar href="lib/sac.jar"/>
        </resources>
    <application-desc
          main-class="cern.brm.StatusForm">
<argument>
        http://10.176.29.237/dev/PLCCrateStatus.xm
```

```
</argument>
      </application-desc>
</jnlp>
```

6. ENVIRONMENT CONFIGURATION

The framework development is done with the help of the following software tools:

- Eclipse software development environment.
- Maven building tools.
- Subversion as a version control tools.

6.1 Start up

At the starting point, you need to have the project code. It can be acquired with by using the Subversion command:

```
svn co svn+ssh://<username>@svn.cern.ch/reps/cmsbrm/trunk cmsbrm
```

This creates the directory cmsbrm under the user's home and check the cmsbrm project code to the cmsbrm folder.

NOTE: The command requires the password to the Subversion repository.

For additional information about Subversion, please, visit:

```
http://subversion.apache.org/docs/
```

Next step is to source the BRM enviroment variables, that can be done using the script brm-env-cms.sh under cmsbrm/interfacescripts.

The project is originally built using Maven building tool, so it is possible to rebuild it with Maven. First, it is necessary to get the project resources.

```
cd cmsbrm
mvn dependency:resolve
```

After the dependencies are downloaded to the user's home directory, the following commands could be used:

To build the package:

```
mvn package
```

To clean files and directories generated by Maven during its build:

```
mvn clean
```

For additional information on Maven functionally, please, visit

```
http://maven.apache.org/
```

6.2 Eclipse configuration

6.2.1 Importing the project

In the File menu, choose “Import” and in the Select Window that appears, in the “General” menu choose “Existing projects into workspace”. Click “Next”, and in the following window choose ”Select root directory” and click “Browse”, see Figure 6.1

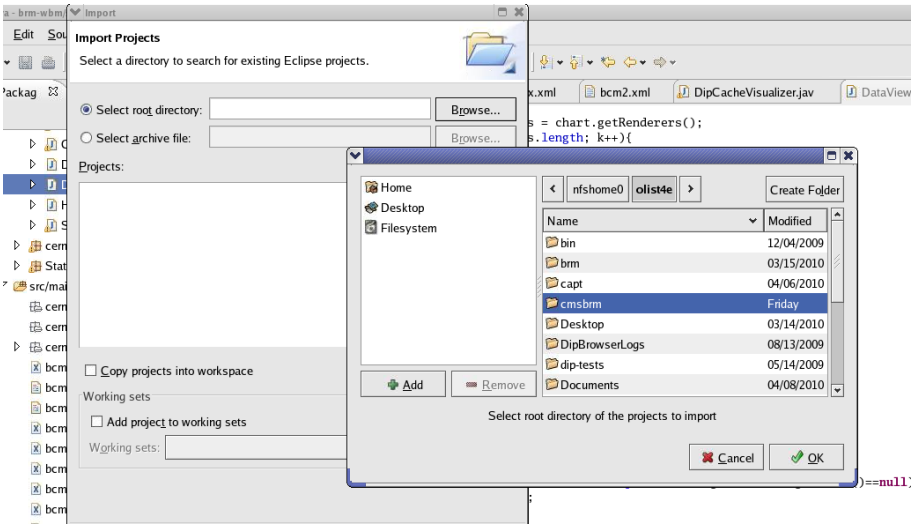


Fig. 6.1: Importing project.

Find the cmsbrm project and press “Finish”.

6.2.2 Environment variables

After the project is imported to Eclipse, it is necessary to configure the classpath variables. For the cmsbrm project, the M2_REPO variable should be set. To do so, go to the “Window” menu and choose “Preferences”. In the

window, that will appear, go to Java –> Build Path –> Classpath Variables. Then click "New" and enter the following information, as in Figure 6.2:

- Name: M2_REPO.
- PATH: <path to the user's home directory> /.m2/repository.

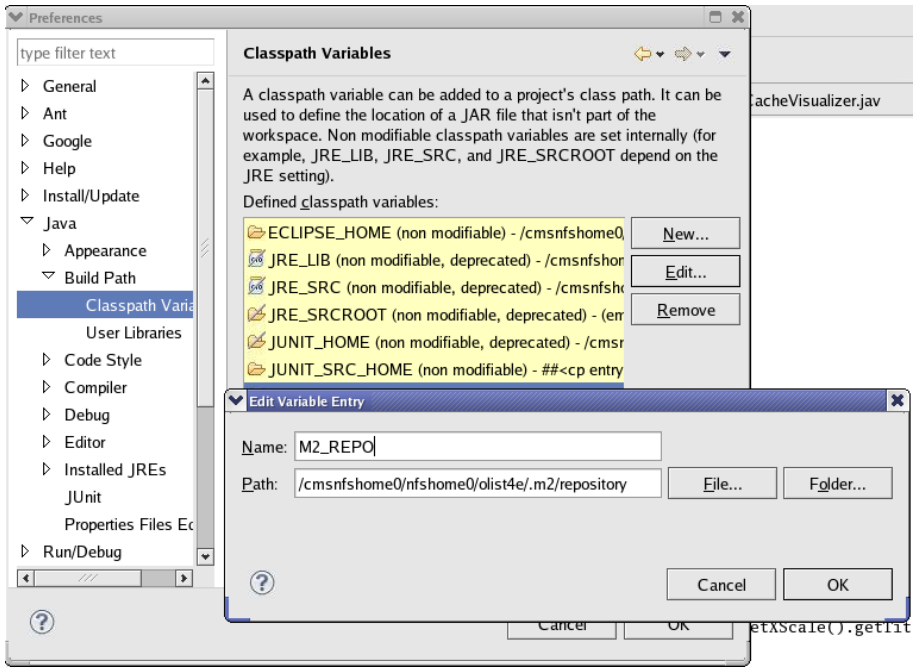


Fig. 6.2: Classpath variable definition.

Once the Maven classpath variable is set, it is necessary to run the Maven Eclipse command:

```
mvn eclipse:eclipse
```

If all the steps mentioned above are completed correctly, it should be possible to start working with the framework code that can be found at cmsbrm/displays/brm-wbm.

6.3 Building and deploying the project

The monitoring framework should be built in the brm-wbm directory with the command:

```
mvn package
```

The package command will create a jar file, which is the framework main resource. To create a binary distribution from the project that includes supporting scripts, configuration files, and all runtime dependencies or in other words to make a tar.gz file in the target directory in the brm-wbm project, use:

```
mvn assembly:assembly
```

To deploy the new version of the package, the brm-wbm tar.gz file should be copied to the node1 machine to the directory /var/www/html/dev. In order to see the changes, it is necessary to extract the file with the command:

```
tar -xzf <brm-wbm package name>.tar.gz
```

The final step of deploying the new framework version is the jar signing. Java web start framework requires to sign all the jar files that the framework is using. It could be done manually or with the help of the simple script in dev directory. The name of the script is signJars.sh and it can be executed as bash command. Note, that the jar signing process requires the passwort!

To make sure that the changes are applied successfully, run

```
javaws <name of the file>.jnlp
```

7. CONCLUSIONS

At the current time the monitoring framework is used by Beam and Radiation monitoring group and also tracker group, pixel group, Run Filed Manager, Shift Leaders, Technical shifters, CSC, and Luminosity group. The visualization monitors as a real-time system reflects the beam conditions in a real time with the low latency level, thus it is the first place at the CMS detector where the beam collisions are observed. Thereby it is essential to keep the monitors running all the time to provide stable 24/7 service. The crash rate of the system is low, but in case of failure, the monitors should be immediately restarted. BRM team constantly runs the set of seven displays, that uses approximately 18 DIP subscriptions. The monitors provide clear human-readable information for the real-time beam condition analysis at the Control room. They reflect the status of such BRM detector as BCM1l, BCM2, BSC and BCM1F, and also provide the summary information.

BIBLIOGRAPHY

[1] E. Lyndon and P. Bryant, "The CERN Large Hadron Collider: Accelerator and Experiments," *Journal of Instrumentation 3*, vol. 3, 2008.

[2] S. Chatrchyan, "The CMS experiment at the CERN LHC," *Journal of Instrumentation 3 S08004*, 2008.

[3] A. J. Bell, "Beam and Radiation Monitoring for CMS," in *Nuclear Science Symposium Conference Record*, no. 2322 in 3, IEEE, 2008.

[4] M. Peryt and M. M. Marquez, "Fesa 3.0 : Overcoming the XML/RDBMS impedance mismatch," *12th International Conference On Accelerator And Large Experimental Physics Control Systems*, vol. 4, 2009.

[5] A. Macpherson, "Beam condition monitoring and radiation damage concerns of the experiment," *Proceedings of the XV LHC Project Chamonix Workshop*, vol. 5, pp. 258–263, 2006.

[6] C. Pignard, "Online radiation monitoring for the LHC machine and experimental caverns," *Journal of Instrumentation 3*, p. 12, 2006.

[7] D. Chong, "Validation of synthetic diamond for a beam condition monitor for the compact muon solenoid experiment," *IEEE Trans. Nucl. Sci.*, vol. 54, pp. 182–236, 2009.

[8] B. Dehning, "The Beam Loss Monitor system," *Proceedings of the XIII LHC Project Chamonix Workshop*, vol. 3, pp. 2667–2670, 2004.

[9] E. Effinger, "The LHC Beam Loss Monitoring system's data acquisition card," *Proceedings of LECC*, vol. 5, pp. 108–112, 2006.

[10] C. Zamantzasl, "The LHC Beam Loss Monitoring system's surface building installation," *Proceedings of LECC*, vol. 4, pp. 552–556, 2006.

[11] K. Kostro, "The controls middleware (CMW) at cern status and usage," *Proceedings of ICALEPCS*, vol. 4, 2003.

[12] B. Eckel, *Thinking In Java*. Upper Saddle River, NJ: Prentice Hall PTR, 2006.

[13] CERN, *Java DataViewer Documentation*, 2009.

[14] Sun, *org.xml.sax Documentation*, February 2010.

[15] CERN, Fermilab, *scientific linux official web site*, April 2010.

[16] M. Hollingsworth, *Generalized Unilateral Transfer Server Framework*. CMS Detector note.

[17] M. Hollingsworth, "DIP Cache Documentation." in preparation.

[18] O. Filyushkina, "Integrating and automating the software environment for the Beam and Radiation Monitoring for CMS," 2010.