Remote View – Developer guide

F.Momal

CERN / div. LHC

Introduction

RemoteView is a MMI (Man Machine Interface) system in Java dedicated to real-time data visualization. The starting idea was to define a set of interfaces between the different parts of such a system. It would be then easy to share network drivers, graphical components, window manager, etc between developers. The main objective was to keep those interfaces as simple as possible while avoiding to loose possible functionality. We also wanted to benefit from all the new technologies brought by Java as well as the easiness of today windowing interfaces. We wanted to go a step further than those constraining applets we see everywhere.

The core of such a system is some kind of real-time database maintaining the current status of the tags (a tag here is some process variable defined by a value and a set of associated data such as the acceptable limits, the units, etc.). We defined a first interface, which we called **TagArrays**. This TagArrays holds the definition of a tag (metadata), its last known value and its recent history.

Data servers fetch, usually through the network, the current values of the defined tags and update the TagArrays accordingly. An interface for those data servers have been defined and called **DataServer**. A DataServer is usually a network driver of some sort, but it could be a sequencing machine, a process simulation program, etc.

The user sees the values of the tags in some graphical representation. It could be a trend curve, a process control synoptic, a set of dials, a table or whatever strikes the developer's imagination. An interface for those representations have been define and called **TagsView**. Those TagsViews normally work in an event driven way, which means that they wait either for a change of the value of the tags or for a user input.

The TagsView must be inserted in a container of some sort through which the user may interact with a TagsView to do such things as resizing a view, opening a new view, etc. A minimum interface for these containers have been defined and called **ViewManager**.

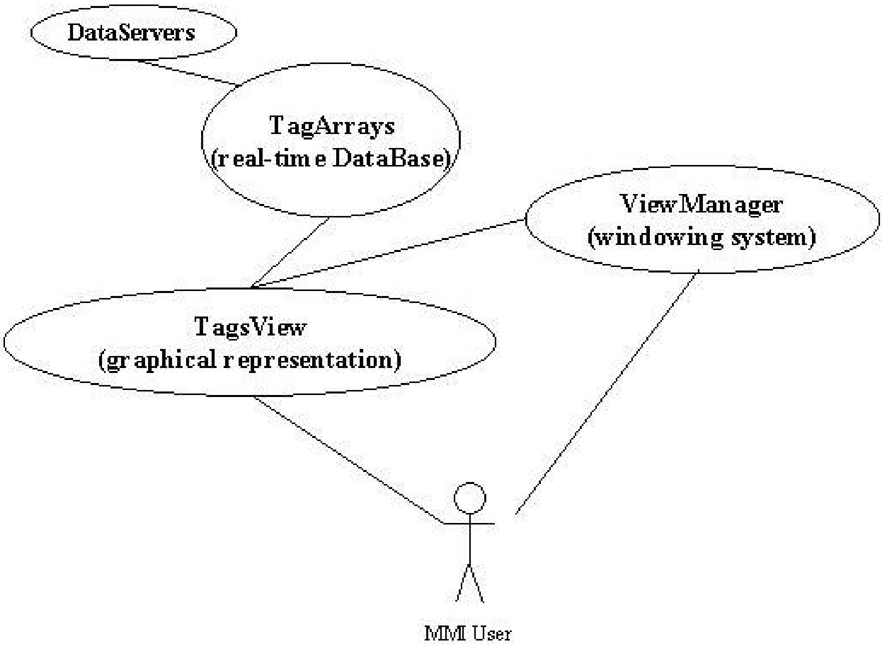


Figure 1: The general organization of the system

Implementations of those interfaces have been developed and are proposed her. Different DataServers exist to access data through a socket-based protocol, through the http protocol or through the MD2S® protocol (a product made by Indium International). The socket driver is event driven.

Different TagsViews have also been developed. One of them is a **dgsl** (Dynamic Graphic Scripting Language – see appendix 1) interpreter. This simple language enables the user to quickly describe the behavior of a graphical view and to run it. The dgsl interpreter reads the script, do the necessary settings and then waits for a change on one of the tags named in the script. Whenever a change is noticed (in the case of the TagsView, this will be done when a TagChange event is received – see below), the entire script is reexecuted so that the view represents the current state of the real-time database.

An implementation of the TagArrays has been developed (TagContainer).

FrameViewMgr, which is an implementation of ViewManager, offers an up-to-date windowing environment to the user, with a toolbar on the top and a tree-like representation of the TagsView on the side of the window.

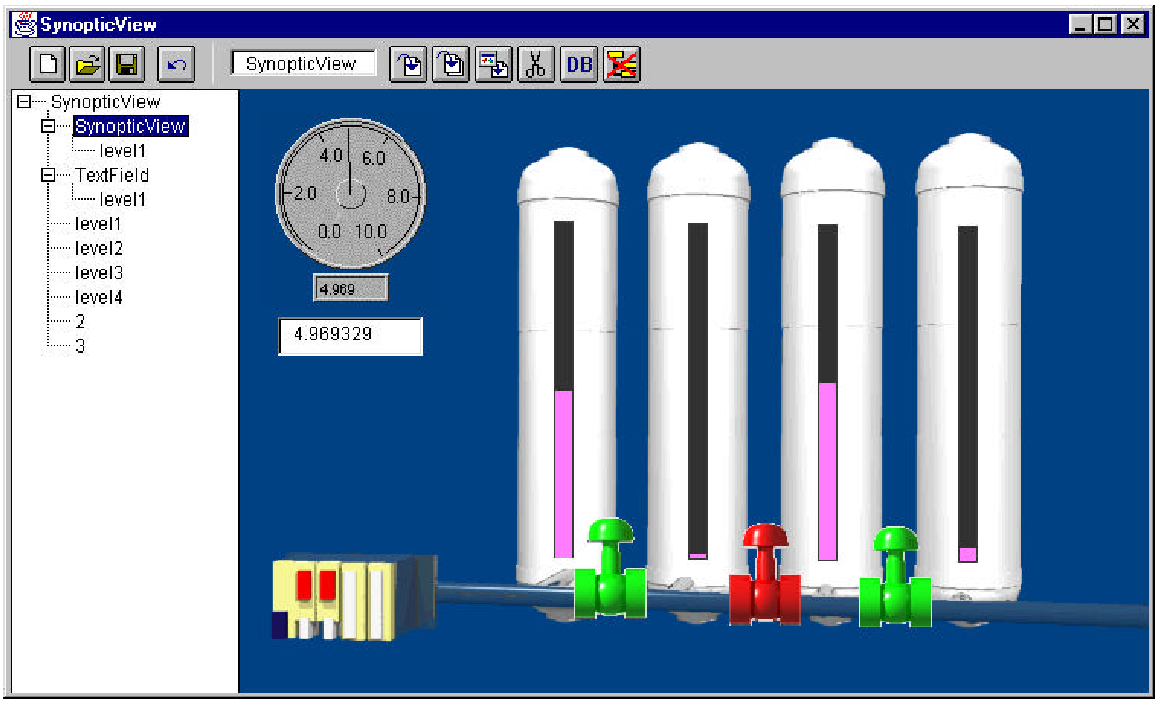


Figure 2: The FrameViewMgr in action

Three important classes have also been developed: componentsType, DataServerManager and configurationMgr. The class componentsType is a TagsView component loader. Given the name of the component, it will automatically fetch the component from a remote host (a Web server or a database for example. So all the rest of the interface doesn't need to know about the existing TagsViews and where they are hosted. DataServerManager do the same for the DataServers and configurationMgr handles all the process of saving and loading TagsView configurations avoiding TagsView to bother about remote file systems, etc.

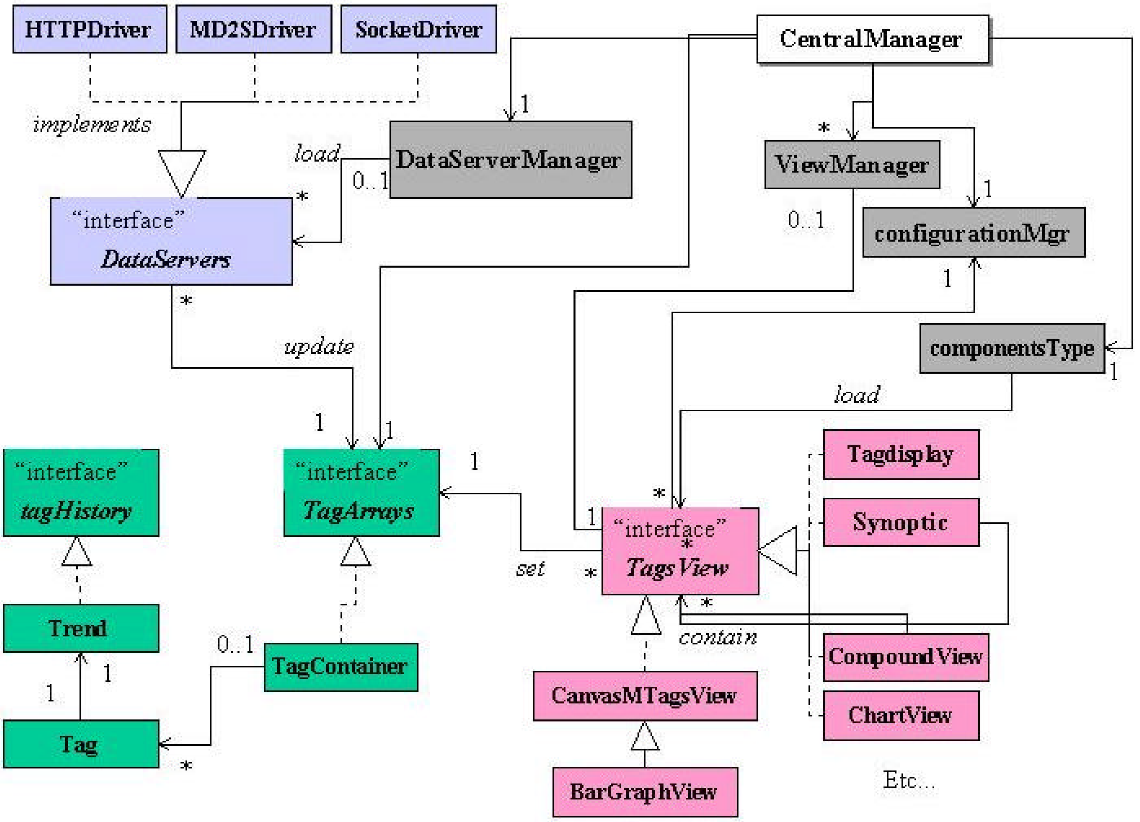
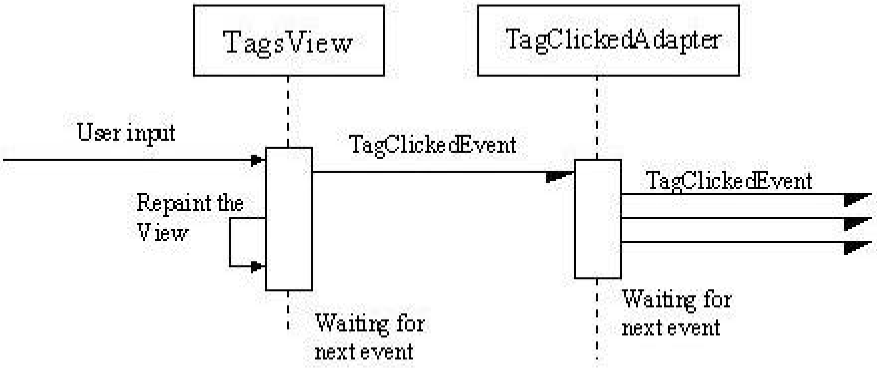
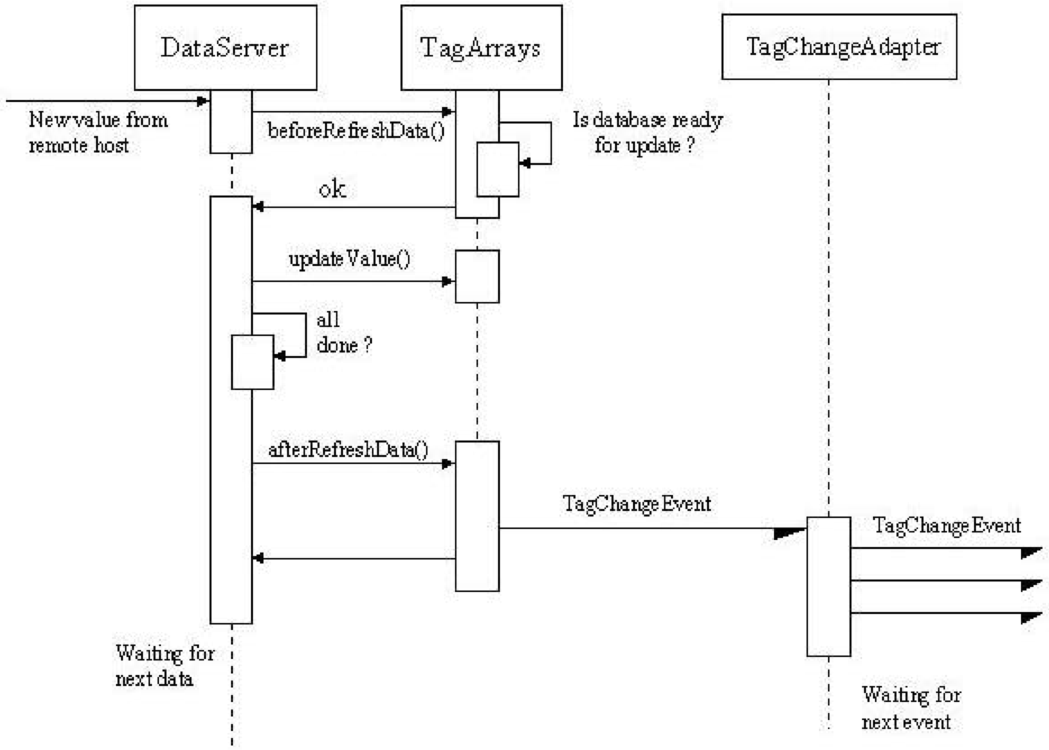


Figure 3: RemoteView classes

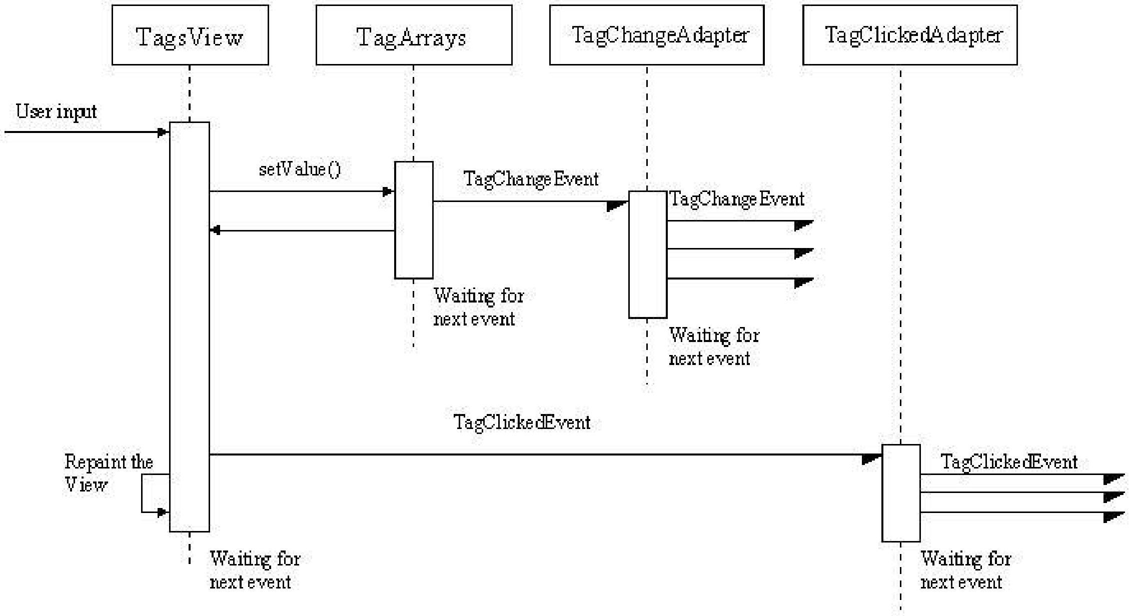
Different kinds of events are passed between the different parts of the system. **TagChangeEvent**s are sent by TagArrays whenever a tag’s value changes. This is done via a TagChangeAdapter. Classes wishing to be alerted when some specific tags change must subscribe for these tasks to the TagChangeAdapter.
TagClickedEvents are sent by TagsViews whenever the user clicks on a tag. The TagClickedAdapter handles the broadcasting of this event.



Interactions between TagsView and TagClickedAdapter

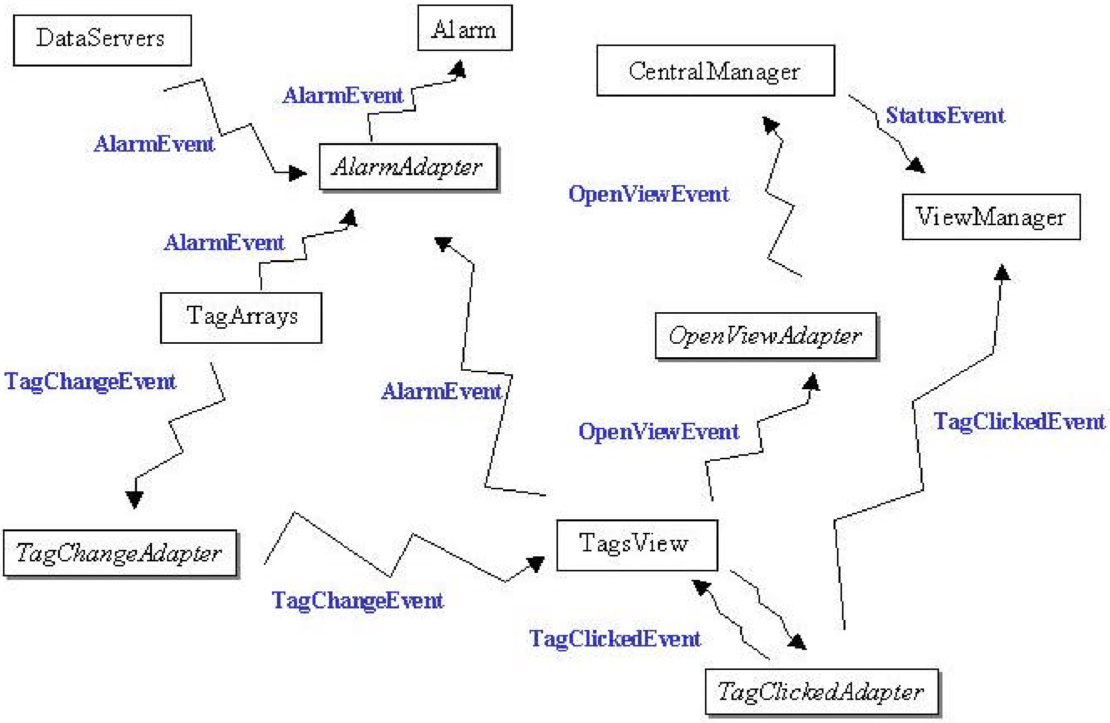


DataBase update by DataServers



User Modification Action

OpenViewEvents are sent by TagsViews wishing to open a new window containing a TagsView. **AlarmEvent**s are sent whenever an abnormal situation occurs.



Event Propagation

DataServers – Network drivers

interface DataServer

public interface **DataServer**

This interface describes a driver used to access the value of the Tags.
The case when the way to send the tags' values is different than the one to read
the values has been taken into account (setReadHost and setWriteHost,...).

If the driver works in a pooling way (or if it has been set to pooling via setType
for example, then a call to enable() will launch the pooling mechanism
(if pooling_interval is set to a value different from -1).
In that case, the driver must know about the TagArrays where to send the data.

DataServer may be used without TagArrays: the following methods can be used without
setting TagArrays:
public void write(String name, String stringValue);
public String read(String name);

Each instance of the driver is named using a "Process Name". So the user may indicates
to which process corresponds a Tag.

The config which is returned by toString and that can be sent to setConfig has the
following format:
ProcessName;Protocol;Type;ReadHost;ReadPort;WriteHost;WritePort;PoolFrequency;subscriptionMode[;
param1=val1[;...]]
ex: String;Socket;OnChange;myHost;3000;;;-1;true

DataServers may send AlarmEvents in case of errors.

class HTTPDriver

class HTTPDriver implements Runnable,**DataServer**

This class is an implementation of DataServer using the http protocol. The URL used to fetch the data is passed to the driver with the method setReadHost.
Ex : setReadHost(« http://www.mydomain.ch :8000/ cgi-bin/getTagsValue»)
The method setWriteHost must be called if data have to be sent back across the network. It is not needed to call setReadPort and setWritePort, the port number being passed in the Host String.
To start a pooling mechanism, one has to set the pooling interval and then call the method enable().
There is one parameter which may bet set with setParameter. It is a TagNamesInURL which is a boolean.
Ex : setParameter(« TagNamesInURL », « true »)
If this parameter is set to true, then the names of the requested tags will be appended to the URL in the following way :
Ex : http://www.mydomain.ch :8000/ cgi-bin/getTagsValue ?tags=tag1,tag2

class MD2Sdriver

class MD2SDriver implements Runnable,**DataServer**

This class is an implementation of DataServer using the API from the MD2S® product by Indium International.

Only one communication channel is needed to read and write data, so it is not necessary to call setWriteHost and setWritePort.
To start a pooling mechanism, one has to set the pooling interval and then call the method enable().

class Socketdriver

class SocketDriver implements Runnable,**DataServer**

This class is a socket based network driver. It implements
the interface DataServer. It works either on a pooling base or
in an event driven way. By default it works on an event driven way.
The protocol used is the following:
1/To get the values of tags, the following command is sent
GET tagname1,tagname2,...\r\n
The driver will then wait for the value. The returned string is
assumed to have the following format:
tagname1=value1,tagname2=value2...
2/To subscribe to some tags in an event driven way, the following
command is sent
SUB tagname1,tagname2,...\r\n
It then wait for any changes on the data. The server has the
responsibilty to send the changes.
Whenever the subscription tags' list has changed, a new list is sent which
replaces the previous one.
3/To end the subscritpion mechanism, the following command is sent:
UNS
4/To set tags' values, the following command is sent
SET tagname1=value1,tagname2=value2...\r\n

It is assumed that the server can process in parallel a subscription mechanism
and atomic commands like GET or SET.
Changing one of the property while a connection is established
doesn't automatically close the connection. Thus it is the responsibility
of the calling class to close and reopen a connection if needed.

Loading the DataServers - class DataServerManager

public class **DataServerManager**

extends Object

This class is a DataServer class loader. Given the name of the driver, it will automatically create an instance of the DataServer and do the necessary configuration. So all the rest of the interface doesn't need to know about the existing Drivers.

Databases and Tag management

Interface tagHistory

A tagHistory object stores the history of a tag.
The value taken by the tag and the date for each of these values are stored together with a validity flag. This flag tells if the value,at that date, was considered valid or not.

class Trend

public class Trend implements tagHistory

It is a simple implementation of tagHistory

class Tag

public class Tag

This class holds the metadata related to a tag:
name, min-max, validity limits, type, date, value, tagHistory, increment, unit,...

It sends PropertyChangeEvent whenever a tag's property (not the value) changes

interface TagArrays

A **TagArray** handles a set of Tags. The objects (typically the TagsView or the DataServer) willing to modify the value of a tag are requested to do so using a TagArrays object. By doing so, the correct events will be send.
TagsView objects should use the method setValue. This method will send it, if necessary, to the related DataServer.
DataServer objects should use the method updateValue which doesn't send it back to the network.
It it possible to set a default DataServer which will be used by all the remote tags which don't have a link to a specific DataServer.

If the value of a tag has been changed, a TagArray sends the TagChangeEvent.

class TagContainer

class **TagContainer** implements TagArrays,Runnable

TagContainer is an implementation of TagArrays.
The Tags are stored in a Vector.
TagContainer knows how to handle TagNames with embedded properties such as *Name*.lowerLimit .
It fires **AlarmEvents** when a Tag is updated with a value which is outside the limits.

Events and Adapters

class TagClickedEvent

public class TagClickedEvent extends java.util.EventObject

This event is used to broadcast the fact that the operator has clicked on the representation of a tag in some TagsViews. Menu bars that show the name of the clicked tag and tree representation of TagsViews subscribe to this event.

class TagClickedAdapter

public class TagClickedAdapter implements TagClickedListener

This class is an event adapter which centrally manages the TagClicked events.
We prevent sending the same event with a frequency higher than 500 ms

interface TagClickedListener

public interface TagClickedListener extends java.util.EventListener

class TagChangeEvent

public class TagChangeEvent extends java.util.EventObject

This event is usually sent whenever a list of tags had their values changed in the rtdb

class TagChangeAdapter

public class TagChangeAdapter implements TagChangeListener

This class is an event adapter which centrally manages the TagChange events.
Each listener may ask for a specific tag list. In that case, they will receive an event
only if one of the tags belonging to their task list has changed.
If there is no task list (default) then the listener receive an event when any
of the tags change.

interface TagChangeListener

public interface TagChangeListener extends java.util.EventListener

class OpenViewEvent

public class OpenViewEvent extends java.util.EventObject

This event is usually sent by TagsViews that wish to open a new window containing a TagsView.

class OpenViewAdapter

public class OpenViewAdapter implements OpenViewListener,Runnable

This class is an event adapter which centrally manages the OpenView events.

interface OpenViewListener

public interface OpenViewListener extends java.util.EventListener

class AlarmEvent

public class AlarmEvent extends java.util.EventObject

The AlarmEvent is an event used to send informations about any anormal situation detected in the RemoteView system. It will typically be sent by TagArrays when the value of a Tag is outside limits or by DataServers when a connection with a remote server is lost.

class AlarmAdapter

public class AlarmAdapter implements AlarmListener

This class is an event adapter which centrally manages the Alarm events. We prevent sending the same event with a frequency higher than 1 s.

interface AlarmListener

public interface AlarmListener extends java.util.EventListener

class StatusEvent

public class StatusEvent extends java.util.EventObject

StatusEvent is an event used to pass global information on the system to the different elements.

interface StatusListener

public interface StatusListener extends java.util.EventListener

Viewing components – TagsViews

interface TagsView

This interface defines a component used to visualize in some means the behaviour of the tags. Typical implementations of this interface could be graphical trending system, alarm display windows, control system's supervisory synoptics, the representation of a led, the representation of a dial, ...
A TagsView must be an awt component of some sort (such as a canvas or a panel) to be able to be inserted in windows or other components. Some TagsViews may then be containers for other TagsViews.

VIEW TYPE:
Each TagsView implements one or several "view types", which is a name describing the represention. Using the methods getType(), getPossibleTypes() and doYouImplement(String type)the system can dynamically make the correspondance between a representation asked by a user and the component class which offers this kind of representation.
This "view type" can be different from the class' name enabling a single class to offers different view types.
If the system doesn't find a preloaded class implementing the required view type, it will try to dynamically find and load a class which name is equal to the view type.

CONFIGURATION:
To enable serialization and to easily configure the TagsView, each of them must be implements the methods toString() and setConfig(String config). By doing so, TagsViews don't have to bother about saving or loading their configuration. This will be handled by the system (see configurationMgr).

EVENTS:
It is assumed that the TagsView sends the TagClickedEvent when the operator clicks on the representation of a Tag, enabling so a uniform and easy behaviour of the interface.

CLASS Tagdisplay

public class Tagdisplay extends Canvas implements TagsView

TagsView type implemented: TEXT_TYPE
LED_TYPE
BAR_TYPE
TREND_TYPE

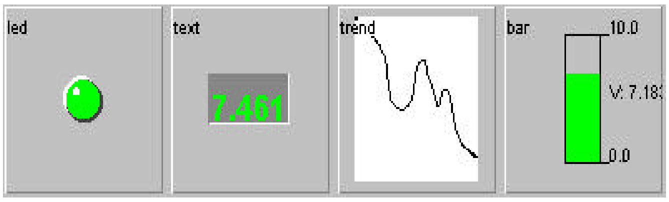
This class is an implementation of the TagsView interface. It allows to display the value of a tag or of a **dgsl** statement. If the value displayed is a tag, then the behavior of the display depends on the metadata related to the tag (if the value is outside the limit, it will be shown). Different types of display types are supported: TEXT_TYPE, LED_TYPE, BAR_TYPE, TREND_TYPE
Only the TEXT_TYPE is allowed when the result of a statement is displayed. In the case, this TagsView is polymorphic: it means that the user may change the type of the representation by double-clicking on the component.

Events:

As any TagsView, it fires TagcCliked events. This event is sent when the user clicks in the component.

Configuration:

The configuration is the name of the tag or a dgsl statement



CLASS JcomptagView

public class JComptagView extends Panel implements TagsView

JComptagView is a wrapper class around standard java and symantec components.
The implemented java components are:
java.awt.Label labelComponent;
symantec.itools.awt.HorizontalSlider hSlider;
symantec.itools.awt.VerticalSlider vSlider;
java.awt.Button button;
java.awt.ProgressBar progressBar;
java.awt.TextField textField;

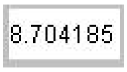
TagsView type implemented: TextLabel
HSlider
VSlider
Button
ProgressBar
TextField

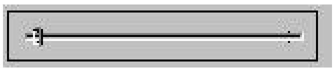
Events:

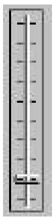
As any TagsView, it fires TagcCliked events. This event is sent when the user clicks in the component.

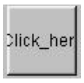
Configuration:

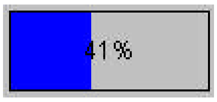
The configuration is the name of the tag or a dgsl statement

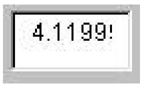












CLASS FrameDBView – The DataBase Viewer

public class FrameDBView extends Frame implements TagsView, TagClickedListener

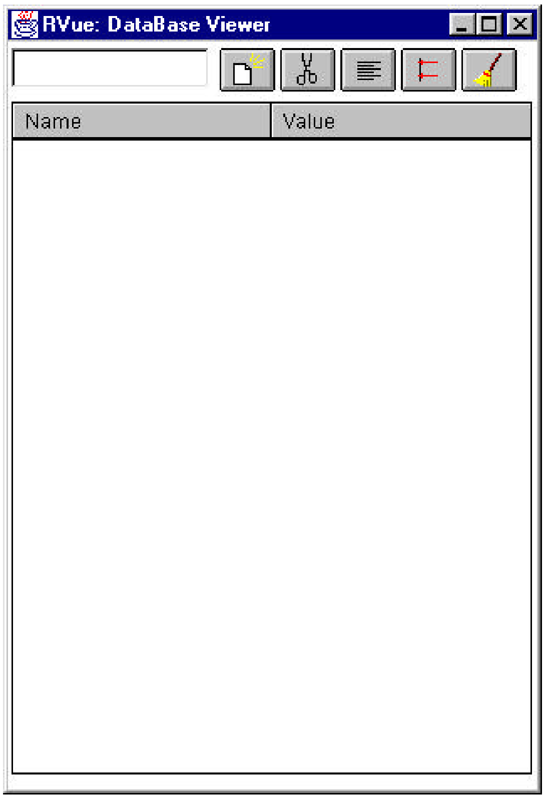
TagsView type implemented: DBView
TagsView type implemented: DBView

This class which is a "TagsView" enables the operator to list the content of the real-time database (rtdb). The Frame has 5 buttons which are:
1- new tag: The tag, which name is entered in the text field located in the menubar, is added to the rtdb. By default the tag is "remote".
2- cut tag: The tag, which name is entered in the text field located in the menubar,is deleted from the rtdb.
3- History list: The history of the choosen tag is listed in a new window.
4- Set limits: A new window appears and enables the operator to set the limits of the choosen tag.
5- Garbage collection: All the tags which are not subscribed by any TagsView are deleted.

Events:
As any TagsView, it fires TagcCliked events. This event is sent when the user clicks on the tag's line.
It receives also the TagClicked events.

Configuration:
The configuration is a file containing a list of tag names and their configuration.

```
#ViewType DBConfig
#TagDef NewTag2;3.2;0;12;1;11;true;true;TestProcess;true;1;Kelvin,
#TagDef NewTag3;4.2;0;15;0;14;true;true;Process2;true;1;degre,
```



CLASS ChartView

public class ChartView extends Panel implements TagsView,JCPickListener

TagsView type implemented: ChartView

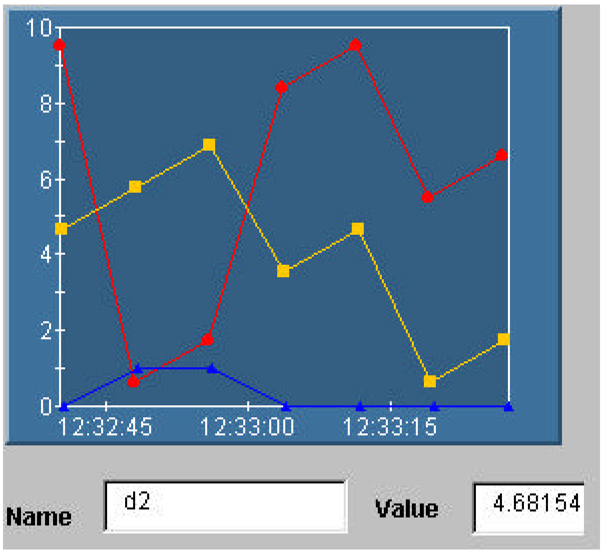
This class which is a "TagsView" draws the trend curves of the values of rtdb tags.
It is possible to add tags to trend by calling the addTag method or by setting
the configuration.
When a new tag is added, the chart fetches the tag's history in the local buffer.
When the user clicks on a trend curve, the closest value and the tag's name is
displayed.

Events:

As any TagsView, it fires TagcCliked events. This event is sent when the user clicks on a trend curve.

Configuration:

The configuration is a string containing a list of tag names separated by commas (ex: level1,PT215,TT141).



Name d2

Value 4.68154

CLASS Synoptic

public class Synoptic extends Panel implements TagsView

TagsView type implemented: SynopticView

The class Synoptic is a dgsl interpretor. It reads a dgsl (Dynamic Graphic Scripting Language) file and animates a synoptic according to the script commands (see DGSL specifications).

Events:

As any TagsView, it fires TagcCliked events. This event is sent when the user clicks on a dynamic object created by the interpretor.

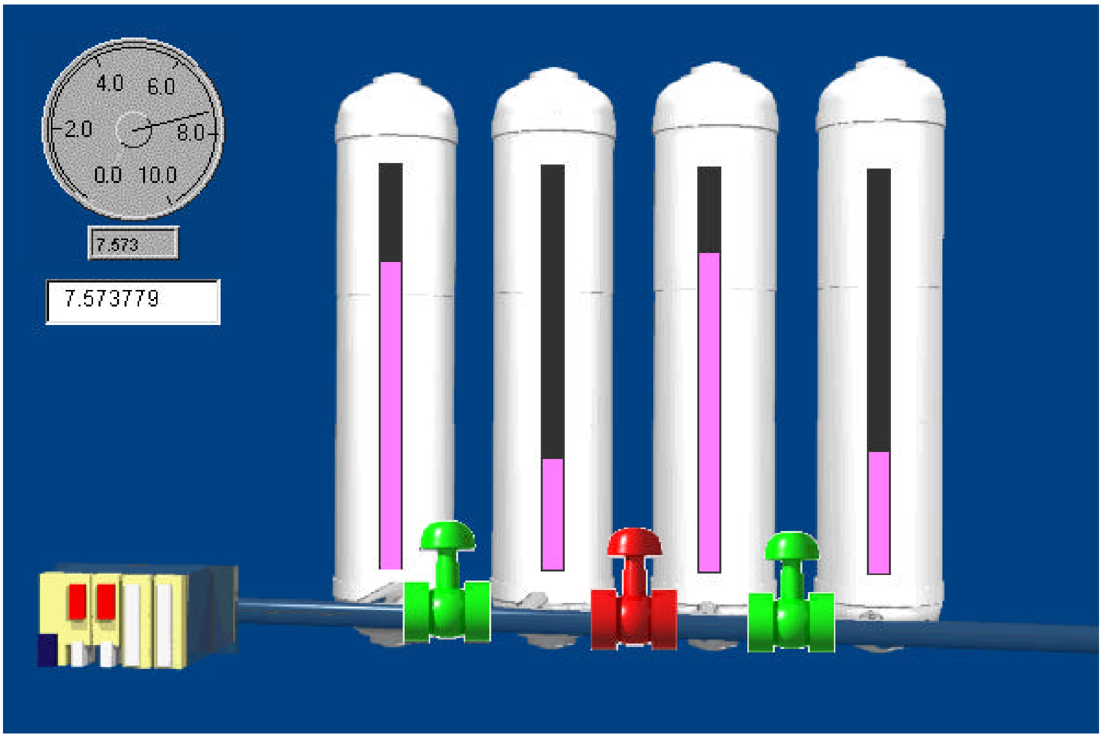
It also sends OpenViewEvent events when the user calls the "map" dgsl command.

Properties:

String1 String2 : String substitution in the script (typically tag names) if called before instantiate

Configuration:

The filename of the dgsl file.



Class CompoundView

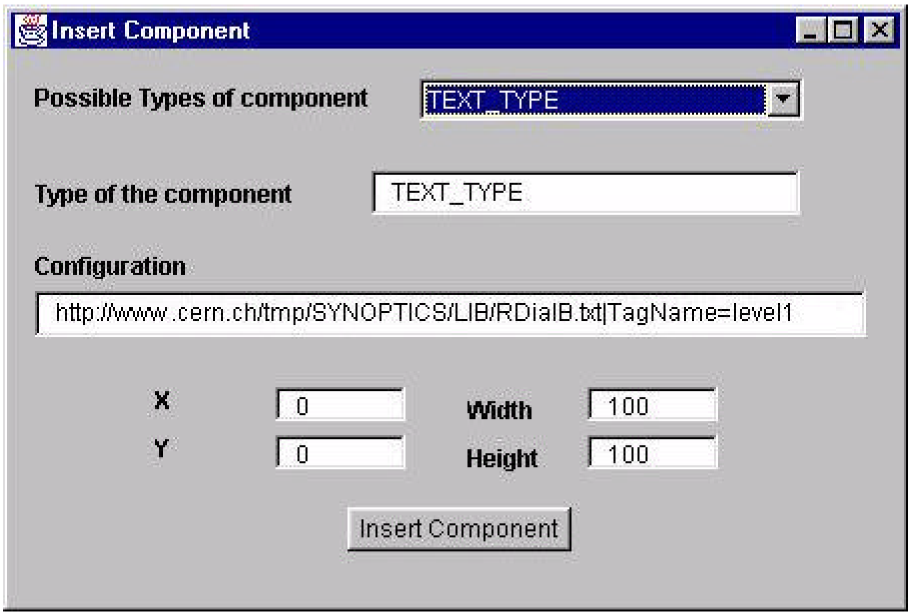
Public class CompoundView extends Panel implements TagsView
TagsView type implemented: CompoundView

Class CompoundView is a container for other TagsView.
It is possible to dynamically create a compoundView. When CompoundView receives a request to insert a Tag, a list of Tags or a TagsView, it opens a dialog window in which it is possible to specify how and where to insert the component.
It is also possible to move a TagsView component inside the CompoundView. To do so, double click in the CompoundView, but outside any included components. A dialog window will open. This new window subscribes to the TagClicked event. So, to move a component, one has to click on it in the CompoundView main window and then change its coordinates in the dialog window.

Public Methods:
public String **toString** (): this method will save remotely the description of the file and will return the URL of the remote file. It needs to get the name of the remote save program through the GlobalStatus.getRemoteSavePg method.
public void **setConfig**(String config) : This method will configure the CompoundView according to the content of the file which URL is given by the parameter config.

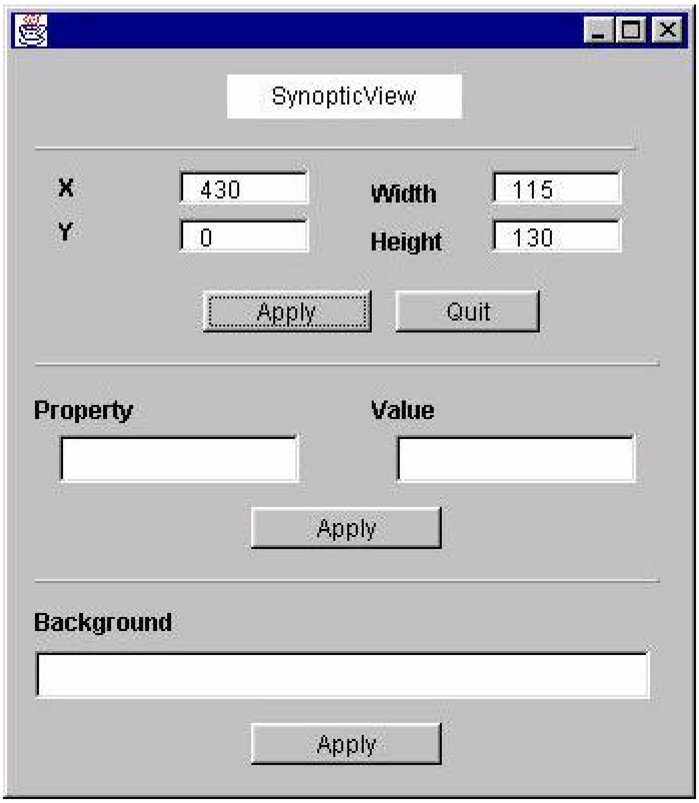
Configuration: the URL of the file containing the description in standard dgsl
Ex:

```
#ViewType CompoundView
#Size 600 105
#Background http://www.myhost.ch/BACK/C_panel.gif
# format: component type x y width height component_configuration
component TREND_TYPE 1 1 95 95 d3
component TEXT_TYPE 100 1 95 95 d2
component LED_TYPE 200 1 95 95 2
component TextLabel 300 1 55 25 d2
component HScrollbar 400 5 55 25 d2
```



Possible Types of component: TEXT_TYPE
Type of the component: TEXT_TYPE
Configuration: http://www.cern.ch/tmp/SYNOPTICS/LIB/RDialB.txt|TagName=level1
X: 0
Y: 0
Width: 100
Height: 100

The dialog window for inserting TagsViews



SynopticView

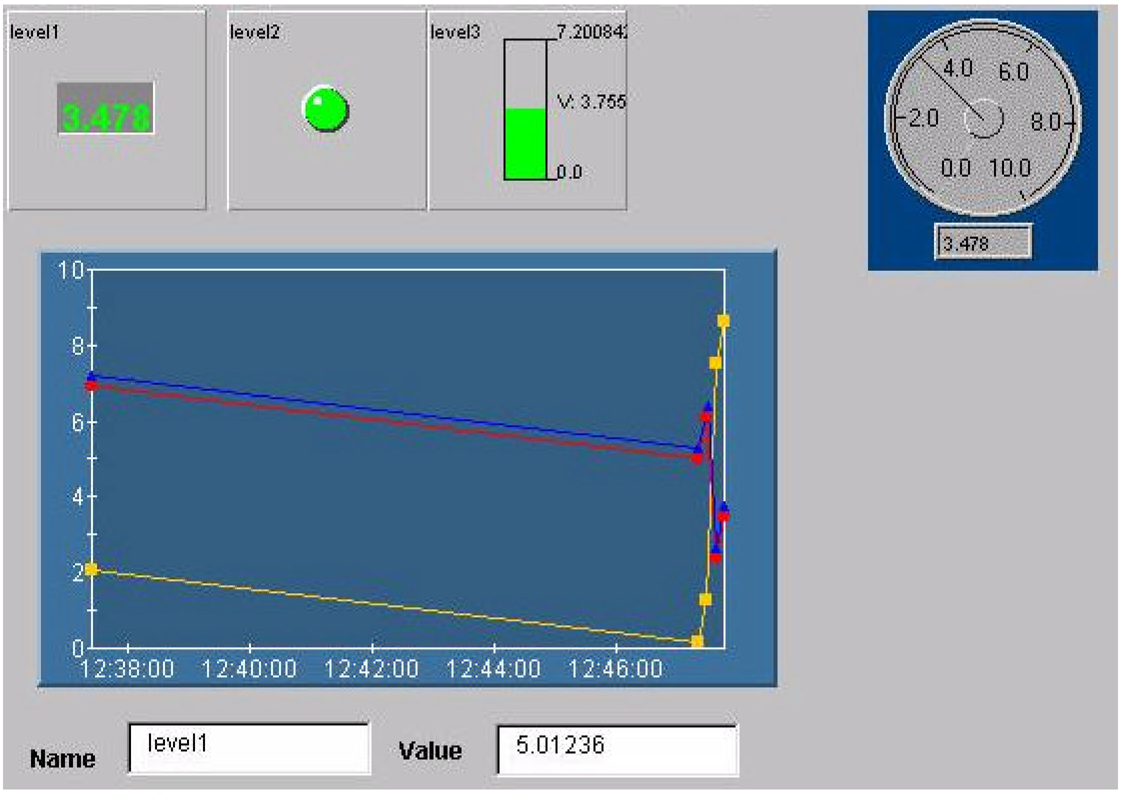
X 430

Y 0

Width 115

Height 130

The Dialog Window to move and resize the included TagsViews



Name level1

Value 5.01236

An exemple of a CompoundView

Class BarGraphView

Public class BarGraphView extends extends CanvasMTagsView

TagsView type implemented: BarGraphView

This TagsView gives a Bargraph representation.

Base Class CanvasMTagsView

public class CanvasMTagsView extends Canvas implements TagsView
TagsView type implemented: CanvasMTagsView

This class which is a "TagsView" is a simple base class for a TagsView which display multiple tags. This class is intended to be extended (see example class BarGraphView)

Events:
As any TagsView, it may fires TagcCliked and Alarm events.

Configuration:
The configuration is a string containing a list of tag names separated by commas (ex: level1,PT215,TT141).

Base Class CanvasSTagsView

public class CanvasSTagsView extends Canvas implements TagsView

TagsView type implemented: CanvasSTagsView

This class which is a "TagsView" is a simple base class for a TagsView which display one tag. It is intended to be extended.

If the component doesn't want the tag's name to be changed, it must set the public boolean Changeable to false;

Events:
As any TagsView, it may fires TagcCliked and Alarm events.

Configuration:
The configuration is the name of the tag

Base Class PanelMTagsView

public class PanelMTagsView extends Panel implements TagsView

> TagsView type implemented: PanelMTagsView
>
> This class which is a "TagsView" is a simple base class for a TagsView which display multiple tags. It is intended to be extended.
>
> Events:
> As any TagsView, it may fires TagcCliked and Alarm events.
>
> Configuration:
> The configuration is a string containing a list of tag names separated by commas (ex: level1,PT215,TT141).

Base Class PanelSTagsView

public class PanelSTagsView extends Panel implements TagsView

TagsView type implemented: PanelSTagsView

This class, which is a "TagsView", is a simple base class for a TagsView which display
one tag. It is intended to be extended.

If the component doesn't want the tag's name to be changed, it must set
the public boolean Changeable to false;

Events:
As any TagsView, it may fires TagcCliked and Alarm events.

Configuration:
The configuration is the name of the tag

Loading the TagsViews - Class componentsType

public class **componentsType**

This class is a TagsView component loader. Given the name of the component, it will automatically create an instance of the TagsView. So all the rest of the interface doesn't need to know about the existing TagsViews.
This class finds the correct TagsView for a given type, create an instance of this TagsView and returns it. If an alias list has been loaded it is used to access the correct component. In case the type of the TagsView is unknown, we try to locate a class which name is the type of the TagsView.

Managing the configurations

Class configurationMgr

public class **configurationMgr**

This class handles all the configurations of the TagsView and the windows which contains them. A "window configuration" contains the description of the window itself, the type of the TagsView to create, a "TagsView configuration" and informations on how to instanciate the TagsView. It can also be a set of "window configurations", thus enabling to load multiple windows with a single action. A list of window configurations is maintened on the server. To each window configuration is associated a name. This list may be retrieved with the method loadWindowConfigurationList . It is possible to add a new window configuration to this list using the publishWindowConfiguration method. A TagsView configuration is, depending on the TagsView, either a String or a pointer (such as an URL) to a remote configuration file. This TagsView configuration will be passed to the TagsView upon creation. If it is a pointer to a remote configuration file, it is up to the TagsView to load the configuration file. This can be done using the loadConfigurationFile method. The TagsViews can use saveConfigurationFile to save their configurations. If the TagsView uses the loadInitConfigurationFile method to load a configuration file, then the initializations of the tags in the real-time database and the initializations of the dataservers will be done automatically before returning. This implies that the configuration file uses the standard way of defining tags and dataservers. This implementation uses a Web server and a set of remote programs to offer the service.

Managing windows

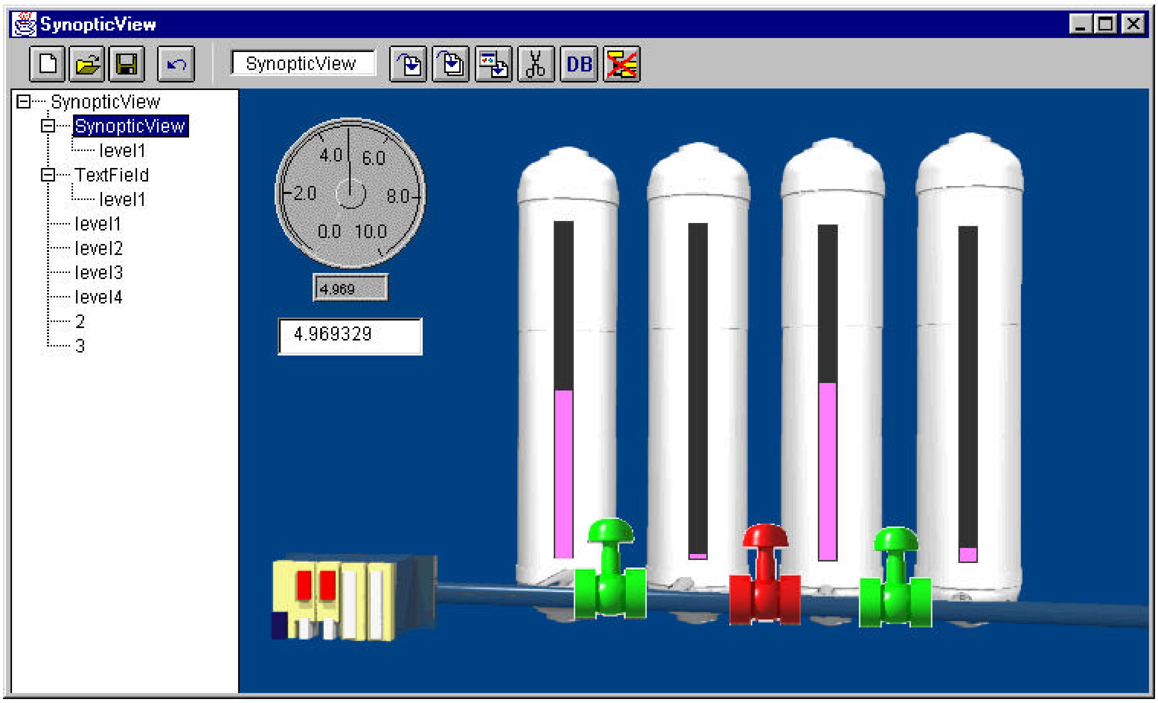
Interface ViewManager

Although a TagsView may be inserted in any java.awt.Container, or in another TagsView (see CompoundView), some container specifically dedicated to TagsView have been developped (it could be a Frame of some sort). They share a common behavior described in this interface.
Typically the top level TagsView component will be contained in a ViewManager.

Class FrameViewMgr

public class **FrameViewMgr** extends Frame implements ViewManager

This is an implementation of ViewManager as a Frame. The TagsView contained in this ViewManager is opened in a new window. The window has a bar of buttons on the top to interact with the TagsView. On demand, a tree-like representation of the TagsView is shown.



A TagsView of type « Synoptic » inside a FrameViewMgr. The tree-like representation is shown on the left of the window.

Class PanelViewMgr

public class **PanelViewMgr** extends Panel implements ViewManager

This ViewManager, which is a Panel needs a window. This class is usefull to insert a TagsView in an Applet.

Putting all together

Class CentralManager

public class **CentralManager** implements OpenViewListener

CentralManager is in charge of instanciating the main objects, such as the frames containing the TagsView or the DataServers. It knows where to store the configurations which are to be saved. CentralManager holds global informations on the running system. It knows, for example, whether remote data are being updated or not. Notification for the modifications on these global informations are sent to listeners through StatusEvent

Services and utilities

CLASS dgslStatement

This class is able to interpret a set of dgsl statements which are connected to a parameter of a dgsl command. Such statements are in the form of "data streams"

- **input data streams** are typically used to change one of the parameters of the dgsl command according to values contained in a "TagArrays"
- **ouput data streams** are typically used to affect TagArrays according to some user action on the dgsl object created by the dgls command.

```
Data stream syntax:
 data_stream
            data_stream, data_stream, …
 data_stream
            [input_data_stream]
        [output_data_stream]
 input_data_stream
            $name
            $(condition;statement;[validity])
 output_data_stream
            @name
            @([condition];name=statement;[validity])
            @([condition];name;[validity])
 condition
            statement returning a boolean
            default
 statement
            32+$T3…
 validity
            l1, l2, …       list of possible values
            low-high

Examples:
 a/ $d1
 b/ $(default;$d1+3;)@d2
 c/ $($d1<5;$d2+20;)$(default;100;)
 b/ @($d5==1;d4=$input+$d6;)@d7$(default;100;)@($d1<5;d7;1,2,3)@(;d8=$input1+$input;)
```

CLASS Evaluate

This class is able to evaluate expressions.

Appendix 1 – Dynamic Graphic Scripting Language 1.0

DGSL specifications

DGSL is a simple scripting language which enables anyone to quickly describe the behavior of a graphical view. Any dgsl interpretor must be connected to a data source (real-time database) to make the graphical view dynamic.

Graphical Commands

arc cx cy width height start_angle end_angle line_property [fill_property]
draws an arc of ellipse centered in cx,cy

line x1 y1 x2 y2 line_property
draws a line.

whline x1 y1 width height line_property
draws a line.

dashedline x1 y1 x2 y2 line_property
draws a dashed line.

whdashedline x1 y1 width height line_property
draws a dashed line.

filledrectangle x1 y1 x2 y2 fill_property
draws a filled box.

whfilledrectangle x1 y1 width height fill_property
draws a filled box.

rectangle x1 y1 x2 y2 line_property [fill_property]
draws a rectangle.

whrectangle x1 y1 width height line_property [fill_property]
draws a rectangle.

polygon number_of_points x1 y1 x2 y2 ... line_property [fill_property]
draws a polygon.

filledpolygon number_of_points x1 y1 x2 y2 ... fill_property
draws a polygon and fills it.

image x y *width height filename*
insert an other image on top of the current image.

string font_property x y text text_property
writes text.

symbol number_of_symbol x y *value* file1 file2 ...
depending on the *value*, insert one of the images on top of the current one.

view offset_x offset_y scale_x scale_y
changes the origin of the coordinate system (useful to draw bar-graphs or to position an object depending on a value).

cartesian
back to normal coordinate system after a set of commands made in a polar system (see the polar command).

polar offset_angle scale_angle scale_length
change the coordinate system to a polar one (useful to draw dials or to rotate objects). In polar coordinates, x becomes the radius and y becomes the angle.

sound number_of_symbol x y value soundfile1 soundfile2 ...
depending on the *value*, plays one of the sounds which URL is soundfilen

Other statements

map x1 y1 x2 y2 configuration_file_URL

defines a sensitive area in the image. When the user clicks inside (x1,y1)-(x2,y2) a new configuration is loaded and interpreted.

Component *component_type* x1 y1 width height tagname *[Param1 v1] [Param2 v2]…*

Insert component *component_type* at position x1, y1 and resize it to width, height. Parameters Param1… and their initialization values v1… are passed to the component.

set statement

#remarks

#ImageBase URL

defines the base URL for the images

#Background image_URL

draws the image on the background.

#Size width height

defines the size of the graphic.

#include file [oldstring1 newstring1 [oldstring2 newstring2 […]]]

includes another file and replaces oldstringn by newstringn during the process

All the commands starting with a # are only interpreted once when the script is loaded. It is thus impossible to connect any part of them to a variable.

Control Flow

```
if  (condition)
    Statement 1
else
    Statement 2
end
```

Ex :

```
if ($d1<5)
    whfilledrectangle 230 200 50 25 red
else
    whfilledrectangle 205 200 50 25 green
end
```

Properties

line property

color*[/width=w/…/]*

fill_property

color

text_property

color

font_property

small

large
face=font_name[/size=font_size/]

color

white
blue
red
green
black
grey
yellow
orange
magenta
#XXXXXX

Making the graphic dynamic

To make the graphic dynamic, it suffice to replace one or several of the commands previously described by a variable's name or by an expression containing a variable. Thus by replacing

rectangle 1 1 100 100 blue

by

rectangle 1 $MyVariable 100 100 blue

we link the Y position of the line's starting point to the variable *MyVariable*

In all the previously described commands, blue parts may be replaced by an expression also called here a "**data stream**" because, on a running system, the command is redrawn each time the variable change. A data_stream may be combined of a list of data_stream, each of them being either an "**input data stream**" describing a data coming from the database or an "**output data stream**" describing a data which may be changed by the user and then sent to the database.

data_stream

data_stream, data_stream, …

data_stream

[input_data_stream]
[output_data_stream]

input_data_stream

$name
$(condition;statement;validity)

output_data_stream

@name
@(condition;name=statement;validity)

condition

statement returning a boolean
default

statement

32+$T3…
$input
$input***n***
$***n***

validity

l1, l2, …	list of possible values
low-high	

DGSL statement examples

1/ Example 1:

symbol 2 137 57 $led@($led==1;led=0;)@($led==0;led=1;) ledred.gif ledgreen.gif

Input part: **$led**

Here we display an image on a fixed position (137,57). The content of the image depends on the value of the variable ***led***. If the variable ***led*** contains 0, then the image ledred.gif is displayed. If it contains 1, then the image ledgreen.gif is displayed.

Output part: **@($led==1;led=0;)@($led==0;led=1;)**

This output data stream enables the user to switch between the two possible images. When the user double click on the symbol, the output data stream will be made active. The conditions will be evaluated, and if they are true, the statement part of the stream will be executed. Here is the logic:

```
For each output data stream
  Test the condition
  If true than execute the statement
```

Because all the output data streams of a DGSL command are done in parallel, the led variable won’t be switched to 0 and then immediately to 1.

2/Exemple 2: (one line)

string small 1@(;;1-$d3) $Yposition@Yposition $($d1<5;$d2+20;)@($d5==1;d4=$input+$d6;)@d7$(default;100;)@($d1<5;d9;1,2)@(;d8=$input1 +$1;) black

Input part: **$($d1<5;$d2+20;)$(default;100;)** on the text argument part of the command
$Yposition on the Y property part of the command.

If the variable d1 holds a value lower than 5, the sum of d2 and 20 will be displayed. If the variable d1 holds a value greater than 5, “100” will be displayed by this command.
The text will be displayed on the Y-axis depending on the value of the variable Yposition.

Output part: **@(;;1-$d3)** on the X property part of the command
@Yposition on the Y property part of the command
@($d5==1;d4=$input+$d6;)@d7@($d1<5;d9;1,2)@(;d8=$input1+$2;) on the text property part of the command.

The user will be able to modify interactively the X position of this text, in a possible range of values going from 1 to whatever d3 contains. The chosen position will be used in the statement d8=$input1+$2 ($2 represents the value modified by the user for the 2nd property of the command). Notice that the position will not remain changed and will immediately go back to 1.
The user will be able to modify interactively the Y position of this text. There are no limits. The Y position will remain changed because it has been stored in the variable Yposition and this same variable is used to position the text on the Y axis.
If the user double clicks on the text, the system will ask him to enter a value for the text (only values 1 or 2 will be accepted) and for $input1and then:

- If d5 equals 1 it will put the sum of d6 and what has been entered for the text in d4.
- It will put what has been entered for the text in d7.
- If d1 is smaller than 5 it will put what has been entered for the text in d9.
- It will put the sum of the chosen X position and what has been entered for input1 in d8.

Some real examples of dgsl scripts

Example 1 :

```
#ViewType SynopticView
#ImageBase http://myhost/SYNOPTICS/SYMBOLS/
#Size 600 400
#Background http://myhost/SYNOPTICS/BACK/synoptic.gif
# TagDef lines are sent to TagArrays (rtdb) for definition of tags
```

```
#TagDef NewTag;3.2;0;12;1;11;true;true;TestHTTP;true;1;Kelvin,
#TagDef level1;0;0;10;0;9.0;true;true;TestHTTP;true;.01;percent,
# ---------- components ----------------------------
component SynopticView 15 15 115 130
http://myhost/SYNOPTICS/LIB/RDialB.txt|TagName=level1
component TextField 25 150 95 25 level1
#
# Les tanks -------------------------------------
filledrectangle 207 308 218 $(;308-(22*$level1);) $($level1<8;#FF7DFF;)$(default;red;)
filledrectangle 295 308 306 $(;308-(22*$level2);) $($level2<8;#FF7DFF;)$(default;red;)
#
# -------- Les vannes
#
symbol 2 219 281 $2 van4.gif van3.gif
symbol 2 321 285 $3 van4.gif van3.gif
#
# hyperlink to another description file
map 18 301 111 369 http://wwwlhc.cern.ch/tmp/SYNOPTICS/DESC/C_panel.txt
```

Example 2 :

```
#ViewType SynopticView
#Background http://myhost/SYNOPTICS/BACK/back1.gif
#ImageBase http://myhost/tmp/SYNOPTICS/SYMBOLS/
#Size 600 530
#---- Texte ----------------------
string Serif|18 1 1 $level1 black
#----- map --------------------------------
map 0 0 30 30 http://myhost/SYNOPTICS/DESC/syn2.txt
#---- moveable rectangle --------------------
whfilledrectangle $(;$X+100;)@(;X=$input-100;100<>200) 0 10 10 yellow
# the tank
view 335 145 1 -9.8
filledrectangle 0 0 57 $d1 orange
view 0 0 1 1
string small 408 50 $d1 black
#
# Now the 2 leds
#--------------------------------------------
#
symbol 2 137 57 $bool2 ledred.gif ledgreen.gif
symbol 2 266 57 $bool3 ledred.gif ledgreen.gif
#
# And now the dial
#----------------------------------------------------------
polar 120 30 1
# centre de l'aiguille
view 76 101 1 1
# 3 eme argument = longueur, le 4 = angle
line 0 0 43 $double2 black
cartesian
view 0 0 1 1
string small 55 158 $double2 black
```

Appendix2 – Source of a TagsView - BarGraphView

```
package cern.lhcias.csgui.TagsView;

import cern.lhcias.csgui.interfaces.*;
import cern.lhcias.csgui.Events.*;
import cern.lhcias.csgui.rtdb.*;
import cern.lhcias.csgui.Utils.*;
import java.io.*;
import java.awt.*;
import java.applet.Applet;
import java.util.*;

/*---------------------------------------------*/
public class BarGraphView extends CanvasMTagsView {
   int nb_of_ticks = 10;
   int x_origin=40, y_origin=5;       //5
   int x_border = 60, y_border = 25;   //7
   boolean gridOn = false;
   Font legendFont;
   Color backgroundColor;

    public BarGraphView() {
        super();
        TYPE = "BarGraphView";
        legendFont = new Font("Helvetica",Font.PLAIN,10);
        backgroundColor = new Color(51,51,102);
    }

    void TagListHasChanged() {
        repaint();
    }

   int getIndex(int x,int y) {
       int nb = TagstoView.size();
       if (nb < 1) return -1;
      int bar_width= (getBounds().width - x_border)/nb;
      int num = (x - x_origin)/bar_width;
      if (num >= TagstoView.size()) return -1;
      return(num);
   }

   public boolean mouseDown(Event evt,  int x, int y) {
        if (x < x_origin) {
            if (gridOn) gridOn = false;
            else gridOn = true;
            repaint();
            return false;
        }
      int clicked_tag = getIndex(x,y);
      if (clicked_tag > -1) fireTagClickedEvent((String)
TagstoView.elementAt(clicked_tag));
      return false;
   }

   public void paint(Graphics g) {
        Rectangle rec = getBounds();
      int width= rec.width - x_border, height=rec.height - y_border;
        g.setColor(backgroundColor);
        g.fillRect(x_origin,y_origin,width,height);
      g.setColor(Color.white);
      g.drawLine(x_origin,y_origin,x_origin,y_origin+height);
      g.drawLine(x_origin,y_origin+height,x_origin+width,y_origin+height);

      double d_m[] = getLimits();
      drawVerticalTicks(g,x_origin,y_origin,width, height,nb_of_ticks,d_m[0],d_m[1],3,
                      gridOn, Color.black, Color.white);

      g.setFont(legendFont);
      FontMetrics fm = g.getFontMetrics();
```

```
     int nb = TagstoView.size();
   int bar_width = (width/nb) -5;
     Vector Tags = getTags();
   for (int i=0; i<nb; i++) {
       int xpos = x_origin + 4 + i*(width/nb);
       g.setColor(Color.black);
       g.drawString((String) TagstoView.elementAt(i),
               xpos, y_origin + height + fm.getMaxAscent());
      paint_single_bar((Tag)Tags.elementAt(i), g, xpos,
              y_origin, bar_width, height, d_m[0], d_m[1],
              MyUtils.get_indexed_color(i));
   }
}

/*-------------------------------------------------------------------*/
private void paint_single_bar(Tag tag, Graphics g,int x_origin, int y_origin,
   int bar_width, int bar_height, double min, double max, Color couleur)
{
   double b_height = (new Double(bar_height)).doubleValue();
   double height, height_to0;
   int iheight ,iheight_to0;
   g.setColor(couleur);

   boolean load_error = tag.getAcquisitionError();
   double double_value = tag.doubleValue();

   if (tag.isValid()) {
      height = (double_value - min) * b_height/(max-min);
      if (height > b_height) height = b_height;
      if (height < 0) height = 0;
      iheight = (new Double(height)).intValue();
      if (max*min < 0) {
         height_to0 = (double_value) * b_height/(max-min);
         iheight_to0 = (new Double(height_to0)).intValue();
         if ((iheight_to0 >= 0)) {
         if (load_error)
            g.drawRect(x_origin, y_origin+bar_height-iheight,
                     bar_width,iheight_to0);
         else
            g.fillRect(x_origin, y_origin+bar_height-iheight,
                     bar_width,iheight_to0);
         }
         else {
         int iheight0 = (new Double((0 - min) * b_height/(max-min))).intValue();
         if (load_error)
            g.drawRect(x_origin, y_origin +bar_height- iheight0,
                     bar_width,-iheight_to0);
         else
            g.fillRect(x_origin, y_origin +bar_height- iheight0,
                     bar_width,-iheight_to0);
         }
      }
      else {
         if (max <=0) {
            height_to0 = (double_value) * b_height/(max-min);
            iheight_to0 = (new Double(height_to0)).intValue();
            if (load_error)
               g.drawRect(x_origin, y_origin,
                        bar_width,-iheight_to0);
            else
               g.fillRect(x_origin, y_origin,
                        bar_width,-iheight_to0);
         }
         else {
            if (load_error)
               g.drawRect(x_origin, y_origin+bar_height-iheight,
                        bar_width,iheight);
            else
               g.fillRect(x_origin, y_origin+bar_height-iheight,
                        bar_width,iheight);
```

```
                }
            }
        }

    }
}
```