Tango ATK Tutorial

This document is a practical guide for Tango ATK programmers and includes several trails with examples and demonstrations. Most of the examples and demonstrations are provided as Macromedia Flash documents.

In this document we assume that the reader has a good knowledge of the Java programming language, and a thorough understanding of object-oriented programming. Also, it is expected that the reader is fluent in all aspects regarding Tango devices, attributes, and commands.

Before going through the trails and examples, the Tango ATK architecture and key concepts are introduced. After this introduction the rest of the document is organized in a set of trails.

Introduction

Tango Application Toolkit also called “ATK” is a client framework for building applications based on Java Swing in a Tango control system.

Goals of Tango ATK

The main goals of ATK are the following:

- Speeding up the development of Tango graphical clients.
- Standardizing the look and feel of Tango applications.
- Implementing the core of “any” Tango application.

To achieve the first and the second goals ATK provides several swing based components to view and/or to interact with Tango device attributes and Tango device commands and also a complete synoptic viewing system. To achieve the third goal ATK takes in charge the automatic update of device data either through Tango events or by polling the device attributes. ATK takes also in charge the error handling and display. The ATK swing components are the Java Beans, so they can easily be added to a Java IDE (like NetBeans) to speed up the development of graphical control applications.

The Software Architecture of Tango ATK

Tango ATK is developed using the Model-View-Controller design pattern also used in the Java Swing package. The Tango basic objects such as device attributes and device commands provide the model for the ATK Swing based components called viewers. The models and the viewers are regrouped in two separate packages respectively ATKCore and ATKWidget.
Important notes:
ATK is based on Swing. Mixing the use of other “non swing” objects such as SWT (eclipse) with ATK is not recommended.
ATK hides Tango Java API (TangORB). It is highly recommended not to use TangORB methods and objects directly in the application code. Always use the interface provided by ATK to access the control system.

The key concepts of Tango ATK
Reminder: the central Tango component is the DEVICE. The Tango control system can be seen as a collection of devices distributed over the network. The tango devices provide attributes (for reading and setting data) and commands.

The central ATK components, to access the Tango control system, are: attributes and commands and not the devices. Through Tango Java API (TangORB) the control system is a
collection of devices where through ATK the control system is a collection of attributes and commands.

In addition to ATK attributes and ATK commands ATK provides two components, which are ATK attribute lists and ATK command lists.

So the central ATK components are:

- ATK Attribute (interface to Tango device attribute),
- ATK Command (interface to Tango Device command),
- ATK AttributeList (collection of ATK Attributes)
- ATK CommandList (collection of ATK Commands).

**Tango ATK viewers**

ATK viewers are provided as Java Beans and as such they can easily be added in a Java IDE (like NetBeans) to speed up the development of the graphical applications. This way the programmer can easily build up his (her) panels, mixing pure Swing objects and Tango ATK viewers.

ATK viewers are provided for different types of Tango Components. They can be divided into different categories such as: error history window, error popup window, simple attribute viewers / editors, attribute list viewers, simple command viewers, command list viewers, …etc.
Synoptic drawing and viewing

A synoptic is a drawing in which each object can be linked to a Tango object. A part of the synoptic drawing can be linked to the state attribute of a Tango device where another part is associated to a numerical attribute of another Tango device. The main idea of synoptic drawing and viewing system is to provide the application designer with a simple and a flexible way to draw a synoptic and to animate it at runtime according to the values and states read from the control system. ATK provides two components for this purpose:

- A graphical editor called “Jdraw”. This tool is used during the design phase to draw and to specify the synoptic. The synoptic is saved to a file.
- A synoptic viewer called “SynopticFileViewer”. This viewer is used in the graphical user interface of the application. SynopticFileViewer loads and browses the synoptic drawing file and animates its elements at runtime according to their state or to their value.
Getting Started

The following short tutorial takes you through some of the basic steps necessary to develop a Tango Java application based on Tango ATK.

In this tutorial we don’t use any Java IDE features. All the java code is entered manually using a java source editor. The NetBeans java source editor is used as any source editor.

Let’s specify the application we want to build in terms of the Model-View-Controller design pattern described before.

Our “Getting Started” application will need to show two tango device attributes and one tango device command all related to the same device. The tango device name used in this tutorial is “jlp/test/1”. The application will show the “state” and the “att_spectrum” attributes of this device and will give access to its “Init” command.

1. The type of the “state” attribute (jlp/test/1/state) is “DevState” and its format is “Scalar”
2. The type of the “att_spectrum” attribute (jlp/test/1/att_spectrum) is “DevDouble” and its format is “Spectrum”
3. The “Init” command (jlp/test/1/Init) has no input and no output argument (input and output argument types are DevVoid)

The ATKCore components are used to create and initialise the “model” part of the design pattern:
- One attribute list
- Two attributes (jlp/test/1/state, jlp/test/1/att_spectrum)
- One command list
- One command (jlp/test/1/Init)

The ATKWidget components are used to create and initialise the “view” part of the design pattern. These components are the objects adapted to the type of the tango component we want to visualize. They are also called “viewers” (attribute viewers, command viewers, … etc.).
- One State viewer (a viewer adapted to the DevState Scalar attributes)
- One NumberSpectrum viewer (a viewer adapted to any numerical spectrum attribute)
- One VoidVoidCommand viewer (a viewer adapted to the any command with no input and no output argument).
The controller part consists of making the relationship between the “view” components and the “model” components. Calling the “setModel” method of the view object makes this relationship. For example the call “stateViewer.setModel(stateAtt)” will make the relationship between these two objects.

Click on the following link to view a Flash demo of how to build the “GettingStarted” application.

[Getting Started (Flash Demo)]

**The Structure of an ATK application**

Any ATK application should perform a minimum set of operations. The following lists this minimum set of operations:

1. Declaration and initialization of ATKCore objects (AttributeLists, CommandLists, individual ATKCore attributes and individual ATKCore commands).
2. Declaration and instantiation of ATKWidget Error viewers to handle errors
3. Connection to attributes and commands by adding them to the appropriate list
4. Creation of the specific Attribute and command viewers, and add them to a swing window
5. Associate each viewer to an appropriate ATKCore attribute or command
6. Start the refresher thread associated to the attribute list

The following slide show will present in detail the skeleton of an ATK application:

[ATK application skeleton (Flash Slide Show)]

**Using ATK inside a Java IDE (NetBeans)**

Several Java IDEs (Integrated Design Environments) are available on the market and also as freeware. You can search the Internet to choose the most appropriate one for your usage. Here you can find some links to start with:

[NetBeans (free download)]
[Eclipse (free download)]
[Intelligent Idea (commercial tool)]
The use of the Java IDEs especially those including a good graphical user interface builder speed up the development of Tango ATK applications. From now on all the examples in this tutorial are made using the NetBeans 5.5 or 6.1, Java IDE. The present section presents the manner in which the ATK Java Beans can be integrated to the NetBeans Palette and used to build the user interface of the final ATK application.

If you are using another Java IDE please refer to its documentation to find out how to integrate and use the ATK Java Beans inside the IDE, to build a graphical user interface.

**Learning NetBeans**

You should first download and install the NetBeans IDE from the NetBeans Web site. If you have never installed JDK on your computer or if the JDK on your computer is out of date, you may consider to install the bundle NetBeans+JDK depending on the version of NetBeans and JDK you wish to install. For example inside the download page of NetBeans you can find:

- **NetBeans IDE 6.1 with JDK 5.0 Bundle**

This download will install JDK 1.5 and NetBeans 6.1 in a single operation. Once the NetBeans is installed you can browse: NetBeans Tutorials, Guides, and Articles which help you, learn more about NetBeans.

If you are a beginner with NetBeans we recommend you to go through the two following quick start guides:

- Guided Video Tour of NetBeans IDE 6.0 and 6.1
- NetBeans 6 IDE Quick Start Tutorial
- Designing a Swing GUI in NetBeans IDE

**Create an ATK Application project in NetBeans**

To create and ATK application project, you may go through the following steps:

1. Create the NetBeans Java Application Project
2. Add the Tango and ATK jar files in the project’s class path
3. Add several ATK Java Beans (ATK viewers) to the NetBeans palette

**ATK application project using NetBeans (Flash Demo)**

**First ATK Simple GUI application**

Now we build a simple Tango GUI using ATK viewers, which have been added to the NetBeans palette. Here are the steps to follow to build this GUI:

1. Create a source package
2. Create a new Jframe form in this package
3. Add ATK beans from the palette into the form and place and resize them as you wish
4. Add the necessary source code to create and to initialize the ATKCore (model) objects
5. Associate each viewer with its model
6. Start the ATK refresher(s)
7. Build and run this GUI

**First ATK Simple GUI application (Flash Demo)**
ATK Quick Tour

This section includes the first list of tutorials, which give you a quick tour of the Tango ATK components by guiding you through the creation of a simple generic application very similar to AtkPanel. During this quick tour you will learn how to view device state and status attributes, and how to display a collection of tango scalar attributes all aligned with each other. You will also use a viewer to display a collection of tango device commands.

Device state and device status

The state and the status of the device are two attributes of any Tango device (IDL 3 and above). Atk provides two attribute viewers one called StateViewer and the other StatusViewer to display them. These viewers are included in the fr.esrf.tangoatk.widget.attribute package. The model for the StateViewer is the state attribute (DevStateScalar) and the model for the StatusViewer is any scalar attribute of type String (StringScalarAttribute).

You can go through the following simple demo to see how to use these two viewers.

State and Status viewers (flash demo)

Display a list of scalar attributes

The ATK attribute list viewers / setters are provided to be able to display a collection of attributes all aligned together. In fact, the ATK attribute list viewers handle only scalar attributes. An attribute list viewer’s model is an attribute list. This means the model for this type of viewers cannot be an individual attribute and should be an attribute list. The attribute list viewers are all included in the fr.esrf.tangoatk.widget.attribute package.

The ATK list viewers provide the application with three major advantages:

- The first advantage is that all the single attribute viewers are aligned in a coherent manner inside the attribute list viewer.
- The second advantage is that the application can be “generic”. An application program with no knowledge of the exact names and types of the scalar attributes of a particular device, can display all of them easily with two lines of code.
- The third advantage is that the application programmer does not need to know which type of attribute viewer is adapted to which type of tango attribute. The ATK list viewers automatically select the adapted viewer and / or setter for each type of device attribute.

There are three classes for attribute list viewing:

- ScalarListViewer
- NumberScalarListViewer
- ScalarListSetter.
The ScalarListViewer and NumberScalarListViewer are almost the same. The only difference is that the NumberScalarListViewer will display only the scalar attributes which are numerical where ScalarListViewer will display also StringScalar attributes, BooleanScalar and EnumScalar attributes in addition to the numerical scalar attributes.

The attributes, members of the attribute list are displayed vertically. In each line an individual attribute is displayed in the following manner:

1. At the left the “label” property of the tango attribute
2. Next to the label the “read” value of the attribute is displayed according to the “format” and the “unit” properties of the tango attribute
3. In the third column the “setpoint” of the tango attribute is displayed inside a viewer (mostly called editor), which allows setting the attribute value.
4. The last (forth) column is used to display a pushbutton with three dots. A click on this pushbutton pops up a window called “SimplePropertyFrame”. In this window the user can modify any property of the tango attribute such as: label, min alarm, max alarm, unit,...etc.

The application programmer can easily hide any three columns among four. There is always one column, which cannot be hidden.

- **ScalarListViewer**: three columns, which can be hidden, are label, setPoint editor (setter), and property button. The “read” value column cannot be hidden. All the attributes, members of the Attributelist model should be scalar attributes. All attributes with another format (Spectrum) will be ignored.
- **NumberScalarListViewer**: three columns, which can be hidden, are label, setPoint editor (setter), and property button. The “read” value column cannot be hidden. All the attributes, members of the Attributelist model should be scalar and numerical. All attributes with another type (String) and / or format (Spectrum) will be ignored.
- **ScalarListSetter**: three columns, which can be hidden, are label, “read” value, and property button. The setPoint editor (setter) column cannot be hidden. All the attributes, members of the attributeList model must be scalar and writable. The read-only attributes members of the attributeList model are ignored.

ScalarListViewers and ScalarListSetters (Flash Demo)
**View a list of device commands**

There is only one class provided for the command list viewing: **CommandComboViewer**. This viewer is based on the Swing “JComboBox”. The user can select any of the commands displayed in the list and send it to the device. The selection of an item in this list leads to the execution of the device command.

The viewers studied above (StateViewer, StatusViewer, ScalarListViewer and CommandComboViewer) can be used to build a generic tango device panel.

**A generic tango device panel**

The application we try to build in this tutorial is a generic tango device panel, which displays all the **scalar** attributes (no spectrum attribute, no image attribute) of a device and gives access to all commands of the same device. The application is generic because it has no knowledge of the attribute names and command names of the device. The device name should be passed as a parameter through the class constructor so that this panel can be used for any Tango device.

The ATK viewers we will use for this exercise are:

1. **StateViewer** (fr.esrf.tangoatk.widget.attribute.StateViewer)
2. **StatusViewer** (fr.esrf.tangoatk.widget.attribute.StatusViewer)
4. **CommandComboViewer** (fr.esrf.tangoatk.widget.command.CommandComboViewer)

The two last viewers are so-called “list viewers”. It means that, their corresponding model should not be an individual attribute or an individual command. Their corresponding model should be respectively an attribute list and a command list.

[Generic single device panel (Flash demo)]
ATK Guided Tour

In this chapter you will study the essential components of the ATK starting with the simplest ones used to visualize individual tango attributes and / or tango commands. The final part of this chapter is dedicated to the synoptic system provided with ATK. You can study this chapter in any order.

Scalar attributes

A scalar attribute is a Tango attribute whose format is Scalar whatever the data type of the attribute. In this chapter we will see how to view and / or set a single scalar attribute. We will also see how to view a collection of scalar attributes.

One single scalar attribute

Use a generic scalar attribute viewer (used to view and / or to set)

This solution consists of using the same viewer for any type of scalar attributes (number, string, boolean). The attributeList viewers such as ScalarListViewer can be used to view a single scalar attribute. All you have to do is to build an attributeList in which you add only one single scalar attribute, which is the one you want to view. Create a ScalarListViewer and set it’s model to this attributeList with one single attribute inside. See the code sample below:

```java
AttributeList attl = new AttributeList;
Try
{
    attl.add("my/test/device/onescalaratt");
    ScalarListViewer slv = new ScalarListViewer();
    sv.setModel(attl);
}
catch ()
{
}
```

The use of ScalarListViewer even for an individual attribute allows that the attribute value is displayed and formatted with it’s unit and eventually accompanied of it’s label, a value setter, and a pushbutton to access and to edit the other attribute properties.

Moreover the ScalarListViewer automatically uses the appropriate viewer according to the type of the attribute. For example a BooleanCheckBoxViewer is used for the Boolean attributes and a SimpleScalarViewer is used for numerical and string attributes. For this reason the use of scalarListViewer makes the application code to be independent of the type of the scalar attribute to be displayed.
The ScalarListViewer is used to display the read value of the attribute and also to set the attribute if the attribute is writable.

By hiding one or the other part of the scalarListViewer (label, setter, propertyButton) you can adapt the display to what you really want to make available to the application’s user. The screen shots below show the same scalar attribute displayed always with a ScalarListViewer. From left to right, the propertyButton, the setter and finally the label have been hidden.

**Using a specific viewer / setter adapted to the attribute type**

The use of specific viewers is dependent on the type of the scalar attribute to view and or to set. Normally a specific ATK viewer is designed either to display the read value of the attribute or to set the setPoint value of a writable attribute. But the specific ATK viewer generally does not do both of them. As we have seen before the list viewers (generic attribute viewers) can do both of these two functions read / write.

The specific viewer to use depends on the data type of the attribute and the fact that we want to use it for setting the attribute or only to display the read value. Therefore the source code also depends on the type of the attribute and the viewer. The code sample below is given for a NumberScalar attribute displayed by a SimpleScalarViewer. This code sample can be modified and adapted to other attribute types and viewers or setters.

```java
AttributeList attl = new AttributeList;
Try
{
    INumberScalar ins = (INumberScalar) attl.add("my/test/device/oneNumberScalarAtt");
    SimpleScalarViewer ssv = new SimpleScalarViewer();
    slv.setModel(ins);
}
catch()
{
}
```

Note that when using individual attribute viewers (instead of attribute list viewers) we need to keep a reference to the scalar attribute (“ins” in the code sample) and use it to set the model of the scalar attribute viewer.

The code sample above has been adapted so that instead of viewing the read value of the attribute we want to set the setPoint value of it.
AttributeList attl = new AttributeList;

Try {
    INumberScalar ins = (INumberScalar) attl.add("my/test/device/oneNumberScalarAtt");
    NumberScalarWheelEditor nswe = new NumberScalarWheelEditor();
    nswe.setModel(ins);
}

catch ()
{
}

NumberScalar attributes
By number scalar attribute we mean any Tango Attribute whose format is “Scalar” and whose data type is one of the numerical types. No matter if it’s a DevLong, DevDouble, or whatever numerical type.

There are several viewers, which can be used to display the “read” value of a Number Scalar attribute. There are also several classes in ATK provided for setting the value of a number scalar attribute

1. **SimpleScalarViewer** : can be used to display the read value of a NumberScalar or a StringScalar attribute. The value of the NumberScalar attribute is formatted according to the “format” attribute property. The attribute value is displayed followed by it’s unit (the attribute property unit). This viewer is actually the one used by ScalarListViewer to display the value of any Number or String scalar attribute.

2. **NumberScalarViewer** : can be used to display the read value of a NumberScalar. This viewer has a different character spacing and does not display the unit. This viewer should be used if you wish to align vertically the read value of a numberScalar attribute with it’s setPoint value displayed with a NumberScalarWheelEditor.

3. **NumberScalarProgressBar** : gives a view of the attribute based on a progress bar.

4. **NumberScalarWheelEditor** : displays the setpoint value of a NumberScalar and the user can use the top and bottom arrow buttons to set the NumberScalar attribute value. The value of the NumberScalar attribute is formatted according to the “format” attribute property. The unit is not displayed. This component is the default component used for setting a NumberScalar attribute in ScalarListViewer.

5. **NumberScalarComboEditor** : allows to set the value of a number scalar attribute by selecting the value in a list of predefined possible values. The possible values are formatted according to the “format” attribute property and the unit property is displayed with these values. If a list of predefined possible values are defined for the attribute the ScalarListViewer will automatically use this component instead of the default one (NumberScalarWheelEditor) to set the attribute.
The figure below shows the screen shots of the viewers.

StringScalar attributes
By string scalar attribute we mean any Tango Attribute whose format is “Scalar” and whose data type is DevString.

1. The **SimpleScalarViewer** is used to display the value of a string scalar attribute. This viewer is the one used by ScalarListViewer to display the read value of a string scalar attribute.
2. **StringScalarEditor** : displays the set value of a StringScalar and the user can type inside the text field to set the value of the StringScalar attribute. This component is the default component used for setting a StringScalar attribute in ScalarListViewer.
3. **StringScalarComboEditor** : allows to set the value of a StringScalar attribute by selecting the value in a list of predefined possible values. If a list of predefined possible values are defined for the attribute the ScalarListViewer will automatically use this component instead of the default one (StringScalarEditor) to set the attribute.
The figure below shows the screen shots for the “viewer” and “setter” components provided for StringScalar attributes.

**SimpleScalarViewer**

**StringScalarEditor**

**StringScalarComboEditor**

**BooleanScalar attributes**

By boolean scalar attribute we mean any Tango Attribute whose format is “Scalar” and whose data type is DevBoolean.

1. **BooleanScalarCheckBoxViewer** is used to view and to set the value of a boolean scalar attribute. In fact the BooleanScalarCheckBoxViewer is a mixed component. It’s a viewer and a setter. This component is used in ScalarListViewer to display the read value of the Boolean attributes.
2. **SignalScalarLightViewer** is used to display the read value of a Boolean Scalar attribute.
3. **BooleanScalarComboEditor** : this component is the default component used in ScalarListViewer to set a boolean attribute. This component refreshes it’s view according to the change in the “setpoint” value of the boolean attribute.
4. **SignalScalarButtonSetter** : this component is a pushbutton which is used to set the value of a boolean attribute always to the same value. The value (true or false) which is sent to the attribute at each click on the pushbutton is defined when the component is instantiated.

**Averaging** **BooleanScalarCheckBoxViewer**

**SignalScalarLightViewer** *(default behaviour)*

**SignalScalarLightViewer** *(customized)*
EnumScalar attributes
The Enumerated attributes will be available within the future releases of Tango but for the
time being, Tango does not provide such a feature.

Nevertheless under some conditions ATK provides the possibility to see some numeric and
scalar attributes as enumerated attributes.

The condition for numeric scalar attributes to be considered as enumerated scalar attributes
(EnumScalar) is:
- The attribute data type should be DevShort.
- A property whose name is EnumLabels should be defined for the attribute.
- Eventually (it is optional) another property whose name is EnumSetExclusion can
  also be defined for the attribute

The first property (EnumLabels) specifies the list of all the possible values the attribute can
have. This list is an ordered list. Each label in the list corresponds to a numeric value. The first
label is always associated to zero (0).

The second property (EnumSetExclusion) if specified, gives the list of labels, which can never
be used to set the attribute. The labels specified by this property are possible values the
attribute can have when we read it but they can not be used as possible set values. If this
property is not specified, all the values / labels specified in EnumLabels, can be used to set the
attribute value.

In the screen shot below, you can see how a DevShort scalar attribute (jlp/test/1/att_six) can
be configured using JIVE such that ATK considers it as an enumerated attribute:
In this example the possible values for jlp/test/1/att_six are 0, 1, 2, and 3 respectively associated to “first val”, “second one”, “third” and “last”. Note that the value “third”=2 can be read from the attribute but can never be used to set the attribute.

The **SimpleEnumScalarViewer** is used to display the read value of a enumerated scalar attribute. This component is used by the ScalarListViewer to view the enumerated attributes. *Do not forget that enumerated attribute is an ATK concept and in Tango the real type of the attribute is DevShort.*

The **SimpleEnumScalarViewer** reads the value of the attribute and displays the “label” corresponding to the read value. This label is one of those specified by the property *EnumLabels* associated to the attribute.

The **EnumScalarComboEditor** is used to set an EnumScalar attribute. This component is used by ScalarListViewer to set the enumerated attributes. This component displays the setPoint value of the attribute converting it to a label specified by the attribute property *EnumLabels*. In the comboBox drop down list all the labels specified by *EnumLabels* property are displayed, excepted those defined in *EnumSetExclusion* property.

The picture above shows at the left side a SimpleEnumScalarViewer and at the right side an **EnumScalarComboEditor** both associated with the same attribute jlp/test/1/att_six. As you
can see the label “third” is not proposed in the comboBox drop down list for setting since this label is included in the *EnumSetExclusion* property. But if this value (numerical value = 2) is read on the attribute the `SimpleEnumScalarViewer` on the left side will display “third”.

**DevState Scalar attributes**

By DevState scalar attribute we mean any Tango attribute whose format is “Scalar” and whose data type is DevState. The “StateViewer” is one of the viewers used to view a DevState scalar attribute. The state is converted to a color by the ATK state viewers. The following color – state correspondence is used by all the ATK viewers:

<table>
<thead>
<tr>
<th>State</th>
<th>Colour</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON, OPEN, EXTRACT</td>
<td>Green</td>
</tr>
<tr>
<td>OFF, CLOSE, INSERT</td>
<td>White</td>
</tr>
<tr>
<td>MOVING, RUNNING</td>
<td>Light Blue</td>
</tr>
<tr>
<td>STANDBY</td>
<td>Yellow</td>
</tr>
<tr>
<td>FAULT</td>
<td>Red</td>
</tr>
<tr>
<td>INIT</td>
<td>Beige</td>
</tr>
<tr>
<td>ALARM</td>
<td>Orange</td>
</tr>
<tr>
<td>DISABLE</td>
<td>Magenta</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>Grey</td>
</tr>
</tbody>
</table>

As you can see in the table above the **Open** and **Extract** states are represented by the *green* color. Green color represents a normal operational state. But the **Close** and **Insert** states are represented by the *white* color which means abnormal operational state. In practice, in some cases the green color should be associated to “Close” instead of Open, because close state is the normal operational state of a particular device. The inversion of the colors can also be acceptable for Extract and Insert states in some cases.

ATK allows to invert the color correspondence only for “Open” and “Close” states and for “Extract” and “Insert” states.

To invert the color correspondence for “Open” and “Close” states the attribute property `OpenCloseInverted` should be set to **True**.

To invert the color correspondence for “Extract” and “Insert” states the attribute property `InsertExtractInverted` should be set to **True**.

1. **StateViewer** is used to view the read value of a DevState Scalar attribute. The state is represented as a colored rectangle besides the name or the alias of the Tango Device.

ATK does not provide any component for setting a DevStateScalar attribute.
A Collection of scalar attributes

AttributeList viewers
As we have already studied them the attribute list viewers are the components which use an attribute list as their model (not an individual attribute). They display only the scalar attributes and ignore the non scalar attributes contained in the attribute list. They automatically choose the appropriate viewer depending on the type of the attribute. ATK proposes 3 attribute list viewers: NumberScalarListViewer, ScalarListViewer, ScalarListSetter. Please have a look into the section: View a list of scalar attributes.

A set of scalar attributes in a table (MultiScalarTableViewer)
The MultiScalarTableViewer is used to view a collection of scalar attributes inside a table. Each attribute is associated to a cell. The MultiScalarTableViewer will select the appropriate scalar attribute viewer according to the type of the attribute (NumberScalar, StringScalar, BooleanScalar or EnumScalar). The viewer is used inside the corresponding cell to display the read value of the attribute.

The user can also set the attribute value. To do so, (s)he should double click inside the cell. This will display a set panel adapted to the type of the scalar attribute. A double click on a read-only attribute has no effect.

If the keyboard focus is on the table, when the mouse enters a cell a tooltip will display the precise tango name of the attribute.

Using the ATK MultiScalarTableViewer (Flash demo)

A set of DevStateScalar attributes (TabbedPaneDevStateScalarViewer)
The TabbedPaneDevStateScalarViewer is used to view a collection of state attributes in the titles of the panes of a tabbedPane. Each state attribute is added to the viewer by the call to addDevStateScalarModel. This method needs also the index of the tab to be associated to the state attribute. The screen shot below shows this viewer:

![TabbedPaneDevStateScalarViewer](image)

TabbedPaneDevStateScalarViewer  Each DevStateScalar attribute is used as the model of this viewer and associated to one of the tabs. The state attribute value is represented by the background color of the title of the corresponding tab. When the mouse enters the tab's title, a tooltip displays the name and the value of the underlying state attribute.
**Trend of Scalar attributes**

**The trend of number scalar attributes**
The ATK component `Trend` allows the user to follow the evolution of the value of one or more number scalar attributes during the time. `Trend` accepts an attribute list as model. The number scalar members of the attributeList can be plotted inside a chart during the time. Each `NumberScalar` attribute included in the attribute list will be read at the frequency of the refresh period and displayed as a separated plot.

[Using the ATK Trend (Flash demo)](image)

**The trend of boolean scalar attributes**
The ATK component `BooleanTrend` allows the user to follow the evolution of the value of one or more boolean scalar attributes during the time. `BooleanTrend` accepts an attribute list as model. The boolean scalar members of the attributeList can be plotted inside a chart during the time. Each `BooleanScalar` attribute included in the attribute list will be read at the frequency of the refresh period and displayed as a separated plot.
Spectrum attributes

A spectrum attribute is a Tango attribute whose format is Spectrum (one dimensional array) whatever the data type of the attribute. In this chapter we will see how to view and / or to set a single spectrum attribute. We will also see how to view a collection of spectrum attributes.

One single spectrum attribute

NumberSpectrum attributes

By number spectrum attribute we mean any Tango Attribute whose format is “Spectrum” and whose data type is one of the numerical types. No matter if it’s a DevLong, DevDouble, or whatever numerical type.

The **NumberSpectrumViewer** is used to display the read value of a number spectrum attribute. This viewer displays the spectrum attribute as a plot in a chart. The user can display the values inside the spectrum in a table using the mouse right button menus. You can use this viewer following the code sample below:

```java
AttributeList attl = new AttributeList;
Try
{
    INumberSpectrum spect = (INumberSpectrum) attl.add("my/test/device/onespectrumatt");
    NumberSpectrumViewer nsv = new NumberSpectrumViewer();
    nsv.setModel(spect);
}
catch()
{
}
```
The following screen shot shows a **numberSpectrumViewer**. Note that the table on the right, has been displayed using the chart menus under the right mouse button.

**ATK does not provide any component for setting a NumberSpectrum attribute.**

**StringSpectrum attributes**

By string spectrum attribute we mean any Tango Attribute whose format is “Spectrum” and whose data type is DevString.

The **SimpleStringSpectrumViewer** is used to display the value of a StringSpectrum attribute. The **SimpleStringSpectrumViewer** displays the spectrum attribute as a scrolled text. Each string element of the spectrum is displayed in a new line. The code sample is very similar to the one given in the previous section for the use of NumberSpectrumViewer. You just need to replace NumberSpectrumViewer by SimpleStringSpectrumViewer and replace INumberSpectrum by IStringSpectrum.
**DevStateSpectrum attributes**

By DevState spectrum attribute we mean any Tango Attribute whose format is “Spectrum” and whose data type is DevState.

The **DevStateSpectrumViewer** is used to display the value of a DevState Spectrum attribute. This viewer displays the elements of the state spectrum attribute vertically. Each elements is displayed in a line with three different areas: in the left a text label is displayed with the name of the attribute and the index of the element in the spectrum, in the middle a colored rectangle displays the state value and in the right side a text label displays the state value converted to a string.

![Attribute Values]

The label displayed on the left side of each element can be customized. By default this label is the attribute name + [ + index + ]. To define another label for the spectrum elements the tango attribute property **StateLabels** should be defined. In the example above, this attribute property has been defined using JIVE:

```
DevStateSpectrumViewer
```

The label on the left is the name of the attribute + index of the element in the spectrum. But the viewer can also display a customized label using the tango attribute property **StateLabels**.
A collection of Spectrum attributes

A set of NumberSpectrum attributes in one single chart
The MultiNumberSpectrumViewer is used to view a collection of number spectrum attributes inside a chart. Each number spectrum attribute is displayed as an individual plot. All plots are displayed inside the same.

The following code example uses the MultiNumberSpectrumViewer to view 2 NumberSpectrum attributes: "jlp/test/1/att_spectrum", "jlp/test/2/att_spectrum".

```java
INumberSpectrum ins;
AttributeList attl = new AttributeList();
MultiNumberSpectrumViewer mnsv = new MultiNumberSpectrumViewer();

Try{
    ins = (INumberSpectrum) attl.add("jlp/test/1/att_spectrum");
    mnsv.addNumberSpectrumModel(ins);
    ins = (INumberSpectrum) attl.add("jlp/test/2/att_spectrum");
    mnsv.addNumberSpectrumModel(ins);
    ...
}
catch (Exception ex)
{
    System.out.println("Cannot connect device");
    ex.printStackTrace();
}
```
The following screen shot shows the result of the execution of this code example:

As you can see, this viewer associates each attribute plot to a colour in the order the attributes have been added by the call to “addNumberSpectrumModel” method. The user has the possibility to change the visual aspects (colour, line width, affine transform, marker, …etc.) of each plot.

**Trend of Spectrum attributes**

**The trend of number spectrum attributes**

There are two ATK viewers which allow the user to follow the evolution of the values of the array elements of a NumberSpectrum attribute.

1. **NumberSpectrumTrendViewer**
2. **NumberSpectrumItemTrend**

The first component (NumberSpectrumTrendViewer) will display and follows the evolution of *ALL* elements of the spectrum.

The second component (NumberSpectrumItemTrend) is more flexible. It can display the trend of all elements of the spectrum as the first one does. But you can also specify which elements (items) of the spectrum you want to see in the trend.

The following code sample illustrates the use of the **NumberSpectrumItemTrend**.

```java
NumberSpectrumItemTrend nsit = new NumberSpectrumItemTrend();
try {
    ins = (INumberSpectrum) attList.add("fp/test/1/wave");
    nsit.setPlotAll(false);
    nsit.setModel(ins);
    nsit.plotItem(30, NumberSpectrumItemTrend.AXIS_Y1, "wave[30]");
    nsit.plotItem(1, NumberSpectrumItemTrend.AXIS_Y1, "wave[1]");
} catch (Exception ex)
```
System.out.println("caught exception : " + ex.getMessage());
    System.exit(-1);
}

mainFrame = new JFrame();
mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
mainFrame.getContentPane().add(nsit);

attList.startRefresher();

mainFrame.setSize(800,600);
mainFrame.pack();
mainFrame.setVisible(true);

// Test hide and show item!
for (int i=0; i<10; i++)
{
    try
    {
        Thread.sleep(5000);
    }
    catch(Exception ex)
    {
    }
    nsit.hideItem(7);
    try
    {
        Thread.sleep(5000);
    }
    catch(Exception ex)
    {
    }
    nsit.showItem(7);
}
}

AttributeList attl = new AttributeList();
StringImageTableViewer sitv = new StringImageTableViewer();
Try
{
    isi = (IStringImage) attl.add("my/test/dev/att_str_image");
    sitv.setAttModel(isi);
}

catch (Exception ex)
{
    System.out.println("Cannot connect device");
    ex.printStackTrace();
}

The screenShot below show the NumberSpectrumItemTrend used for only two elements (index 1 and index 30) of a numberSpectrum attribute:
Image attributes

An image attribute is a Tango attribute whose format is Image (2 dimensional array) whatever the data type of the attribute. In this chapter we will see how to view and / or set a single image attribute.

One single image attribute

NumberImage attributes

By number image attribute we mean any Tango Attribute whose format is “Image” (2 dimensional array) and whose data type is one of the numerical types. No matter if it’s a DevLong, DevDouble, or whatever numerical type. All the attributes which are not a video image such as a 2 dimensional array of numeric data, are considered to be NumberImage attributes.

The NumberImageViewer is used to display the value of a 2 dimensional array of numeric data (not a video image). The following code sample illustrates the use of the NumberImageViewer.

```java
INumberImage                  ini;
AttributeList                 attl = new AttributeList();
NumberImageViewer             niv = new NumberImageViewer();
Try
{
    ini = (INumberImage) attl.add("jlp/test/1/att_image");
    niv.setModel(ini);
}
catch (Exception ex)
{
    System.out.println("Cannot connect device");
    ex.printStackTrace();
}
```
The following screen shot shows the result of the execution of the code sample above:

ATK does not provide any component for setting a NumberImage attribute.

**RawImage attributes**

RawImage attributes are used for the images coming from video camera, CCDs. By convention the Raw Image data (image coming from video camera, CCDs) should be sent as attributes with format = image and data type = DevUchar. The RawImage feature is not available for the moment in the standard ATK. We are waiting for a tango definition of CCD / vidéo camera images with different formats (jpeg, png, ...) in order to implement RawImages in standard ATK. The ATK RawImage viewer will be supported when the attribute data type “DevEncoded” will be available in Tango API.

*ATK does not provide any component for setting a RawImage attribute.*
**StringImage attributes**

By string image attribute we mean any Tango Attribute whose format is “Image” (2 dimensional array) and whose data type is DevString.

The *StringImageTableViewer* is used to view a StringImage attribute (a 2 dimensional array of string). Each element of the attribute array will be displayed in a cell in a swing JTable.

The following code sample illustrates the use of the *StringImageTableViewer*.

```java
IStringImage isi;
AttributeList attl = new AttributeList();
StringImageTableViewer sitv = new StringImageTableViewer (

Try
{
    isi = (IStringImage) attl.add("my/test/dev/att_str_image");
    sitv.setAttModel(isi);
}
catch (Exception ex)
{
    System.out.println("Cannot connect device");
    ex.printStackTrace();
}
```

*ATK does not provide any component for setting a StringImage attribute.*
Device Commands

Display a single tango device command

There are several viewers available to represent a Tango device command. The choice of the viewer depends on the type of the input and output argument of the command. For example the `VoidVoidCommandViewer` is used for all commands with no input argument and no output argument.

Commands with no input and no output argument (VoidVoidCommand)

The commands with no input and no output argument are called VoidVoid commands in ATK. The following list presents all the command viewers suitable for VoidVoidCommands:

1. `VoidVoidCommandViewer`: is a sub-classes of swing JButton. The label of the JButton is the name of the command. A click on a VoidVoidCommandViewer will immediately launch the execution of the corresponding command on the tango device. When the mouse enters the button a tooltip will display the name of the tango device on which the command will be executed.
2. `ConfirmCommandViewer`: is also a sub-classes of swing JButton. The difference with previous viewer is that the click on the ConfirmCommandViewer button will just popup a confirmation dialog window. The device server’s command is executed only if the user confirms the dialog window. As for the VoidVoidCommandViewer when the mouse enters the button a tooltip will display the name of the tango device on which the command will be executed.

![VoidVoidCommandViewer example](example.png)

![ConfirmCommandViewer example](example.png)
The code sample below can be used for these two viewers indifferently:

```java
ICommand ic;
CommandList cmdl = new CommandList();
VoidVoidCommandViewer vvcv = new VoidVoidCommandViewer();
Try
{
    ic = (ICommand)cmdl.add("elin-gun-beam/Off");
    vvcv.setAttModel(ic);
}
catch (Exception ex)
{
    System.out.println("Cannot connect device");
    ex.printStackTrace();
}
```

Commands with DevBoolean input argument and no output argument
(BooleanVoidCommand)

The commands with DevBoolean input argument and no output argument are called BooleanVoid commands in ATK. The following list presents all the command viewers suitable for BooleanVoidCommands:

1. **OnOffCheckBoxCommandViewer** is a sub-classes of swing JCheckBox. A click on a OnOffCheckBoxCommandViewer will immediately execute the corresponding command on the tango device. The value of the input parameter passed to the device command depends on the state of the checkBox. If the checkBox is selected the device command is called with “true” parameter, otherwise the “false” parameter is sent to the command.

2. **OnOffSwitchCommandViewer**: A click on a OnOffSwitchCommandViewer will immediately execute the corresponding command on the tango device. The value of the input parameter passed to the device command depends on the state of the switch. The difference with the previous viewer is only in the graphical representation.

   ![OnOffCheckboxCommandViewer](image)

   **OnOffCheckboxCommandViewer** a click on the checkbox will execute the "averaging" command of the tango device with a boolean parameter. The boolean parameter passed to the command is true if the checkbox is selected and false if the checkbox is not selected.

   ![OnOffSwitchCommandViewer](image)

   **OnOffSwitchCommandViewer** a click on the switch button will execute the "averaging" command of the tango device with a boolean parameter. The value of the boolean parameter passed to the command depends on the position of the switch button.
Commands with DevString input argument and no output argument

The following list presents all the command viewers suitable for the commands with DevString input argument and no output argument.

1. **OptionComboCommandViewer**: is a sub-class of Swing JComboBox. The limited possibilities for the input strings are displayed in the combobox drop down list. A click in this list will launch the execution of the Tango device command with the input parameter equal to the item selected in the combobox item list.

Commands with any type of input argument and any type of output argument

The following list presents all the command viewers suitable “any” tango command

1. **AnyCommandViewer**: is a sub-class of Swing JButton. This viewer is convenient for the tango device commands with input arguments and/or output arguments of any type. A click on the button will display a window (see the screen shot below) in which the user can enter the input argument, click on execute will execute the command with the specified input argument and if there is any output argument, it will be displayed in the lower area (scrolled text area) of this window.

![Screen shot of AnyCommandViewer](image)
Display a collection of tango device commands

ATK provides a viewer **CommandComboViewer** to display a collection of device commands in a Combo drop down list. Each element of this list acts as a “VoidVoid CommandViewer” if the command has no input and no output argument. The command list element acts as “AnyCommandViewer” if the command has an input and / or output argument.

![Command Combo Viewer](image)

The model for this viewer is a CommandList. All the members of the commandList will be displayed in the comboBox drop down list no matter what is the type of their input and / or output arguments.

When one of the items of the list is selected:

- If the command has no input argument it is immediately executed.
- If the command needs an input argument a “anyCommandViewer” window will be displayed asking for the argument to be entered.
Error Handling

All the exceptions thrown by Tango and caught by ATK are transformed into an ATK error event. Below is a list of some situations in which the exceptions are caught by ATK and transformed into an ATK error event:

- Tango device access timeout during the refreshing of the attributes
- Tango device access timeout during the actions like: setting the value of an attribute, execution of a command
- Exceptions thrown by the device servers because of a non-authorized action or value setting

What is important to note is that normally all the exceptions thrown by the Tango API are caught inside ATK and transformed into error events. The only exception, which is not transformed to an ATK error, is the ConnectionException. This exception is thrown by ATK if and only if the initial connection to the Tango device fails. So apart from the ConnectionException the ATK application programmer does not need to catch any Tango related exception.
There are two kinds of errors in ATK. The first type of errors, called “Error”, is produced when the Tango DevFailed Exception occurs during the reading of an attribute or during the execution of a command. The second type of errors, called “SetError”, is produced when the Tango DevFailed Exception occurs during the setting value of an attribute. This is done to be able to make a clear separation between the errors which happen during the setting of an attribute and those which happen during the reading of the same attribute.

In addition to the ATK error events generated, ATK provides two classes of error viewers: **ErrorHistory** and **ErrorPopup**. They are the graphical viewer classes which listen to ATK error events and display the error to the application end user.

### How to handle and display errors

The provided error viewer classes can be used to collect and to display ATK errors generated during the application session. Here are the steps to perform to handle errors:

1. **Create one or more ErrorViewer(s)**
   ```java
   ErrorHistory errh = new ErrorHistory();
   ErrorPopup errorpopup = ErrorPopup.getInstance();
   ```

2. **Add one or more error viewer(s) as error listeners to the empty attribute list just after it’s instantiation**
   ```java
   AttributeList attl = new AttributeList();
   attl.addErrorListener(errh);
   attl.addSetErrorListener(errorpopup);
   ```

3. **Add one or more error viewer(s) as error listeners to the empty command list just after it’s instantiation**
   ```java
   CommandList cmdl = new CommandList();
   cmdl.addErrorListener(errh);
   ```

4. **Connect to the attributes by adding them to the attribute list**
   ```java
   attl.add(att_one);
   attl.add(att_two);
   ```

5. **Connect to the commands by adding them to the command list**
   ```java
   cmdl.add(cmd_one);
   cmdl.add(cmdm_two);
   ```

6. **Start the attribute list refresher**
   ```java
   attl.startRefresher();
   ```

The error viewers are registered as error listeners of the attribute list and the command list. This way they will be registered as the error listeners of all the members added to these lists. It is very important to register them as error listeners of the list before the first adding of the elements.
**Error Viewers**

There are two error viewer classes provided by ATK: **ErrorHistory** and **ErrorPopup**. To use them the application programmer should add them as error listeners to either attribute and command lists or to the attribute and command entities directly.

**ErrorHistory**
The ErrorHistory viewer is used to log all of the errors it receives and keep the history of all the errors received. It will display the list of these errors. If the same error occurs repeatedly, to save place in the window, only the timestamp of the error is changed. This way only the date and the time of the last time the error occurred is displayed.

The code sample below shows how to use ErrorHistory:

```java
ErrorHistory eh = new ErrorHistory();
AttributeList attl = new AttributeList();
attl.addErrorListener(eh);
attl.addSetErrorListener(eh);
```

The call to “**addErrorListener**” will add the ErrorHistory as a listener for all errors excepted those happening during the attribute set value. If we want to log into the ErrorHistory the attribute setting errors we should call the “**addSetErrorListener**” in addition to “**addErrorListener**”.

A right click on one of the errors displayed in the list, will display detailed information about that particular error. “Show Error” will display on the right panel the Tango error stack.
**ErrorPopup**

The ErrorPopup viewer is a singleton class in ATK. This viewer is a dialog window which pops up as soon as it receives an error. The error description is displayed and the user can get the detailed description of the error. The ErrorPopup window waits for the user click to disappear.

Normally the ErrorPopup should NOT be used for the errors which occur during the attribute refreshing. It should be used for errors which occur rarely like the setting of an attribute or the execution of a command.

The code sample below shows how to use ErrorPopup:

```java
    ErrorPopup errpp = ErrorPopup.getInstance();
    AttributeList attl = new AttributeList();
    CommandList cmdl = new CommandList();
    attl.addSetErrorListener(errpp);
    cmdl.addErrorListener(errpp);
```

Note that the ErrorPopup is only added as “SetErrorListener” to the attribute list.
Synoptic drawing and programming

ATK provides a complete synoptic system. As already mentioned in the introduction, the main idea of the synoptic drawing and viewing system is to provide the application designer with a simple and a flexible way to draw a synoptic and to animate it at runtime according to the values and states read from the control system.

What is a synoptic application?

In an application based on a synoptic the user can see a “free style” drawing, in which different parts can report on the tango device states and/or the tango attribute values of the control system. We say that the drawing is “animated” at run-time according to the values / states of the control system objects.

The following picture is the snapshot of the ESRF Linac control application based on a synoptic. The synoptic is the drawing in the center with the background color in blue.

As you can see the drawing components in the synoptic have different colors according to the device state attribute to which they are linked. For example the drawing component linked to “elin/gun/aux/State” is colored in orange because the value of this state attribute is Alarm. Moreover you can see the red arrow (Beam Stop) on the slider pointing to the value of the tango attribute “elin/master/op/SRCT_limit” which is 92 as it is also represented outside of the synoptic on the top of the window.

What kind of animations are provided at run-time?

The run-time behavior of the synoptic is predefined in ATK and it depends on the type of the graphic object (free drawing, dynos, sliders, buttons, …) on one hand and the tango control
object to which it is linked (state attribute, numerical attribute, boolean attribute, tango command …). The exact run-time behaviour in each case will be discussed in a further section.

**Draw the synoptic : Use Jdraw editor**

ATK includes a graphical editor, to design the graphic shape of the synoptic. The editor is called **Jdraw**. This editor is included in ATK so you don’t need to download any specific jarfile.

To launch **jdraw** you should start the following class :

```
fr.esrf.tangoatk.widget.util.jdraw.JdrawEditorFrame
```

Once the jdraw is started you will see the following window :

You can now start to draw the synoptic. As in any drawing editor you can group basic objects (like rectangle, circle, lines, …) to obtain more elaborated shapes. Once the drawing is done you need to associate parts of the drawing to a Tango control system object. Click on the following link to view a Flash demo of how to draw a simple tango synoptic.

[First synoptic (Flash Demo)](Flash Demo)
Test the synoptic: “Tango Synoptic view”
As soon as the synoptic is saved in a file, it’s run time behavior can be tested. Inside Jdraw, select “Tango Synoptic view” from the “Views” pulldown menu to test the run time behavior of the synoptic.

You can also start the same simple synoptic application outside jdraw editor. This application is called SimpleSynopticAppli and is included in ATK. You don’t need any specific jarfile to use it. To launch SimpleSynopticAppli you should start the following class:

```
fr.esrf.tangoatk.widget.jdraw.SimpleSynopticAppli
```

You can pass the absolute path of the synoptic file as the argument to the SimpleSynopticAppli. In case no argument is passed on the command line, the application will popup a file selection dialog in order to get the name of the synoptic file to load.

The following screenshot shows the SimpleSynopticAppli with the synoptic file loaded.
As you can see the rectangle shows the value of fp/test/1/state attribute. The state attribute value is represented by its corresponding color (blue). Moreover you can see the value of the “fp/test/1/double_scalar” value displayed by the SimpleScalarViewer inside the synoptic.

If the mouse enters the blue rectangle (while the focus is inside the synoptic window), a tooltip will display the name of the Tango attribute associated to the rectangle. The same goes for the region where the numeric value (177.83) is displayed.

A mouse click inside the blue rectangle will launch an AtkPanel for the device fp/test/1. The AtkPanel is started by default in read-only mode: device commands and attribute setters are not displayed.

Conclusion:
✓ A synoptic drawing can easily be made using the drawing tool Jdraw included in ATK.
✓ The association between the graphic components inside the drawing and the control system Tango objects are made through the name of the graphic components.
✓ A Simple Synoptic application is provided to test the run-time behaviour of the synoptic. It can be started directly from the Jdraw editor’s menubar or outside Jdraw.

Jdraw editor
We will not explain in detail Jdraw editor. In fact, it is a very intuitive editor and you can just try it to get experience with it. Nevertheless there are some Jdraw features which will be described in this section.

Jdraw libraries
You can draw your own standard shapes and save them in a file such that they can be used in other synoptic files.
1. Draw your standard / predefined shapes in the Jdraw editor.
2. Save the file by naming it with a “.lib” suffix.
3. Move your “.lib” file to a well defined location on your disk.
4. Set the “LIBPATH” environment variable to the folder where is located your “.lib” file.
5. Start the jdraw java machine with “-DLIBPATH=$LIBPATH”.
6. Note on the top right corner the “Palette Libraries” ComboBox. You should see inside the drop down list the name of the “.lib” file.

When the library name is selected it’s content is displayed in a separate window. You can simply click one component in the library window and click the jdraw window to add it into your drawing. See the screenshot below:
ATK Viewers in Jdraw

A small set of ATK viewers are available in Jdraw so that they can be added inside the synoptic drawing. To add one of them click on “ATK Swing” button and select an appropriate viewer from the list. When using the ATK viewers you may need to set some of their “bean properties” to make them behave as you wish. A subset of the properties of each viewer is accessible through Jdraw. To see and to edit those properties, double click the atk viewer, then select the “Extension” tab in the property window. For example in the screen shot below you can see all the bean properties available in Jdraw for the SimpleScalarViewer.

Dynamic Objects (Dynos)

Dynamic Objects also called Dynos in Jdraw are the graphic components for which the user has defined a dynamic (run-time) behavior. For example a Dyno can be any graphic component associated to a numeric tango attribute and for which the user has defined a specific background color depending on the value of the attribute. You can see how to create and use the dynos in Jdraw in the following flash demo.

Dynamic Objects in Jdraw (Flash Demo)
Panel class definition
In a synoptic application when a graphic component is clicked by the user, in most cases, we
need to launch a specific panel. In Jdraw you have the possibility to define the name of the
class you want to start when the Jdraw object is clicked. To associated a Jdraw graphic
component to a panel follow the steps below :
1. Double click the Jdraw graphic object to show the Properties window
2. Select the Extension Tab inside the Properties window
3. Click on the “New” button to add a new extension and give it the name “className”
4. Type in the fully defined class name of the panel you want to show, in the value field
attached to className extension
5. Optionnally click on the “New” button to add another extension and give it the name
“classParam”
6. Type in the string which is passed to the constructor of the panel class

The “panel class” defined with “className” extension :
• Should be a subclass of JFrame or Jdialog
• Must have a constructor with a String parameter (even if the parameter is ignored)
• Should not call system.exit() when it’s window is closed

Conclusion
✓ The specific full qualified panel class name should be specified in the “Extension” tab
of the property window under the name of “className”.
✓ The string parameter which will be passed to the panel class constructor can also be
defined in the “Extension” tab of the property window under the name of
“classParam”.
✓ If the classParam is not defined the constructor of the class is called with the name of
the jdraw graphic object which has been clicked.
✓ If the className is not defined and the jdraw graphic object is associated to a Tango
state attribute, atkpanel will be started in read only mode.
Include the Synoptic in an ATK application

Once the synoptic is drawn and well tested, it can be used through the generic application *SimpleSynopticAppli*. To launch the *SimpleSynopticAppli* start the following class:

```
fr.esrf.tangoatk.widget.jdraw.SimpleSynopticAppli
```

In most cases, the synoptic should be integrated inside a specific ATK application in the middle of other ATK viewers.

As you can see in the screen shot the synoptic is only part of the application's main window. There are other ATK attribute and command viewers outside of the synoptic area. Moreover there is also a specific menu bar with a lot of application specific commands.

**SynopticFileViewer**

ATK provides a viewer called *SynopticFileViewer* which belongs to the package: `fr.esrf.tangoatk.widget.jdraw`. This viewer can be used as any other ATK viewer. It can be added into any Swing container. It can also be added to a Java IDE palette (for example Netbeans palette) as a Java Bean.

Once the SynopticFileViewer is instantiated, the programmer should specify the synoptic file to be loaded by the viewer. There are two methods for synoptic file specification:

1. Load the synoptic from a file specified by a path name on the disk
2. Load the synoptic from an Input Stream Reader
Load the synoptic from a file

The application programmer will specify the file path name of the synoptic file to load. The drawback of this option is that the application programmer must know the absolute path name of the synoptic file and this path name is constant even if the application is deployed in different hosts and sites.

The following code sample shows how to use a SynopticFileViewer and specify the synoptic file to load:

```java
SynopticFileViewer sfv = new SynopticFileViewer();
sfv.setToolTipMode(TangoSynopticHandler.TOOL_TIP_NAME);
sfv.setAutoZoom(true);
try {
    sfv.setJdrawFileName("/my/root/dir/jdraw_file mySynoptic.jdw");
} 
Catch (Exception ex) {}
```

The call to “setJdrawFileName” will load the synoptic file if it can be found and opened, otherwise an exception is thrown.

Load the synoptic from an Input Stream Reader

The main advantage of this method is that the synoptic jdraw file can be included into the application jar file. An input stream reader is created through the file resource by the application code. This input stream reader is passed to the SynopticFileViewer to load the synoptic.

This option allows that the synoptic file is packed inside the application Jar file and we don’t make any assumption on the exact physical location of the synoptic file on the disk. The following code sample shows how to use a SynopticFileViewer and specify the synoptic file to load:

```java
SynopticFileViewer sfv = new SynopticFileViewer();
sfv.setToolTipMode(TangoSynopticHandler.TOOL_TIP_NAME);
sfv.setAutoZoom(true);
InputStreamReader inStrReader =null;
InputStream jdFileInStream = this.getClass().getResourceAsStream("/mypakcage/file.jdw");
if (jdFileInStream!=null)
inStrReader = new InputStreamReader(jdFileInStream);
if (inStrReader!=null)
{
    Try{
    sfv.loadSynopticFromStream(inStrReader);
    }
    Catch (Exception ex) {}
}
```

The call to “loadSynopticFromStream” will load the synoptic from the input stream if possible. In case of bad format or an empty stream (no component) an exception is thrown.
**Predefined run time behavior**

The synoptic file is loaded by ATK at run-time. All the run time animation / behavior is coded inside Atk class which loads the synoptic. All of the run time behavior is listed in this section.

**Tango State Attribute**

A Tango state attribute can be associated to any jdraw graphic object. From a simple drawing to a complex shape made of successive groups. A tango state attribute can also be associated to a Dyno.

**Associated to a Jdraw Object (not a Dyno)**

ATK will color the object according to the value of the state attribute. The state/color mapping is the same as the one used in all other parts / viewers of ATK.

- If the object is filled : the fill color is changed
- If the object is not filled : the line color is changed
- If the object is made of successive groups, the change is made recursively in each group until the basic graphic objects are reached. In this hierarchy of objects, the graphic objects whose name is “IgnoreRepaint” do not change their color at all.

**Associated to a Dyno (Dynamic Object)**

As described in the previous section a Dynamic Object (Dyno) has a specific dynamic behavior which has been defined during the drawing phase. In order to define your own behavior with a Dyno associated to a State attribute, you should define the mapping between each different tango state numeric values and the caracteristique affected by the value.

It’s important to know that the Dyno will receive at run time a numeric value associated to the state attribute value. You can find the mapping between the numeric values and the tango state values in the Tango documentation:

**User interaction**

When the **mouse enters** the graphic component associated to the state attribute, the name of the state attribute is displayed inside a **tooltip**.

When an object associated to a state attribute is clicked by the user at run time, ATK tries to popup a panel.

- If the **className** extension is defined, the class is instantiated using a constructor with a String parameter.
- If the **className** extension is not defined the AtkPanel in read-only mode is instantiated.
- If the **classParam** extension is defined, the string is passed as the argument to the constructor of the panel class.
- If the **classParam** extension is not defined, the device name behind the state attribute is passed as the argument to the constructor of the panel class.
**Tango Numeric Attribute**

A Tango numeric attribute can be associated to a Dyno (Dynamic Object) or to an adapted Atk Viewer (for example SimpleScalarViewer).

**Associated to a Dyno (Dynamic Object)**

As described in the previous sections a Dynamic Object (Dyno) has a specific dynamic behavior which has been defined during the drawing phase. In order to define your own behavior with a Dyno associated to a tango numeric attribute, you should define the mapping between different values of the tango attribute and the caracteristique affected by the value. You can for example associate “value intervals” to a caracteristique change. See the [Dynamic Objects in Jdraw](Flash Demo).

It’s important to know that the Dyno will receive at run time the numeric value of the tango attribute when it changes.

**Associated to an Atk Viewer (for example SimpleScalarViewer)**

The tango attribute will be set as the model of the AtkViewer (SimpleScalarViewer) and that’s it. All the run-time behavior is defined by the AtkViewer which is used. Some of the bean properties of the Atk Viewer are available in the extension Tab of the Jdraw properties window.

**User interaction**

When the **mouse enters** the graphic component associated to the tango numeric attribute, the name of the tango attribute is displayed inside a **tooltip**.

When an object associated to the tango attribute is selected by the user at run time, ATK tries to popup a panel:

- ✓ If the **“className”** extension is defined, the class is instantiated using a constructor with a String parameter.
- ✓ If the **“classParam”** extension is defined, the string is passed as the argument to the constructor of the panel class.
- ✓ If the **“classParam”** extension is not defined, the name of the Jdraw object (which is the name of the Tango numeric attribute) is passed as the argument to the constructor of the panel class.
- ✓ If the **“className”** extension is not defined nothing happens.
**Tango Boolean Attribute**

A Tango boolean attribute can be associated to a Dyno (Dynamic Object) or to an adapted Atk Viewer (for example BooleanScalarCheckboxViewer).

**Associated to a Dyno (Dynamic Object)**

In order to define your own behavior with a Dyno associated to a tango boolean attribute, you should define the mapping between the two values of the boolean attribute (true and false) and the caracteristique affected by the value. See the [Dynamic Objects in Jdraw](http://www.jdraw.com) (Flash Demo).

It’s important to know that the Dyno will receive at run time the numeric value for the boolean attribute. It means that if the attribute value is false, the value 0 is sent to the Dyno and if the attribute value is true the value 1 is sent to the Dyno.

**Associated to an Atk Viewer (for example BooleanScalarCheckboxViewer)**

The tango attribute will be set as the model of the AtkViewer (BooleanScalarCheckboxViewer). All the run-time behavior is defined by the AtkViewer which is used.

Some of the bean properties of the Atk Viewer are available in the extension Tab of the Jdraw properties window.

**User interaction**

When the **mouse enters** the graphic component associated to the tango numeric attribute, the name of the tango attribute is displayed inside a **tooltip**.

When an object associated to the tango attribute is selected by the user at run time, ATK tries to popup a panel:

- If the **className** extension is defined, the class is instantiated using a constructor with a String parameter.
- If the **classParam** extension is defined, the string is passed as the argument to the constructor of the panel class.
- If the **classParam** extension is not defined, the name of the Jd raw object (which is the name of the Tango boolean attribute) is passed as the argument to the constructor of the panel class.
- If the **className** extension is not defined nothing happens
**Tango DevState Spectrum Attribute**

An element of a Tango DevState spectrum attribute can be associated to any jdraw graphic object. From a simple drawing to a complex shape made of successive groups. An element of a Tango DevState spectrum attribute can also be associated to a Dyno. To assign an element of a DevState spectrum attribute we use the brackets. So to associate the 10th element of the state spectrum attribute sr/rf-tra/tra1/SubDevicesStates, the name of the graphic component should be `sr/rf-tra/tra1/SubDevicesStates[9]`.

Associated to a Jdraw Object (not a Dyno)

ATK will color the object according to the value of the element specified in the state spectrum attribute. The state/color mapping is the same as the one used in all other parts / viewers of ATK.

- ✓ If the object is filled: the fill color is changed according to the state value
- ✓ If the object is not filled: the line color is changed according to the state value
- ✓ If the object is made of successive groups, the change is made recursively in each group until the basic graphic objects are reached. In this hierarchy of objects, the graphic objects whose name is “IgnoreRepaint” do not change their color at all.

Associated to a Dyno (Dynamic Object)

As described in the previous sections a Dynamic Object (Dyno) has a specific dynamic behavior which has been defined during the drawing phase. In order to define your own behavior with a Dyno associated to an element of a State spectrum attribute, you should define the mapping between each different tango state numeric values and the caracteristique affected by the value.

It’s important to know that the Dyno will receive at run time a numeric value associated to the state attribute value. You can find the mapping between the numeric values and the tango state values in the Tango documentation: [http://www.esrf.eu/computing/cs/tango/tango_doc/kernel_doc/tango_java_api/classes/constant-values.html](http://www.esrf.eu/computing/cs/tango/tango_doc/kernel_doc/tango_java_api/classes/constant-values.html)

User interaction

When the **mouse enters** the graphic component associated to the state spectrum attribute, the name of the state spectrum attribute + index of the element in the spectrum is displayed inside a **tooltip**.

When an object associated to a state attribute is clicked by the user at run time, ATK tries to popup a panel.

- ✓ If the “**className**” extension is defined, the class is instantiated using a constructor with a String parameter.
- ✓ If the “**classParam**” extension is defined, the string is passed as the argument to the constructor of the panel class.
- ✓ If the “**classParam**” extension is not defined, the name of the Jdraw object (which is the name of the element of a tango DevState spectrum attribute) is passed as the argument to the constructor of the panel class.
- ✓ If the “**className**” extension is not defined nothing happens
**Tango Command**

A Tango Command can be associated to a Jdraw interactive component or to an adapted Atk Viewer (for example VoidVoidCommandViewer).

**Associated to a Jdraw interactive component**

When the interactive graphic component is clicked, the tango command is executed.

**Associated to an Atk Viewer (for example VoidVoidCommandViewer)**

The tango attribute will be set as the model of the AtkViewer (VoidVoidCommandViewer). All the run-time behavior is defined by the AtkViewer which is used. Some of the bean properties of the Atk Viewer are available in the extension Tab of the Jdraw properties window.

**User interaction**

When the **mouse enters** the graphic component associated to the tango command, the name of the tango command is displayed inside a **tooltip**.

When the interactive object associated to the tango command is clicked by the user at run time, ATK sends the command to the associated Tango device.

**Other types of Tango Attributes**

Other type of Tango attributes can be associated only to an Atk viewer available in Jdraw editor under the “Atk Swing” button. They cannot be associated to a Jdraw graphic component. The use of an Atk viewer is mandatory.

The following tango attributes can be used in Jdraw and associated to their corresponding Atk viewers as listed below:

- **String Scalar** attribute should be associated to a **SimpleScalarViewer**
- **Numeric Spectrum** attribute should be associated to a **NumberSpectrumViewer**
- **Numeric Image** attribute should be associated to a **NumberImageViewer**

The run time behavior is the one provided by the Atk viewer.
Advanced ATK programming

Section under construction ....
## Appendix 1: attribute viewers / setters

<table>
<thead>
<tr>
<th>Tango format and data type</th>
<th>View / Set</th>
<th>ATK class used as model</th>
<th>ATK viewer / setter</th>
<th>Tutorial section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar Any type</td>
<td>View and Set</td>
<td>AttributeList</td>
<td>ScalarListViewer ScalarListSetter NumberScalarListViewer</td>
<td>Use a generic scalar attribute viewer</td>
</tr>
<tr>
<td>Single attribute</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalar Numeric type</td>
<td>View</td>
<td>INumberScalar</td>
<td>SimpleScalarViewer</td>
<td>Using specific viewers ...</td>
</tr>
<tr>
<td>Single attribute</td>
<td></td>
<td></td>
<td>NumberScalarViewer</td>
<td>device status</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NumberScalarProgressBar</td>
<td></td>
</tr>
<tr>
<td>Scalar DevString single attribute</td>
<td>View</td>
<td>IStringScalar</td>
<td>SimpleScalarViewer</td>
<td>Using specific viewers ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>StatusViewer</td>
<td>device status</td>
</tr>
<tr>
<td>Scalar DevBoolean Single attribute</td>
<td>View</td>
<td>IBooleanScalar</td>
<td>SignalScalarLightViewer</td>
<td>Using specific viewers ...</td>
</tr>
<tr>
<td>Scalar DevShort, DevUshort Single attribute</td>
<td>View</td>
<td>IEnumScalar</td>
<td>SimpleEnumScalarViewer</td>
<td>Using specific viewers ...</td>
</tr>
<tr>
<td>Scalar DevState Device State single attribute</td>
<td>View</td>
<td>IDevStateScalar</td>
<td>StateViewer</td>
<td>device state</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Using specific viewers ...</td>
</tr>
<tr>
<td>Scalar DevState Single attribute</td>
<td>Set</td>
<td>INumberScalar</td>
<td>NumberScalarWheelEditor NumberScalarComboEditor</td>
<td>Using specific viewers ...</td>
</tr>
<tr>
<td>ScalarNumeric type</td>
<td>Set</td>
<td>IStringScalar</td>
<td>StringScalarEditor</td>
<td>Using specific viewers ...</td>
</tr>
<tr>
<td>Single attribute</td>
<td></td>
<td></td>
<td>StringScalarComboEditor</td>
<td></td>
</tr>
<tr>
<td>Tango format and data type</td>
<td>View / Set</td>
<td>ATK class used as model</td>
<td>ATK viewer / setter</td>
<td>Tutorial section</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------</td>
<td>-------------------------</td>
<td>---------------------</td>
<td>-----------------</td>
</tr>
</tbody>
</table>
| Scalar DevBoolean Single attribute | Set | IBooleanScalar | **BooleanScalarCheckBoxViewer**  
**BooleanScalarComboEditor**  
**SignalScalarButtonSetter** | Using specific viewers ... |
| Scalar DevShort, DevUshort Single attribute | Set | IEnumScalar | **EnumScalarComboEditor** | Using specific viewers ... |
| Scalar Any type Collection of attributes | View and Set | AttributeList | **ScalarListViewer**  
**ScalarListSetter**  
**NumberScalarListViewer** | **AttributeList viewers**  
**AttListViewer Flash Demo** |
| Scalar Numeric type Collection of attributes | View | AttributePolledList | **Trend** | **The trend of numberScalar**  
**Trend Flash demo** |
| Scalar DevBoolean Collection of attributes | View | AttributePolledList | **BooleanTrend** | **The trend of boolean scalar attributes** |
| Scalar Any type Collection of attributes | View and Set | IAttribute | **MultiScalarTableViewer** | **A set of scalar att...**  
**Scalar Table Flash demo** |
<p>| Scalar DevState Collection of attributes | View | IDevStateScalar | <strong>TabbedPaneDevStateScalarViewer</strong> | <strong>A set of DevStateScalar attributes</strong> |
| Spectrum Numeric type Single attribute | View | INumberSpectrum | <strong>NumberSpectrumViewer</strong> | <strong>NumberSpectrum attributes</strong> |
| Spectrum DevString Single attribute | View | IStringSpectrum | <strong>SimpleStringSpectrumViewer</strong> | <strong>StringSpectrum attributes</strong> |</p>
<table>
<thead>
<tr>
<th>Tango format and data type</th>
<th>View / Set</th>
<th>ATK class used as model</th>
<th>ATK viewer / setter</th>
<th>Tutorial section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spectrum Numeric type Collection of attributes</td>
<td>View</td>
<td>INumberSpectrum</td>
<td>MultiNumberSpectrumViewer</td>
<td>A set of NumberSpectrum attributes in a chart</td>
</tr>
<tr>
<td>Spectrum Numeric type Single attribute</td>
<td>View</td>
<td>INumberSpectrum</td>
<td>NumberSpectrumItemTrend</td>
<td>Trend of number spectrum attributes</td>
</tr>
<tr>
<td>Image Numeric type Single attribute</td>
<td>View</td>
<td>INumberImage</td>
<td>NumberImageViewer</td>
<td>NumberImage attributes</td>
</tr>
<tr>
<td>Image DevString Single attribute</td>
<td>View</td>
<td>IStringImage</td>
<td>StringImageTableViewer</td>
<td>StringImage attributes</td>
</tr>
</tbody>
</table>
## Appendix 2 : command viewers

<table>
<thead>
<tr>
<th>Input argument data type</th>
<th>Output argument data type</th>
<th>ATK class used as model</th>
<th>ATK Command Viewer</th>
<th>Tutorial section</th>
</tr>
</thead>
</table>
| DevVoid no input         | DevVoid no output         | ICommand                | VoidVoidCommandViewer  
                        ConfirmCommandViewer  | Commands with no input and no output |
| DevBoolean               | DevVoid no output         | ICommand                | OnOffCheckboxCommandViewer  
                        OnOffSwitchCommandViewer  | Commands with DevBoolean input and no output |
| DevString                | DevVoid no output         | ICommand                | OptionComboCommandViewer  | Commands with DevString input and no output |
| Any Type                 | Any Type                  | ICommand                | AnyCommandViewer      | Commands with any type of input and any type of output |
| Any Type                 | Any Type                  | CommandList             | CommandComboViewer.jpg | A collection of commands |
## Appendix 3: error viewers

<table>
<thead>
<tr>
<th>Type of ATK error</th>
<th>ATK method to use</th>
<th>ATK Error Viewer</th>
<th>Tutorial section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute read error during the attribute refreshing</td>
<td>addErrorListener</td>
<td>ErrorHistory</td>
<td>ErrorHistory</td>
</tr>
<tr>
<td>Attribute setting error during the attribute set value</td>
<td>addSetErrorListener</td>
<td>ErrorHistory</td>
<td>ErrorHistory</td>
</tr>
<tr>
<td>Command execution error</td>
<td>addErrorListener</td>
<td>ErrorHistory</td>
<td>ErrorHistory</td>
</tr>
</tbody>
</table>

- **ADD Error History**: ErrorHistory
- **ADD Error Popup**: ErrorPopup
- **ADD Error History**: ErrorHistory
- **ADD Error Popup**: ErrorPopup