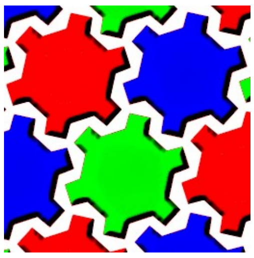
VisualAP 1.2[1]

A framework for component based design

Developer guide



[1]06 January 2010, author: Livio (javalc6@gmail.com)

CONTENTS

1 VisualAp

1.1 Introduction

VisualAp is a visual framework for building application and systems based on visual components. Users can add their own visual components in order to extend the capability of VisualAp.

VisualAp can be used in order to perform audio processing, image processing, text and other process-driven emulation.
VisualAp provides a visual framework based on lightweight components, called proclet.

The user can create an application by selecting the components from a toolbox, configuring the parameters (via the Javabeans framework), and connecting the components together in order to set-up communication channels between the components.

In the first release the user runs code execution through the engine included in VisualAp, in interpreted mode. In later releases it will be possible to generate Java code that can be compiled and run in standalone mode.

1.2 Basic Concepts

VisualAp is based on two main concepts: system and components.
A component is the basic element providing some services. A system is build using a number of components that are configured and connected together in order to achieve a complex functionality.
In VisualAp a system is a graph whose nodes are the components.
In VisualAp a component is also called proclet (processing element).

There are three type of components:

- source: any component that generates some sort of data
- sink: any component that consumes data
- processor: any component that process input data into output data

1.3 Execution mode

When a system is run, there are two possible mode of executing the processing:

- Single Shot
- Multi Shot (or Iterative)

Single Shot mode is used when there are no iterative sources.
In this case, when System->Run is selected, VisualAp will use the following algorithm:

1. foreach component invoke start() method
2. foreach component invoke component specific processing methods
3. foreach component invoke stop() method

Multi Shot mode is used when there are iterative sources.
In this case, when System->Run is selected, VisualAp will use the following algorithm:

1. foreach component invoke start() method
2. foreach component invoke component specific processing methods
3. foreach component invoke iterate() method
4. VisualAp restarts the procedure from step 1, until all iterative components return false to the iterate() method
5. foreach component invoke stop() method

2 Components

2.1 Introduction

A component (or proclet) is the building block of the VisualAp environment.
A component is shown with its icon plus a number of terminals.
Terminals are represented by a small blue square. Terminals are also called pins.
Input terminals are placed in the left side of a component,
while output terminals are placed in the right side of a component.

2.2 How to design components

It is possible to design and release new components to be used by VisualAp.
Component shall be implemented using javabeans pattern. A component will be based at least on the following files:

- main class, providing a number of methods in order to integrate in the VisualAp framework and provide some kind of service.
- support class extending SimpleBeanInfo, providing information about exposed properties, methods and icons
- manifest file
- a 16x16 icon (mandatory), may also include a 32x32 icon (optional)
- optional help file named <name of the bean>.html

Note that in the current implementation any component shall not contain any serialized class .ser

2.2.1 Details about the main class of the component

The main class must be designed in order to:

- implement *Serializable.* The *Cloneable* interface may be implemented, in order to allow copy&paste, for better performances.
- define the serialVersionUID field. When you create a new component, the value can be set by using the *serialver* tool from Sun.
- expose methods that performs processing according to the specification described in paragraph 3 “Specification of methods exposed by components”
- Ordinary fields in the class shall be defined using the modifiers “static final” or “transient”. Only the properties of the component are defined without such modifiers.

The main class may contain the <u>version</u> field, whose format is “n.m”, n is the major version number and m is the minor version number. If version field is not defined, VisualAp will assume the value 0.0 as version number for the component.
When you modify a component with a backward compatible change, just update the

version field. In case of backward incompatible[2] change you have to modify the name of the component.

The main class may include the following optional methods:
- static method getToolTipText(): provides a useful tooltip
- method dispose(): used to clear data at the end of component life cycle (e.g. when a new graph is loaded)
- method setContext(): used by the javabeans to get global variables. At least the blocksize variable is defined and usable by javabeans. The variable blocksize [3] shall be used only by sampled sources. Other type of javabeans shall ignore it.

- source components that support iteration shall implement iterate(). Methods start() and stop() may be used to prepare and terminate related activities, e.g. open and close a file.
- method start() is called to start the iteration process
- method iterate() is called at the end of each iteration. When all source javabeans that supports iterate() return false, current iteration is completed and then stop() is sent to all javabeans.

Note that even after the component has returned false to the iterate() method, the iterations could continue due to other sources producing data.
For this reason, in case the component has returned false to the iterate() method, it is still possible to receive further invocations of the iterate() method, the component shall ignore such invocations and it shall return null in subsequent calls to its processing methods.

- method stop() is called to end the iteration process.

In general any component shall not modify its input data, because the same input data could be used by other components!

In order to learn how to develop a new component, please look at the demo components.
===
Put your the proclets you develop in a package named according to your username, in order to avoid collision with other proclets with the same naming.
===

[2]Example of incompatible changes: removing a pin, deleting a parameter, changing the type of a parameter,...

[3]The variable *blocksize* is used only by source beans. In case of sampled data, the *blocksize* indicates how many samples/frames shall be emitted by the source for each execution shot. Blocksize is useful mainly in multi-shot system. This variable could be used also by source beans that produces strings. In this case the *blocksize* indicates the number of strings that shall be emitted by the source.

2.3 Deploy new components

New components shall be placed in the *beans* directory under VisualAp installation directory.

After a component has been tested, the developer can publish the component on the VisualAp website.

2.4 Demo components

The following components are provided with VisualAp 1.2:

- Viewer: shows the incoming data in a floating window
- ReadFile: read a file, contains a property "file" -> a custom editor is used for file property
- WriteFile: write a file, supported type: text, audio, image
- Mux: generate stereo audio from two mono audio inputs
- DeMux: split stereo audio in two mono audio
- Inspect: shows the type of the incoming data
- Speaker: plays an audio stream
- ToneGenerator: generate a simple audio tone, contains properties that are checked against max values, contains a property "type" -> a custom editor is used to select a specific value
- Microphone: records an audio file

The directory *test* contains the source files of the demo components. The buildtest.bat batch file can be run in the installation directory to build the test beans.

Other included components:

- Delay: introduces a delay in audio stream
- Echo: implements echo effect
- Imagetransform: transforms an image

3 Specification of methods exposed by components

This is version 1.0 of proclet specification

The version 1.0 supports auto-discovery of javabeans capabilities.

Any component is build using at least two classes:

- *componentname*.java: the main class provides the core functionality of the component
- *componentname*BeanInfo.java: support class to provide javabeans support. If the component uses special datatypes, it has to provide some editors extending PropertyEditorSupport

The main class of a component can provide the following types of methods:

- getter and setter according to Javabeans pattern: used to set/get value of component's parameters
- supporting methods: clone(), start(), iterate(), stop(), dispose(), setContext(), getToolTipText(),...
- processing methods: used to perform useful processing. These methods shall be exposed as return value of the getMethodDescriptors() method in the supporting *BeanInfo class.

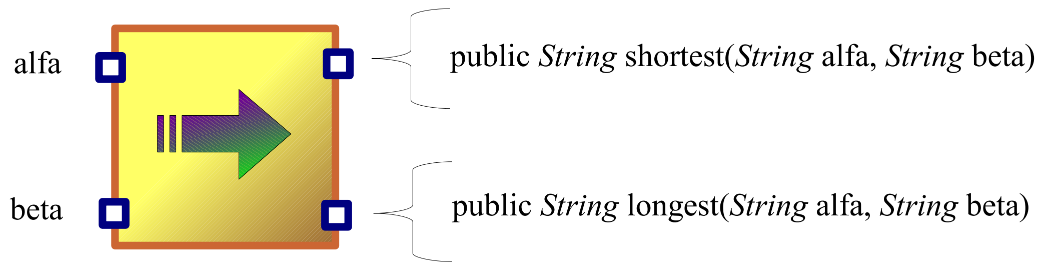
Processing methods belonging to a specific component shall have the same set of input parameters.
Each input parameter is in relation with an input pin of the component, first parameter is related to the upper pin, last parameter is related to lower pin.

Each processing method with a type (different from *void*) returns data that are in relation with the corresponding output pin of the component, first method is related to the upper pin, last method is related to lower pin.

In case of components without any output (i.e. sink) only one method will be defined:
```
public void method-name(typeA argA, typeB argB,...)
```

In case of components with n outputs, it is needed to define n methods:
```
public type1 method1-name(typeA argA, typeB argB,...)
.
public type1 methodn-name(typeA argA, typeB argB,...)
```

Example Component with two arguments and two outputs



public *String* shortest(*String* alfa, *String* beta)

public *String* longest(*String* alfa, *String* beta)

In order to simplify interoperability of visual components, it is recommended to use the following classes as type for input/output parameters:

- BufferedImage, to hold an image
- String and/or String[], to hold a simple text
- SampledAudio, to hold sampled audio

4 Java files

>package visualap

VisualAp.java	main class
GPanel.java	main panel
DialogPref.java	dialog handling preferences
ToolBox.java	toolbox with available javabeans
ErrorPrinter.java	utility to perform error logging
ClassPathHacker.java	workaround for ClassNotFoundException issue
BeanDelegate.java	delegate for JavaBeans
BeanException.java	
Check.java	check the validity of a system
CheckException.java	
Engine.java	performs system emulation
Header.java	header of visualap data files
HelpWindow.java	help file browser
LoadBeans.java	loads all the available JavaBeans
Vertex.java	utility class
VersionException.java	
WebFetch.java	retrieves files from the internet

>package graph

Node.java	abstract node
NodeText.java	node with text
NodeBean.java	node with bean
GList.java	list of nodes and utilities
Pin.java	pin
Edge.java	used to connect two pins
Edges.java	list of edges
Selection.java	list of selected elements

>package property

Property*.java	property editors

>package parser

Parser.java|	main class
Option.java|	
ParserException.java|	

>package common (used by beans)
SampledAudio.java sampled audio

5 Troubleshooting

5.1 Explanation of errors/warnings (tbd)

Errors detected at start-up
<beanname> caused BeanException in <jarfile>
<dirname> is not a directory

Errors detected loading a system (.vas file), in addition to generic file system errors:
Invalid file format
Invalid version found
Not valid SerialUID
Connection cannot be setup

Errors detected by system check:
Detected collision between two or more output pins
Detected a floating input pin
Detected input pins not connected to any output pin
Detected a loop
Detected a cycle

Error detected by system run, in addition to the same errors detected by system check:
IllegalAccessException
IllegalArgumentException: argument type mismatch
Other exceptions

5.2 Error log file

In case of some unexpected exceptions, VisualAp put some information in the file error.log that can be used for troubleshooting.
In order to keep the file small, only a sub-set of exceptions are reported in error.log.
Exceptions generated by the proclets are not reported in the log file.

6 VisualAp file specification (vas files)

The files containing a VisualAp system are coded using the XMLEncoder class available in Java. The developer shall not perform any specific activity on the content of the file.

The template for a vas file is described here:

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.x.y" class="java.beans.XMLDecoder">
 <object class="graph.Header">
  <void method="put">
   <string>application</string>
   <string>VisualAp</string>
  </void>
  <void method="put">
   <string>comment</string>
   <string>Created on dayofweek month day time CET year</string>
  </void>
  <void method="put">
   <string>version</string>
   <string>1.0</string>
  </void>
 </object>
 <object class="java.util.HashMap">

            globalVars

 </object>
 <object class="graph.GList">

            List of nodes

 </object>
 <array class="java.lang.Object" length="m">

            List of edges

 </array>
</java>
```

7 Terminology

Term	Meaning
System	A set of components that are interconnected and interacting. A system is represented as a graph, without cycles/loops, whose nodes are the components of the system.
Component	The basic building block of a system, performing some kind of simple processing.
Sink	A component that consumes data.
Source	A component that produces data.
Proclet	A Java component performing data processing in the VisualAp framework.