Oracle® AutoVue

AutoVue API Developer’s Guide

Release 21.0.0

November 2015

ORACLE®

AutoVue API Developer's Guide, Release 21.0.0

Copyright © 1999-2015, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Portions of this software Copyright 1996-2007 Glyph & Cog, LLC.

Contents

Preface v

1 Introduction

2 AutoVue API Packages

2.1 VueBean Package 2-1
2.1.1 Event Package 2-2
2.1.1.1 VueEvent 2-3
2.1.1.2 VueModelEvent 2-3
2.1.1.3 VueEventBroadcaster 2-3
2.1.1.4 VueFileListener 2-4
2.1.1.5 VueMarkupListener 2-4
2.1.1.6 VueViewListener 2-4
2.1.1.7 VueStateListener 2-4
2.1.1.8 VueModelListener 2-4
2.1.2 MarkupBean Package 2-4
2.1.2.1 Markup 2-5
2.1.2.2 MarkupLayer 2-5
2.1.2.3 MarkupEntity 2-5
2.2 Server Control 2-6
2.3 VueAction Package 2-7
2.3.1 AbstractVueAction 2-7
2.3.2 VueAction 2-7
2.3.2.1 Create an action that performs a single function 2-7
2.3.2.2 Create an action that performs multiple functions 2-8

3 Sample Cases

3.1 Building an AutoVue API Application 3-1
3.2 Custom VueAction 3-4
3.2.1 Action that Performs a Single Function 3-5
3.2.2 Action that Performs Multiple Functions 3-6
3.3 Directly Invoking VueActions 3-9
3.4 Markups 3-9
3.4.1 Entering Markup Mode 3-9
3.4.2 Checking Whether Markup Mode is Enabled 3-10

3.4.3 Exiting Markup Mode 3-10
3.4.4 Adding an Entity to an Active Markup/Layer 3-10
3.4.5 Enumerating Entities 3-10
3.4.6 Getting Entity Specification of a Given Entity 3-10
3.4.7 Changing Specification of an Existing Entity Programmatically 3-10
3.4.8 Adding a Text Box Entity 3-11
3.4.9 Open Existing Markup 3-11
3.4.10 Saving Markups to a DMS/PLM 3-12
3.4.11 Adding a Markup Listener to Your Application 3-13
3.5 Converting Files 3-13
3.5.1 Making a Call to a Convert Method 3-13
3.5.2 Converting to JPEG (Custom Conversion) 3-14
3.5.3 Converting to PDF 3-14
3.6 Printing a File to 11x17 Paper 3-15
3.7 Monitoring Event Notifications 3-15
3.8 Retrieving the Dimension and Units of a File 3-16

4 FAQ

4.1 MarkupBean 4-1
4.2 Printing 4-2
4.3 Upgrading 4-2
4.4 General 4-2
A.1 General AutoVue Information A-1
A.2 Oracle Customer Support A-1
A.3 My Oracle Support AutoVue Community A-1
A.4 Sales Inquiries A-1

Preface

The *AutoVue API Developer's Guide* provides detailed technical information on the AutoVue API concepts introduced in *Oracle AutoVue Integration Guide*. This includes information on the AutoVue API, its fundamental packages and classes, as well as sample code for building your own integration. For the most up-to-date version of this document, go to the AutoVue Documentation Web site on the Oracle Technology Network at
`http://www.oracle.com/technetwork/documentation/autovue-091442.html`.

Audience

This document is intended for Oracle partners and third-party developers (such as integrators) who want to implement their own integration with AutoVue. Note that these developers are expected to have a good understanding of JAVA programming. This guide serves as a good starting point for developers and professional services to become more familiar with the AutoVue API.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program Web site at
`http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see the following documents:

- *Oracle AutoVue Integration Guide*
- *Augmented Business Visualization Developer's Guide*
- *VueBean Javadocs*
- *Oracle AutoVue Installation and Configuration Guide*
- *Oracle AutoVue Planning Guide*
- *Oracle AutoVue Integration SDK Overview*

- *Oracle AutoVue Web Services Overview*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Introduction

The AutoVue Application Programming Interface (API) is a Java-based toolset that provides tools to modify the functionality of Oracle's AutoVue client, and allows you to create your own customized Java applets/applications based on AutoVue API components.

This document presents the technical application of the AutoVue API and its packages and classes. Additionally, basic and advanced applications of the AutoVue API are provided along with their source code.

Note: For a more general introduction to the AutoVue API, refer to the "AutoVue API Solution" section of the *Oracle AutoVue Integration Guide*. For detailed information on the packages and classes included in the AutoVue API, refer to the *VueBean Javadocs*.

---

2 AutoVue API Packages

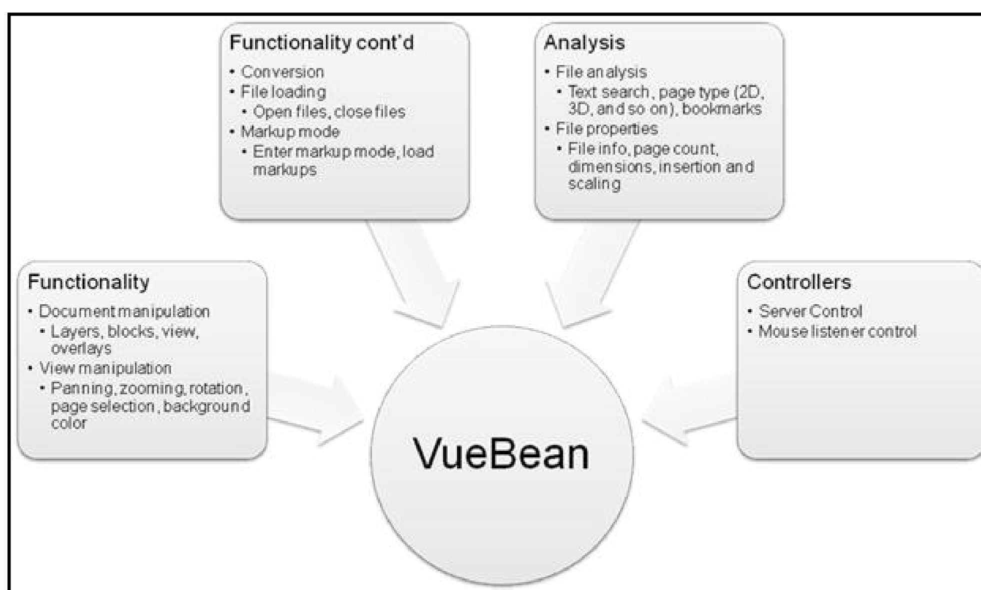
The chapter provides an overview of common classes and interfaces that are used to create a solution based on the AutoVue API.

Note: For more information on classes/packages, refer to the *VueBean Javadocs* located in the <AutoVue Installation>\docs directory.

2.1 VueBean Package

The VueBean component is central to the AutoVue client architecture. An application can embed the VueBean component and use its API to provide comprehensive support for file viewing, markup, real-time collaboration, and so on. The following diagram provides a graphical overview of how the VueBean can be used when developing your own application/applet.

Figure 2–1 VueBean overview



Note: It is possible to have multiple instances of the VueBean class. For example, when AutoVue is in Compare mode there are three instances of the VueBean class.

A typical VueBean usage scenario is as follows:

1. Create a VueBean Object.
2. Create a server control or use the default one obtained from the VueBean.
3. Use the server control to connect to the server and open a session on it.
4. View a file by invoking the VueBean.setFile(DocID) method.

The following file types are supported by the VueBean:

- Vector files (2D and 3D)
- Raster files
- Document files (MS Word, and so on)
- Spreadsheet files
- Archive files

The file type can be queried through the VueBean.getFileType() method and file information can be retrieved through the VueBean.getFileInfo() method.

You may have to convert a file to another file type. To do so, use the VueBean.convert() method.

In its various modes, such as viewing and markup, the VueBean manages the representation of a file including the management of overlays, layers, and external references to other files or resources upon which the file depends. Use the VueBean.getResourceInfoState() method to query for resources that are attached to a file.

To search for a particular string in the file use the VueBean.search(PAN_CtlSearchInfo) method. The following is an example of how to build the PAN_CtlSearchInfo object.

```
// Construct the search object with arguments (Search String, Search Multiple
// Occurrences, Search Downwards, Wrapped Search, Match Case, Whole Word),
// in this example we search for the word "line".
PAN_CtlSearchInfo searchInfo = new PAN_CtlSearchInfo("line", true, true,
                                                     true, false, true);
```

> **Note:** Since the VueBean is only a client-side component, the connection to the AutoVue server must be established before any operation can be performed on the VueBean. Refer to Section 2.2, "Server Control" for more information.

2.1.1 Event Package

`com.cimmetry.vuebean.event`

For VueBean-specific events, the event delegation model of the VueBean is slightly different from the standard Java one. Listeners such as VueViewListener, VueFileListener, VueMarkupListener, or VueStateListener should register to the VueBean's VueEventBroadcaster object instead of to the VueBean itself.

```
For example: vueBean.getVueEventBroadcaster().addFileListener(listener).
```

This package provides interfaces and classes for VueBean event broadcasting. Every VueBean object has an event broadcaster. Depending on the operation type, the broadcaster notifies listeners using an instance of VueEvent or VueModelEvent. The following types of events are supported:

- File events
- View events
- Markup events
- State events
- Model events

Every event type has a corresponding event listener interface which is registered to the broadcaster. Objects that are responsible for handling of events should implement one or more of the listener interfaces.

The following code sample defines and registers an event handler:

```
import com.cimmetry.vuebean.*;
import com.cimmetry.vuebean.event.*;
.
.
.
final VueBean vueBean = getVueBean();// Get the valid active VueBean
if (vueBean != null) {
  VueFileListener eventHandler = new VueFileListener() {
    public void onFileEvent(VueEvent ev) {
      switch (ev.getType()) {
      case VueEvent.ONSETFILE:
          System.out.println("Set file: " + vueBean.getFile());
          break;
      case VueEvent.ONSETPAGE:
          System.out.println("Set page: " + vueBean.getPage());
          break;
      }
    }
  };
  vueBean.getVueEventBroadcaster().addFileListener(eventHandler);
}
.
.
.
```

2.1.1.1 VueEvent

`com.cimmetry.vuebean.event.VueEvent`

VueEvent object encapsulates information for all notifications sent by VueBean and is generated for the VueFileListener, VueViewListener, VueMarkupListener and VueStateListener interfaces. The event type is used to differentiate between a view event, file event, markup event or state event.

2.1.1.2 VueModelEvent

`com.cimmetry.vuebean.event.VueModelEvent`

The VueModelEvent class handles all notifications for model-related events such as entity attributes, 3D transformation, and so on. It is generated for objects implementing VueModelListener interface.

2.1.1.3 VueEventBroadcaster

`com.cimmetry.vuebean.event.VueEventBroadcaster`

VueEventBroadcaster is used to manage the event delegation model for the VueBean. Each listener has to register to a VueEventBroadcaster to be notified of events in the VueBean. By design, each VueBean owns its own VueEventBroadcaster. However, you may find it useful to use only one VueEventBroadcaster for all beans by using the VueBean.setVueEventBroadcaster method.

2.1.1.4 VueFileListener

`com.cimmetry.vuebean.event.VueFileListener`

Objects implementing this interface listen for file event notifications (such as setting file, setting page, and so on).

2.1.1.5 VueMarkupListener

`com.cimmetry.vuebean.event.VueMarkupListener`

Objects implementing this interface listen for markup event notifications (such as entering or exiting markup mode).

2.1.1.6 VueViewListener

`com.cimmetry.vuebean.event.VueViewListener`

Objects implementing this interface listen for view event notifications (such as zoom, begin and end paint, and so on).

2.1.1.7 VueStateListener

`com.cimmetry.vuebean.event.VueStateListener`

Objects implementing this interface listen for state event notifications (such as server error, file error, and so on).

2.1.1.8 VueModelListener

`com.cimmetry.vuebean.event.VueModelListener`

Objects implementing this interface listen for model event notifications (such as model attribute, selection, transformation changes, and so on).

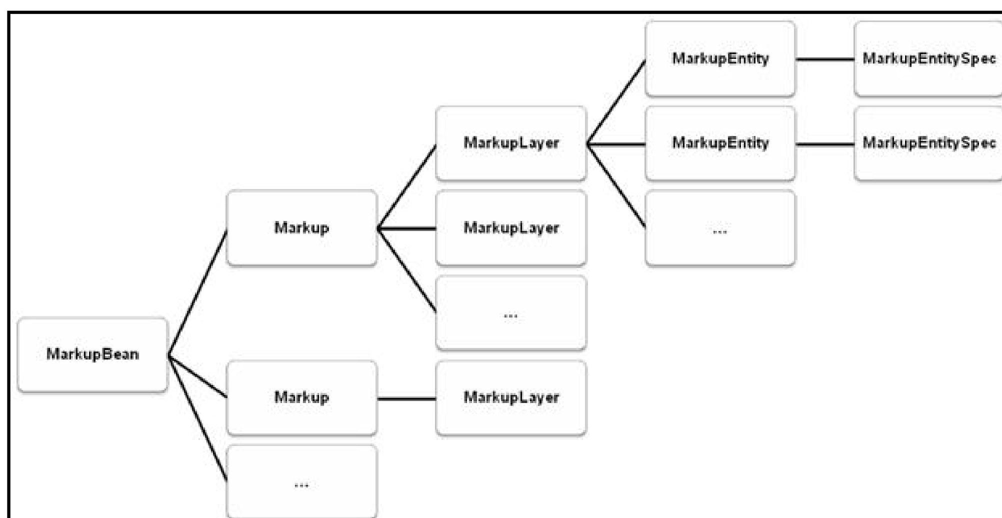
2.1.2 MarkupBean Package

`com.cimmetry.markupbean`

The top-level class for the com.cimmetry.markupbean package is the MarkupBean class. MarkupBean represents the Markup functionality in the VueBean API. Each VueBean instance can contain only one MarkupBean instance, represented by a private member variable. Through the MarkupBean class, you can add/modify/remove Markup Files, Markup Layers, and Markup Entities, as well as open and save Markup Files.

The following diagram displays how the architecture of a Markup is structured into four separate levels: Section 2.1.2.1, "Markup," Section 2.1.2.2, "MarkupLayer," Section 2.1.2.3, "MarkupEntity," and Section 2.1.2.3.1, "MarkupEntitySpec."

Figure 2–2 Markup architecture



2.1.2.1 Markup

`com.cimmetry.markupbean.Markup`

This interface represents an individual Markup file. The key functionalities are as follows:

- Get/set information regarding the Markup files, such as:
  - Name
  - Visibility
  - Whether Markup is modified
  - Whether Markup is read-only
- Get information regarding the base file
- Get the layers in the Markup

2.1.2.2 MarkupLayer

`com.cimmetry.markupbean.MarkupLayer`

This interface represents an individual Markup layer. The key functionalities are as follows:

- Get/set information regarding the specific layer, such as:
  - Name
  - Color
  - Visibility
  - Default line type and width
- Get the entities in the Markup layer

2.1.2.3 MarkupEntity

`com.cimmetry.markupbean.MarkupEntity`

This interface represents an individual Markup entity. The key functionalities are as follows:

- Get/set information regarding the specific Markup entity, such as:
  - Name
  - Author
  - Date modified
  - Color
  - Line type and width
  - Tooltip text
  - Visibility
  - Selection state
- Get children entities of the specific entity
- Perform actions when user double-clicks on entity

2.1.2.3.1 MarkupEntitySpec

```
com.cimmetry.markupbean.MarkupEntitySpec
```

This class represents an entity's specification. Each entity has its own specification class that is derived from this class that defines the attributes specific to that entity's context.

For example, the specification for a rectangle entity includes attributes for the XY coordinates of all four corners, while the specification for a text entity includes attributes for the contained text as well as its alignment.

2.2 Server Control

```
com.cimmetry.vueconnection.ServerControl
```

The ServerControl class handles the server connection object and the user session. Prior to using the VueBean, you must first set its ServerControl properties, connect to the server via the connect() method, and then open a session via the sessionOpen() method.

For example:

```
import com.cimmetry.vuebean.*;
import com.cimmetry.vueconnection.ServerControl;
…
VueBean bean = new VueBean();
ServerControl control = bean.getServerControl();
try {
    control.setHost(<SERVER URL>);
    control.connect();
    control.setUser("scarlati");
    control.sessionOpen();
} catch (Exception e) {
    System.out.println("Failed to connect to AutoVue Server.");
}
…
```

Note: Set the server URL to the VueServlet URL.

For example, http://<HostName>:5098/servlet/VueServlet

2.3 VueAction Package

`com.cimmetry.vueaction`

This package provides a hierarchy of classes implementing the AutoVue action API. It can be used to add graphical user interface (GUI) elements to different contexts (such as menu bar, toolbar, status bar, and so on). For example, when a menu option is selected in the GUI, a VueAction is triggered.

To add a new action to the AutoVue client, create a new action class by extending VueAction.

Use the methods in this package to:

- Specify resources for an action. For example, menu item text, an icon, tooltip text, and so on.
- Specify which resource bundle (a properties file with resource mappings) to search in for the action's resources.
- Specify sub-actions (for example, Zoom In, Zoom Out, Zoom Previous, and so on) for the action if it can perform more than one function.
- Receive a message signifying that the action should be performed. If the action has sub-actions, the sub-action to perform is specified.
- Specify the display properties of the action or its sub-actions that appear in the GUI in the menu bars, toolbars, and popup menus. For example, specify whether the action can be selected (behaves as a checkbox) and/or whether it is enabled.
- Specify groups of sub-actions (if the action includes sub-actions) in which selection is exclusive (that is, in which only one sub-action can be selected at a time).

2.3.1 AbstractVueAction

`com.cimmetry.vueaction.AbstractVueAction`

The abstract class AbstractVueAction is the super class of all action classes. All actions performed on the session must be derived from this class or a descendent of this class.

2.3.2 VueAction

`com.cimmetry.vueaction.VueAction`

VueAction is an abstract class that extends VueActionMultiMenu. It provides a simple yet powerful interface for creating actions.

To create a new action class, you must extend this class. There are two ways to do this depending on whether your action performs a single function or multiple functions. The following sections describe both scenarios.

2.3.2.1 Create an action that performs a single function

1. Make sure your class extends VueAction.

2. In the constructor of your class, call the appropriate super constructor.

> **Note:** Since your action performs only one function, the super constructor takes the two String arguments: resource key and resource bundle. The resource bundle identifies the set of text files (one for each locale your action supports) containing the resources identified by the resource key for your action.

3. Implement a perform() method to override the one in VueAction.

> **Note:** This method is called when your action has been fired. In this method, enter your action's code.

4. Implement event handlers onFileEvent and onViewEvent to ensure that your action is enabled or disabled when appropriate. For example, if no base file has been loaded yet, your action will be disabled. However, once a file has been reloaded, your action must be enabled.
5. Create one or more resource files (one resource file per language your action supports) containing the resource keys and their values needed by your action. Together with any icon files used by your action, these files are referred to as a *resource bundle*. For an example of a resource file, refer to VueFrame_en.properties.
6. Create a copy of AutoVue's .gui file and insert the name of your new action in the appropriate location.

To view an example of implementing an action that performs a single function, refer to Section 3.2.1, "Action that Performs a Single Function."

2.3.2.2 Create an action that performs multiple functions

1. Make sure your class extends VueAction.
2. In the constructor of your class, call the appropriate super constructor.

> **Note:** Since your action performs multiple functions, the super constructor takes one String argument: the resource bundle name. The resource bundle name identifies the set of text files (one for each language your action supports) containing the resources for your action.

3. After you call the super constructor, call defineSubAction() to define each sub-action your action can perform.

> **Note:** In each case, specify the name by which you want to refer to the sub-action and its resource key. The resource key identifies where to find the resources for your action (for example, menu item text, icon, tooltip text and so on) in your resource bundle. Optionally, you can call defineExclusiveGroup() to define a subset of your sub-actions that form an exclusive group. That is, sub-actions that are selectable where only one can be selected at a time.

4. Implement a performSubAction(String) method to override the one in VueAction.

Note: This method is called when your action's sub-action has been fired. The method is passed the name of the sub-action fired, so that you will know which one to perform. In this method, enter your sub-action's code.

5. Implement event handlers onFileEvent and onViewEvent to ensure that your sub-actions are enabled or disabled when appropriate. For example, if no base file has been loaded, your sub-action will be disabled. However, once a file has been reloaded, your sub-actions must be enabled.
6. Create one or more resource files (one resource file per language that your action supports) containing the keys and values needed by your action.

Note: Together with any icon files used by your action, these files are referred to as a *resource bundle*.

7. Create a copy of AutoVue's .gui file and insert the name of your new action in the appropriate location. You must also specify the appropriate sub-actions.

To view an example of implementing an action that performs multiple functions, refer to Section 3.2.2, "Action that Performs Multiple Functions."

Note: When deploying VueAction jar on the web, you have to properly sign the jar. Refer to how to Configure and Run the AutoVue VueActionSample (Doc ID 1677471.1)

3 Sample Cases

This chapter provides information on typical use cases you may come upon when creating an AutoVue API applet/application or adding enhanced functionality to the AutoVue client. Refer to the *VueBean JavaDocs* for more information.

Important: When executing a task in sequence you must make sure the previous task is completed before starting a new one. For example, when opening a file, the process should listen for the file event VueEvent.ONPAGELOADED to be notified. In the event of a file error, the state even VueEvent.ONFILEERROR is notified. When loading markups, listen and wait for the markup event MarkupEvent.ONMARKUPLOADED to be notified.

Note: Throughout this document, m_vueBean is used as a valid active VueBean object and m_JVue as a valid AutoVue applet object. This is done assuming that the methods or segments of code that use objects have access to a class which owns them.

3.1 Building an AutoVue API Application

A good starting point with the AutoVue API is to create an application that opens and displays a file.

This section provides detailed steps for creating a file open application using the AutoVue API.

1. Import required packages.

```
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;
import com.cimmetry.util.Messages;

import com.cimmetry.core.*;
import com.cimmetry.vuebean.*;
import com.cimmetry.vueconnection.ServerControl;
```

2. Create a Java class, ApplicationSample, that can be run as a stand-alone application, and declare all external parameters and internal data members.

```
public class ApplicationSample {
  private String m_host = "http://<HostName>:5098/servlet/VueServlet";
```

```
  private String m_user = "guest";
  private String m_fileName = null;
  private String m_verbose = null;
  private String m_format = "AUTO";
  // Internal data members
  private VueBean m_vueBean = null;
  private ServerControl m_control = null;
  private static JFrame m_frame = null;
  private JMenu m_fileMenu = null;
}
```

3. Create stand-alone application support.

```
public static void main(final String args[]) {
  ApplicationSample app = new ApplicationSample();
  app.init(args);
}
public void init(final String[] args) {
  switch (args.length) {
    case 4:
      m_verbose = args[3];
    case 3:
      m_fileName = args[2];
    case 2:
      m_user = args[1];
    case 1:
      m_host = args[0];
    default:
      break;
  }
  init();
}
```

4. Initialize the application.

```
public void init() {
  // Setup verbosity
  if (m_verbose != null && m_verbose.length() > 0) {
    Messages.setVerbosity(m_verbose);
  }
…
```

> **Note:** The init() method continues until step 13.

5. Establish a connection with the server.

```
m_control = new ServerControl();
try {
  m_control.setHost(m_host);
  m_control.connect();
} catch (Exception e) {
  System.out.println("Unable to connect to:"+m_host);
  e.printStackTrace();
  return;
}
```

6. Open the session.

```
try {
  m_control.setUser(m_user);
```

```
  m_control.sessionOpen();
} catch (Exception e) {
  System.out.println("Unable to open session for " + m_user);
  e.printStackTrace();
  return;
}
```

7. Initialize the frame.

```
m_frame = new JFrame("VueBean Sample");
m_frame.setBounds(100, 100, 640, 480);
m_frame.addWindowListener( new WindowAdapter() {
  public void windowClosing( WindowEvent e) {
    destroy();
  }
});
```

8. Set the menus and actions.

```
setMenuBar();
```

9. Create the bean.

```
m_vueBean = new VueBean(m_format);
m_vueBean.setServerControl(m_control);
m_vueBean.setBackground(Color.lightGray);
```

10. Add the VueBean to the frame.

```
m_frame.getContentPane().add(m_vueBean);
```

11. Display the frame.

```
m_frame.setVisible(true);
```

12. Show the file.

```
updateFile();
}// Closing bracket for init() method
```

Note: This step marks the end of the init() method.

---

13. Close the session.

```
public void destroy() {
  try {
      m_control.sessionClose();
  } catch (Exception ex) {
      ex.printStackTrace();
  }
  m_frame.setVisible(false);
  m_frame.dispose();
  System.exit(0);
}
```

14. Get the attached VueBean.

```
public VueBean getVueBean() {
  return m_vueBean;
}
```

15. Get the attached frame.

```
public JFrame getFrame() {
  return m_frame;
}
```

16. Get the file menu.

```
protected JMenu getFileMenu() {
  return m_fileMenu;
}
```

17. Get the frame. The following method sets the applet's menu bar to File Open, Print, and Exit.

```
public void setMenuBar() {
  m_fileMenu = new JMenu("File");
  JMenuItem menuItem;
  // File open menu item
  menuItem = m_fileMenu.add(new JMenuItem("Open"));
  menuItem.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    showFile();
  }
  });
  // set the Applet's menu bar
  JMenuBar menu_bar = new JMenuBar();
  m_frame.setJMenuBar(menu_bar);
  menu_bar.add(m_fileMenu);
}
```

18. Load the file.

```
public void updateFile() {
  // Set the vuebean's file
  if (m_fileName != null && !m_fileName.equals("")) {
    m_vueBean.setFile(new DocID(m_fileName));
    m_vueBean.setBackground(Color.lightGray);
  }
}
```

19. Display the client-side (upload) File Open dialog and set the selected file in the bean.

```
public void showFile() {
  FileDialog openDlg = new FileDialog(m_frame, "File Open", FileDialog.LOAD);
  openDlg.setVisible(true);
  m_fileName = "upload://"+openDlg.getDirectory() + openDlg.getFile();
  openDlg.dispose();
  updateFile();
}
}//end of class
```

Note: End of class ApplicationSample. In order to run the application properly, an AutoVue server needs to be running on either a local or remote host that is specified through command line arguments. Refer to step 3 for the definition of each argument.

3.2 Custom VueAction

This section presents examples implementing a custom VueAction class.

3.2.1 Action that Performs a Single Function

The following example shows how to implement a custom action for AutoVue that performs a single function. That is, when a user double-clicks on a hotspot, a dialog appears and lists all components of a drawing that are represented by the hotspot.

For information on AutoVue's hotspot capabilities, refer to the *Oracle Augmented Business Visualization Developer's Guide*.

Note: The following are segments of the source code of the VueAction example to illustrate the essential steps of creating a custom action, it may not compile if you just copy and paste the code here. For the complete source code, refer to PartListAction.java.

---

1. Import all required packages.

```
import java.awt.*;
import java.util.Vector;
import com.cimmetry.vuebean.*;
import java.awt.event.*;
import com.cimmetry.vuebean.event.*;
import com.cimmetry.vueframe.*;
import com.cimmetry.vueframe.hotspot.*;
import com.cimmetry.core.*;
import com.cimmetry.dialogs.VueBasicDialog;
import com.cimmetry.vueaction.VueAction;
import com.cimmetry.gui.*;
```

2. Make your class extend VueAction.

```
public class PartListAction extends VueAction { …}
```

3. In the constructor of your class, call the appropriate super constructor. Since this action only performs a single function, a call to the super-constructor of VueAction takes this action's resource key as well as its resource bundle name.

```
public PartListAction() {
      super("LIST_PARTS",RESOURCE_BUNDLE_NAME);
      setViewListener(true);
}
```

Note: The resource bundle name here is the common part of resource bundle files for different languages. The actual name of a resource bundle file should include the language suffix and file extension. For example, PartListAction_en.properties is the resource bundle file for English.

---

4. Implement a perform method for this action.

```
public void perform() {
      PartInfo[] parts = new PartInfo[m_cart.size()];
      m_cart.copyInto(parts);
      PartListDialog dialog = new PartListDialog(getFrame(), parts);
      dialog.show();
}
```

5. Implement the event handlers onFileEvent and onViewEvent to notify when a file has changed and to update the user-interface.

```
public void onFileEvent(VueEvent e) {
      switch (e.getEvent()) {
         case VueEvent.ONPAGELOADED:
         setEnabledByCurrentState();
         break;
      }
}
public void onViewEvent(VueEvent e) {
      switch(e.getEvent()) {
         case VueEvent.ONLINKCLICKED:
         Object[] params = (Object[]) e.getParameter();
          MouseEvent me = (MouseEvent) params[0];
          if (me.getClickCount() == 2) {
             Object link = params[1];
             if (link instanceof HotSpot) {
                  HotSpot hotspot = (HotSpot) link;
                  PartInfo part = getPartInfo(hotspot);
                  m_cart.addElement(part);
             }
         }
          break;
          default:
          super.onViewEvent(e);
          break;
     }
}
```

6. The dialog that lists all components of a drawing extends VueBasicDialog. You must implement your own constructor that calls the super-constructor and over-rides buildDialog() and buttonAction(int).

```
public static class PartListDialog
extends
      VueBasicDialog
implements
      ActionListener (…)
protected void buildDialog() {

  super.buildDialog();
…
}
protected void buttonAction(int index){…}
```

7. You must define a model for the table that represents the displayed product parts list.

```
public static class PartListModel implements CTableModel { …}
```

8. Close the PartListDialog() method.
9. Get a PartInfo associated with a given hotspot.

```
private PartInfo getPartInfo (HotSpot hotspot) {
            return new PartInfo(hotspot.getDefinitionKey(),
            hotspot.getHotSpotKey(),
            hotspot.getProperty(HotSpot.PROPERTY_DESCRIPTION));
)
```

3.2.2 Action that Performs Multiple Functions

The following example shows how to implement a custom action for AutoVue that performs multiple tasks. The custom action consists of several related sub-actions that

access information about parts of a model. One sub-action permits the user to order a part, another permits the user to display part information, and another sub-action displays a list of all the model's parts.

Note: The following are segments of the source code of the VueAction example to illustrate the essential steps of creating a custom action, it may not compile if you just copy and paste the code here. For the complete source code, refer to *PartCatalogueAction.java*.

---

1. Make your class extend VueAction.

```
public class PartCatalogueAction extends VueAction {
  private static final String RESOURCE_BUNDLE_NAME ="/PartCatalogueAction";

  // Names of the sub-actions used in *.gui file
  private static final String ORDER_SUBACTION = "Order";
  private static final String LIST_PARTS_SUBACTION = "ListParts";
  private static final String SHOW_INFO_SUBACTION = "ShowInfo";
  …
}
```

2. In the constructor of your class, call the appropriate super constructor.

```
public PartCatalogueAction() {
        super(RESOURCE_BUNDLE_NAME);
        defineSubAction(SHOW_INFO_SUBACTION,"SHOW_PART_INFO");
```

Note: The resource bundle name used here is the common part of resource bundle files for different languages. The actual name of a resource bundle file should include the language suffix and file extension. For example, PartCatalogueAction_en.properties is the resource bundle file for English.

---

3. Call defineSubAction to define each sub-action your action can perform.

```
defineSubAction(ORDER_SUBACTION,"ORDER_PART");
defineSubAction(LIST_PARTS_SUBACTION,"LIST_PARTS");
defineSubAction(SHOW_INFO_SUBACTION,"SHOW_PART_INFO");
}
```

4. Implement a performSubAction(String) method to override the one in VueAction.

```
public void performSubAction(String subActionName) {
        if (subActionName.equals(ORDER_SUBACTION)) {
          //Code for performing the "Order" subaction
            …
        } else if (subActionName.equals(LIST_PARTS_SUBACTION)) {
            //Code for performing the "List Parts" subaction
            …
          }
…
}
```

5. Implement the event handlers onFileEvent and onViewEvent to ensure that your sub-actions are enabled or disabled when appropriate.

```
public void onFileEvent(VueEvent e) {
  switch (e.getEvent()) {
        case VueEvent.ONSETFILE:
            //Code for handling ONSETFILE event
            …
```

```
            case VueEvent.ONPAGELOADED:
                //Code for handling ONPAGELOADED event
                setEnabledByCurrentState();
                …
                break;
            }
        }
    public void onViewEvent(VueEvent e) {
      switch(e.getEvent()) {
            case VueEvent.ONVIEWCHANGED:
                //Code for handling ONVIEWCHANGED event
                setEnabledByCurrentState();
                …
                break;
            case VueEvent.ONOPTIONSCHANGED:
                //Code for handling ONOPTIONSCHANGED event
                …
                break;
            }
        }
```

6. Create one or more resource files, one per language your action supports, containing the keys and values needed by your action. For example:

```
…
FILE_MARKUP_NEW_MARKUP=&New Markup, 32_new_markup_red.png, New Markup
FILE_MARKUP_OPEN=&Open..., 57_markup_red.png, Open Markup(s)
FILE_MARKUP_SMALL=     &Markup, 57_markup_red_small.png, Markup
FILE_MRU=Recent Files
FILE_NOTFOUND=File not found.
FILE_NOTSUPPORTED=This file format is not supported by your server.
FILE_NOTUPLOADED=Failed to upload file.
FILE_OPEN=&Open...\\tCTrL+O, 59_open.png, Open File
FILE_OPEN_SERVER=Open from &Server..., , Open a file from the server
…
```

Similarly, in our resource bundle file for English language PartCatalogueAction_en.properties, it should contain the resource keys for the PartCatalogueAction shown in the following:

```
…
ORDER_PART = &Order Part, order_part.png, Order a part
LIST_PARTS = &List Parts, list_parts.png, List product parts
SHOW_INFO_SUBACTION = &Show Part Info, show_info.png, Show part information
…
```

Note: Each resource key has three entries separated by a comma ",". The first entry (for example, &Order Part) is the text displayed on the GUI item (such as a menu item or toolbar button) and the ampersand "&" defines a shortcut key. The second entry (for example, order_part.png) is the file of the icon displayed on its GUI item. The third entry is the tooltip text for the GUI item. The second and third entries are optional. You should get the icon files ready if needed and add them to the JAR file for your custom action.

---

7. Make a copy of AutoVue's default.gui file located in the <AutoVue Installation Root>\bin directory, and insert the name of your new action in the appropriate locations of your GUI file. Note that for an action that performs multiple tasks, you must also specify the appropriate sub-actions.

Note: For information on how PartCatalogueAction sub-actions are inserted into a menu bar, tool bar, and custom pop-up menu, refer to default.gui and the custom.gui file located in the <AutoVue Installation Root>\examples\VueActionSample\ directory.

---

8. To allow the custom action to take effect, you may need to create a JAR file with your custom VueAction classes and all resource files they depend on. For example, for the resource bundle files for different languages and icon files, if any, place your JAR file under AutoVue's bin directory or its web root directory and include your JAR file in the classpath of the stand-alone AutoVue (JVue) application or ARCHIVE list of the AutoVue (JVue) applet.
9. You must specify the name of the modified GUI file through Applet or Command line parameters. For more information, refer to the "Customizing the GUI" section of the *Installation and Configuration Guide*.

3.3 Directly Invoking VueActions

You can develop your own customized user interface in an HTML page that incorporates AutoVue functionality. To do so, you must call invokeAction() or invokeSubAction() of the com.cimmetry.jvue.JVue applet from the HTML page. This call to the action can be done purely through JavaScript. For a list of actions/subactions, refer to the default.gui file located in <AutoVue Install Root>\bin directory.

Example 3–1 invokeAction()

```
invokeAction(VueActionFileOpen) //Displays the File Open dialog
```

3.4 Markups

The following sections describes some ways to execute common Markup actions.

Note: Various MarkupBean functionalities (and various functionalities throughout the AutoVue API) require the use of the *Property* class. This class is used to define various property hierarchies for other classes in the API.

3.4.1 Entering Markup Mode

```
VueBean.setMarkupModeEnabled(true)
```

Checks whether the MarkupBean member is null, and if so:

- Instantiates a new MarkupBean object
- Gets the markup settings from the user's profile
- Sets the markup-specific mouse listeners
- Points the VueBean's MarkupBean member to the new instance
- Broadcasts VueEvent.ONENTERMARKUPMODE

3.4.2 Checking Whether Markup Mode is Enabled

```
VueBean.isMarkupModeEnabled()
```

Checks whether the MarkupBean member is enabled.

3.4.3 Exiting Markup Mode

```
VueBean.setMarkupModeEnabled(false)
```

Checks whether the MarkupBean member is null, and if not:

- Sets the MarkupBean member to null
- Removes markup-specific mouse listeners
- Saves markup settings into the user's profile
- Broadcasts VueEvent.ONEXITMARKUPMODE

3.4.4 Adding an Entity to an Active Markup/Layer

```
MarkupBean.setMarkupEntityClass(<class name of desired markup entity>)
MarkupBean.setActionMode(MarkupBean.ACTION_MODE_ADD)
```

Adds a new markup entity to the active layer in an active Markup (based on the class name provided) through user input from the GUI. To programmatically add a markup entity, you must call: MarkupBean.addMarkupEntity(MarkupEntitySpec spec)

3.4.5 Enumerating Entities

```
MarkupLayer.getEntities()
```

or

```
MarkupBean.getMarkupEntities(MarkupLayer layer)
```

Returns an array of MarkupEntity objects in a markup layer.

3.4.6 Getting Entity Specification of a Given Entity

```
MarkupBean.getMarkupEntityFullSpec(MarkupEntity ent)
```

You must pass in the specific entity for MarkupBean to return its specification.

3.4.7 Changing Specification of an Existing Entity Programmatically

```
MarkupBean.exchangeMarkupEntity(MarkupEntity a, MarkupEntity b)
```

Allows you to dynamically change the properties of an existing entity. That is, it replaces markup entity *a* with markup entity *b*. Some properties can be directly changed via the following set methods of MarkupEntitySpec inherited from the MarkupGraphicSpec parent class:

- setColor
- setFillColor
- setFilled
- setFilltype
- setFont

- setLineType
- setLineWidth

For other properties, such as the entity position, entity size, entity text content, and so on, there are no set methods directly on the specification. As a result, you must do the following:

1. Create a new specification instance (with the new properties).
2. Create a new entity instance (with the new specification).
3. Use exchangeMarkupEntity to replace the existing entity.
4. Make a call to MarkupBean.repaint().

3.4.8 Adding a Text Box Entity

The following code shows how to add a text box entity programmatically.

```
import com.cimmetry.markupbean.*;
import com.cimmetry.gui.*;
.
.
.
public void addTextBox(String text){

  m_vueBean.setMarkupModeEnabled(true);

  CTextPane textPane = GUIFactory.createTextPane();
  textPane.setText(text);
  byte[] textRTF = textPane.getRTF();
  PAN_CtlRange rect = new PAN_CtlRange(m_vueBean.getViewExtents());
  rect.scale(0.2);
  TextBoxSpec spec = new
  TextBoxSpec(m_vueBean.getMarkupBean().getMarkupEntitySpec(),
          rect.min, textRTF, rect.max,TextBoxSpec.MRK_ALIGN_BOTTOMCENTER);
  m_vueBean.getMarkupBean().setMarkupEntityClass(spec.getEntityClassName());
  m_vueBean.getMarkupBean().addMarkupEntity(spec);
}
```

3.4.9 Open Existing Markup

```
MarkupBean.readMarkup(InputStream is)
```

InputStream can be relative to the client (for example, a locally-saved Markup), relative to the AutoVue server (for example, managed by AutoVue's markups.map file) or from a DMS/PLM/ERP.

To read a Markup from the AutoVue server, you first must get the InputStream by reading the Markup *Property* from the VueBean, and then choose a child property (that represents a Markup file) you want to read into the stream. The following code illustrates how to create a markup, save it, and then read it into the MarkupBean.

```
import com.cimmetry.markupbean.*;
.
.
Property[] name = {new Property(Property.PROP_DOC_NAME, <your Markup name>)};
Property prop = new Property(Property.PROP_MARKUP, name);
ByteArrayOutputStream os = new ByteArrayOutputStream();
m_markupBean.writeMarkup(os);
m_vueBean.writeMarkup(prop, os);
```

```
Property masterMarkup = m_vueBean.getMarkupProperty();
Property[] listMarkups = masterMarkup.getChildrenWithName(Property.PROP_MARKUP);
Property aMarkup = listMarkup[0];
InputStream is = m_vueBean.readMarkup(aMarkup);
m_markupBean.readMarkup(is);
…
```

3.4.10 Saving Markups to a DMS/PLM

Note: This example is not applicable if you are building an ISDK-based application.

The following example uses the same concept as saving a Markup back to the AutoVue server; you must set the appropriate *Property* and build the OutputStream. In order to build the Markup property, you need to first read the CSI_Markups property so that you can retrieve the values that the user sets in the Markup Save dialog.

```
private void saveMarkupToDMS() {
  // Gets the master markup property for the current file, that is,
  // the property containing the GUI and the markup list
  Property propMaster = m_vueBean.getMarkupProperty();

  // If none, the an output appears stating "Could not get master markup property"
  if ( propMaster == null ) {
    System.out.println("Could not get master markup property!");
    return;
  } else {
      // Get the GUI child property under master markup property
      Property[] listGuiProp =propMaster.getChildrenWithName(Property.PROP_GUI);
      if (listGuiProp == null || listGuiProp.length != 1) {
         System.out.println("No valid GUI property!");
         return;
      }
         Property propGui = listGuiProp[0];
      // Get the user field (Edit) child property under GUI property
      Property[] listEditProp =propGui.getChildrenWithName(Property.PROP_GUI_
         EDIT);
      if (listEditProp == null || listEditProp.length != 1) {
        System.out.println("No valid GUI edit property!");
        return;
      }
      Property propGuiEdit = listEditProp[0];
     // Get the list of user fields from save dialog all children items under GUI
     // edit property
     Property [] itemsEdit = propGuiEdit.getChildren();
    // ToDo: Use the list of edit items (GUI element) to construct a
    // save dialog to get user input for properties under PROP_GUI_EDIT.
    // Assume the input for attribute "CSI_DocName" we got from the dialog
    // is "myMarkup" and the input for attribute "CSI_MarkupType" is
    // "Normal", now the following code using the inputs to construct
    // the markup property contains these two attributes. In reality
    // there can be more than two attributes.
    Property [] listProp = {
        new Property("CSI_DocName", "myMarkup"),
        new Property("CSI_MarkupType", "Normal")
    };
    // Create a Markup property with the specified name & type properties
    Property propMarkup = new Property(Property.PROP_MARKUP, listProp);
    // Save the Markup
    try {
        ByteArrayOutputStream os = new ByteArrayOutputStream();
```

```
            m_vueBean.getMarkupBean().writeMarkup(os);
            m_vueBean.writeMarkup(propMarkup, os);
        } catch (MarkupIOException e) {
            System.out.println("Markup IO Exception!");
        }
    }
}
```

3.4.11 Adding a Markup Listener to Your Application

```
MarkupBean.getMarkupBroadcaster().addMarkupEventListener(MarkupEventListener mel);
```

A Markup listener listens for Markup events related to creating/saving/deleting Markups, Markup entities, Markup file information, fonts, Markup status, and so on. Note that you must implement the com.cimmetry.MarkupBean.event.MarkupEventListener interface (thereby implementing the onMarkupEvent method).

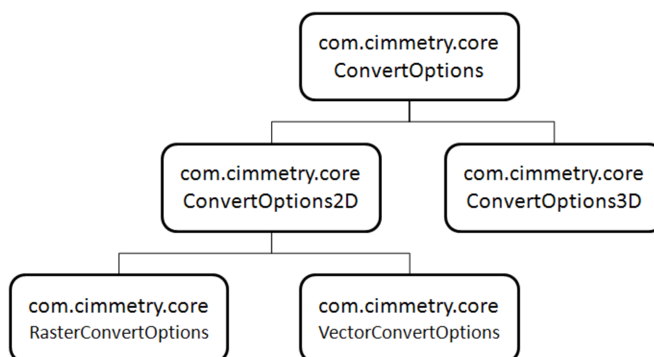
3.5 Converting Files

The following sections discuss how to execute common Conversion actions such as making a call to convert, converting an image to a JPEG using a custom conversion, and converting a vector file to a PDF. In some cases, there are additional methods to achieve the same functionality. Refer to the *VueBean Javadocs* for more information.

Note: Conversion of 3D files or pages containing 3D data is no longer supported.

The class hierarchy for conversion is as follows:

Figure 3–1 Conversion class hierarchy



Note: The classes represent the format which you are converting a file to. For example, if you are converting to a vector format, you should define a VectorConvertOptions and pass it into the conversion method.

3.5.1 Making a Call to a Convert Method

```
com.cimmetry.vuebean.VueBean.convert(ConvertOptions opts)
```

or

```
com.cimmetry.jvue.JVue.convertFile(ConvertOptions opts)
```

Once the convert options are defined, you must call one of the methods to convert.

> **Note:** When making a call from the VueBean you must call VueBean.convert. When making a call from the AutoVue applet layer, you must call JVue.convertFile.

3.5.2 Converting to JPEG (Custom Conversion)

To convert an image to a JPEG, you must use the encode() method that Java provides as part of the com.sun.image.coded.jpeg.JPEGImageEncoder interface. This method encodes buffers of the image data in JPEG data streams. To use this interface, you must provide the image data in raster format or a BufferedImage. The following example illustrates how to use this interface with the AutoVue API:

```
import java.io.*;
import java.awt.*;
import java.awt.image.*
import com.cimmetry.core.*;
import com.sun.image.codec.jpeg.*;
…

double scaling=0.5; BufferedImage bi = new BufferedImage(
  (int)( m_vueBean.getWidth()*scaling),(int)(m_vueBean.getHeight()*scaling),
  BufferedImage.TYPE_INT_RGB);

//Create or get Graphics and RenderOptions object here
Graphics2D g = bi.createGraphics();
RenderOptions optsRender = new RenderOptions();
//TODO: Initialize the Graphics object and RenderOptions object properly such
//as setting the source and destination.
try {
   m_vueBean.renderOntoGraphics(g,optsRender);
   FileOutputStream out = new FileOutputStream("c:\\temp\\my.jpeg");
   JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(out);
   JPEGEncodeParam param = encoder.getDefaultJPEGEncodeParam(bi);
   //TODO: Use the JPEGEncodeParam Interface to set the encoder parameters.
   encoder.encode(bi, param);
   out.flush();
   out.close();
} catch (Exception e) {
   System.out.println("Exception while converting to JPEG ");
   return;
}
…
```

3.5.3 Converting to PDF

To convert a vector file to a PDF you must perform the following steps:

- Create new VectorConvertOptions() object
- Set all appropriate convert options
- Call VueBean.convert and pass in the convert options

The following convertToPDF() method converts a vector file to a PDF.

```
public void convertToPDF() {
      VectorConvertOptions opts = new VectorConvertOptions();

      opts.setStepsPerInch(1);
      PAN_CtlFileInfo fi = m_vueBean.getFileInfo();
      PAN_CtlRange ps = m_vueBean.getPageSizeEx();

      if (fi.getType() == fi.PAN_DocumentFile) {
        ps = fi.getPageSize();
       }
      opts.setInputRange(ps);
      opts.setArea(ConvertOptions2D.AREA_EXTENTS);
      opts.setScaleFactor(100);
      opts.setScaleType(ConvertOptions2D.TYPE_SCALE);
      opts.setUnits(Constants.UNITS_INCH);
      opts.setPages(ConvertOptions2D.PAGES_ALL);
      opts.setFromPage(1);
      opts.setToPage(fi.getPagesNumber());
      opts.setFormat("PCVC_PDF");
      opts.setSubFormatID(0);
      opts.setFileName("c:\\output.pdf");

      //Uploads all currently loaded markups to the AutoVue server
      Property[] p = m_ vueBean.uploadMarkups();

      opts.setProperties(p);
      m_ vueBean.convert(opts);
}
```

3.6 Printing a File to 11x17 Paper

The following code prints a file to 11x17 paper size using the com.cimmerty.common.PrintProperties and com.cimmetry.common.PrintOptions classes.

```
import com.cimmetry.common.PrintProperties;
import com.cimmetry.common.PrintOptions;
public void printFile() {
  PrintProperties paramPrintProperties = new PrintProperties();
  PrintOptions po = new PrintOptions();
  po.setPrinter("AutoVue Document Converter");
  po.setPaperSize(po.PAPER_11X17);
  paramPrintProperties.setOptions(po);
  // The second parameter will enable the bypass of the Windows dialog
  m_ JVue.printFile(paramPrintProperties, true);
}
```

3.7 Monitoring Event Notifications

```
com.cimmetry.vuebean.event
```

If you have a requirement to programmatically execute specific file actions (such as rotation, zooming, and so on) as soon as a file has finished loading, you must monitor for the appropriate event notifications. If you do not check for file load completion, you might call a file action too early which may lead to errors.

The VueBean includes a set of notifications known as VueEvents. You can set up a listener to catch VueEvents, and catch the specific events that represent the completion

of a file loading. In order to catch file loading completion, you must use a file listener, with the VueFileListener interface.

The steps are as follows:

1. Implement your own VueFileListener.
2. In the onFileEvent method, check for occurrence of the Vue.Event.ONPAGELOADED event.
3. Implement your code to be executed when the Vue.Event.ONPAGELOADED event is detected.
4. Add your file listener to the VueBean.
5. Add this to your second applet.

3.8 Retrieving the Dimension and Units of a File

The following sample code shows how to get the dimensions and units of a file.

```
PAN_CtlDimensions pctlDim = m_vueBean.getFileInfo().getDimensions();
double height = pctlDim.getHeight();
double width = pctlDim.getWidth();
double depth = pctlDim.getDepth();
int units = m_vueBean.getFileInfo().getInsertion().units;
```

4 FAQ

The following sections provide frequently asked questions regarding the AutoVue API.

4.1 MarkupBean

Q: How do you determine the layer that a given entity is in?

A: Get the entity's specification and then get the layer from the specification.

Q: Do I have to implement the entire text editing dialog for the Text/Leader/Note entity?

A: No. The text editing dialog is inherent to these entities.

Q: An entity specification is tied to a given entity. Why was it decided to have an entity specification tied to the MarkupBean?

A: The entity specification on the MarkupBean was designed to be a reference to the most recent specification settings. When you create a new Markup entity, it defaults much of its specification attributes to the current specification in the MarkupBean. To retrieve the most recent specification settings, you can call MarkupBean.getMarkupEntitySpec().

> **Note:** The other two methods MarkupBean.getMarkupEntitySpec(MarkupEntity ent) and MarkupBean.getMarkupEntityFullSpec(MarkupEntity ent) are for when you need to get the specification of a specific entity.

Q: What is the difference between MarkupGraphicSpec and MarkupEntitySpec? Why are the specs such as ArcSpec subclass not derived directly from MarkupGraphicSpec?

A: The MarkupGraphicSpec is a top-level specification that manages visual attributes such as color, fill type, and so on. The MarkupEntitySpec is a top-level specification that has access to the overall structure such as the MarkupBean, Markups, layers, pages, and so on.
Since MarkupEntitySpec extends MarkupGraphicSpec, and this is the base class for all markup entities, the ArcSpec subclass is derived from MarkupEntitySpec.

Q: Can you work with MarkupBean independent of VueBean?

A: In theory it is possible to instantiate and work with MarkupBean without having a VueBean. However, there are not many use cases or practical reasons where this would be valuable.

Q: Are the Markup tree and Markup toolbars from the AutoVue Applet accessible if I am building a custom application from VueBean/MarkupBean?

A: No. The UI such as toolbars and Markup tree are part of the "JVue" class. If you build your solution using the JVue class you can use or customize this UI. However, if you build your solution directly from VueBean you need to implement your own UI.

Q: Is it possible to add AutoVue markup capabilities to a third-party application?

A: Yes. There are two primary ways to add markup entities using MarkupBean:

- With user input, using MarkupBean.setActionMode(MarkupBean.ACTION_MODE_ADD)
- Programmatically, using MarkupBean.addMarkupEntity(MarkupEntitySpec spec)

4.2 Printing

Q: What is the purpose of com.cimmetry.core.PrintInfo class?

A: It is used to pass information between the client and server.

4.3 Upgrading

Q: Will my custom code still work when I perform an AutoVue upgrade?

A: Yes. You must recompile your custom code against the latest release and update the path to the new jvue.jar.

4.4 General

Q: Can I perform file type-dependent operations?

A: Yes. You can do so by using the getFileInfo() method. The PAN_CtlFileInfo object that is returned can be queried to determine file format (such as vector, raster, spreadsheet, document, archive, or a database file).

Q: Can I delete server-side Markups when using the VueBean API?

A: No. It is not currently possible to programmatically delete server-managed Markups (referenced in the markups.map file on the server) using the VueBean API.

A Feedback

If you have any questions or require support for AutoVue, please contact your system administrator. If the administrator is unable to resolve your issue, please contact us using the links below.

A.1 General AutoVue Information

Web Site	http://www.oracle.com/us/products/applications/autovue/index.html
Blog	http://blogs.oracle.com/enterprisevisualization/

A.2 Oracle Customer Support

Web Site	http://www.oracle.com/support/index.html

A.3 My Oracle Support AutoVue Community

Web Site	https://communities.oracle.com/portal/server.pt

A.4 Sales Inquiries

E-mail	autovuesales_ww@oracle.com