# Oracle® AutoVue Integration SDK

Design Guide

Release 21.0.0

November 2015

ORACLE®

AutoVue Integration SDK Design Guide, Release 21.0.0

# Contents

# Preface

The *AutoVue Integration SDK Design Guide* provides a high-level overview of the Oracle AutoVue Integration Software Development Kit (ISDK)

For the most up-to-date version of this document, go to the AutoVue Documentation Web site on the Oracle Technology Network (OTN) 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.

## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website 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 in the AutoVue Integration SDK library:

- *Oracle AutoVue ISDK Overview*
- *Oracle AutoVue ISDK Installation and Configuration Guide*
- *Oracle AutoVue ISDK User Guide*
- *Oracle AutoVue ISDK Technical Guide*
- *Oracle AutoVue ISDK Acknowledgments*
- *Oracle AutoVue Javadocs*
- *Oracle AutoVue ISDK Security Guide*

# 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 Oracle AutoVue Integration Software Development Kit (ISDK) is a software package that is designed for Oracle partners and third party integrators to develop new integrations between AutoVue server and enterprise systems such as Document Management Systems (DMS), Production Lifecycle Management Systems (PLM), and so on. The goal of the integration is to enable communication between AutoVue and the enterprise systems as well as to integrate AutoVue's capabilities into their environments.

This document presents a high-level overview of the components that make up a typical integration and provides recommendations for features (such as security) when designing such an integration.

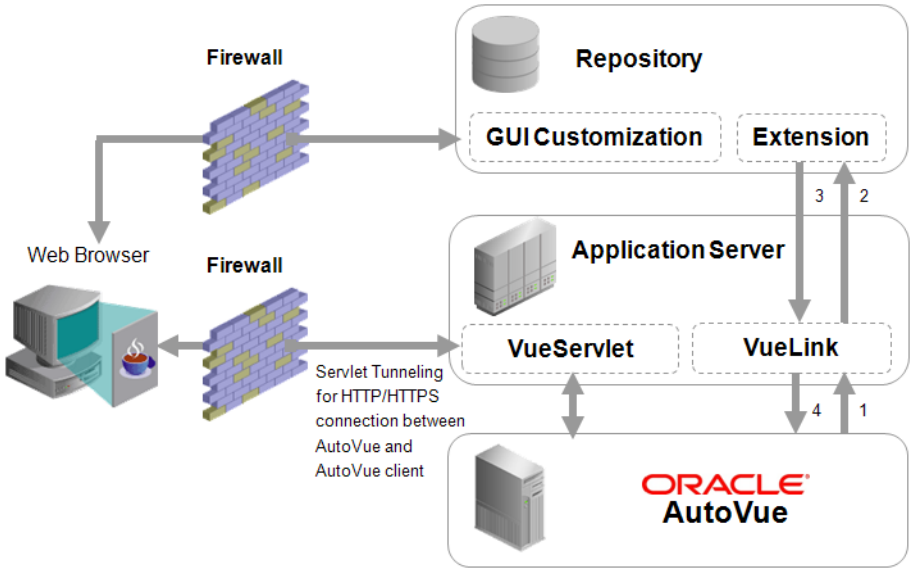
# 2

# AutoVue and Repository Integration

AutoVue is the key component in Oracle's Enterprise Visualization solutions. AutoVue solutions deliver native document viewing, markup, and real-time collaboration capabilities that streamline the information flow and collaborative processes across the global enterprise. AutoVue solutions help organizations in a variety of industries including Utilities, Industrial Manufacturing, Electronics & High Tech, Engineering and Construction, Aerospace and Defense, Automotive, and Oil & Gas. AutoVue streamlines visualization and collaboration across the global enterprise, improves productivity, reduces errors, and accelerates innovation and time to market. In an enterprise, AutoVue can be part of many business workflows and use cases such as collecting comments and annotations during a design review, recording the actions and results for a maintenance work order, comparing archived documents, and collaborating with other users.

AutoVue can offer its capabilities to many different enterprise systems/repositories such as DMS, PLM, Content Management Systems (CMS) and so on. AutoVue needs to be integrated into these repositories in order to be able to access the documents that are stored in them. There exist many such integrations. For example, there is an integration between AutoVue and WebCenter Content (WCC) and AutoVue and Oracle Agile PLM. The Oracle-developed integration is known as a VueLink. The VueLink provides an interface that allows communication between the repository and AutoVue in order to retrieve documents and to store data that is generated by AutoVue for those documents (such as annotations). The VueLink is a Java Web application that is hosted on a Java Web application server. The following figure shows how the communication between AutoVue and the repository is done through a VueLink.

*Figure 2–1 Communication between AutoVue and repository/backend system through a VueLink*




**Note:**

- AutoVue sends a request to the VueLink.
- VueLink forwards the request to the repository.
- The repository sends a response back to the VueLink.
- VueLink forwards the response to the AutoVue server.

Once AutoVue gets access to a document and other related data from the repository, it then streams the view of the document to the AutoVue client via the VueServlet. The AutoVue client is a lightweight Java Applet that interacts with the end user through a Web browser.

As shown in the diagram, the repository contains two important components: the repository extension and the GUI customization.

In order for the VueLink to communicate with the repository there needs to be a component on the backend-side whose interface the VueLink understands. This component is known as the repository extension. For more information, refer to Section 2.2, "Repository Extension."

In a complete integration, the AutoVue client should be launched from inside the repository user interface. The GUI Customization is applied to the repository user interface. For more information, refer to Section 2.1, "GUI Customization."

The application server component of the diagram includes the VueServlet and the VueLink. The VueServlet is a Java Servlet that acts as a tunnel between the AutoVue server and the AutoVue client. The client makes requests using the HTTP/HTTPS protocol to the VueServlet and the VueServlet communicates with AutoVue using AutoVue's socket ports. All the communication between the AutoVue client and the AutoVue server goes through the VueServlet. The VueServlet is available out of the box with AutoVue and can be easily deployed. Information on the VueLink is provided in Section 2.3, "VueLink."

## 2.1 GUI Customization

AutoVue provides the option of customizing your graphical user interface (GUI). The objective of the GUI customization is to launch the AutoVue client from inside the repository user interface. The AutoVue client can be embedded into the repository GUI or it can be launched in a separate window. In either case, an action must be defined in the repository GUI that invokes the AutoVue client. This action can be assigned to a user interface (UI) button, an icon, or a menu item inside the repository UI.

Depending on the underlying technology, the implementation of the GUI customization can vary from one environment to another. For example, a very simple implementation may be a hyperlink to an HTML page that loads the AutoVue client. A more sophisticated implementation may involve a repository-based scripting language or APIs. In both cases, you must refer to the repository's documentation for information on how to modify its UI and the available capabilities.

Since the customization can be applied to different places in the repository GUI, its implementation should be looked at from a usability point of view as well as from a technical point of view. A good example is customizing the search results page in the repository GUI. In this example, each document in the Search Results page is associated with a menu item or icon that launches the AutoVue client to view that particular document. A sample GUI customization is shown in the following screenshot. The Search Results page in Oracle Content Server GUI is customized to launch the AutoVue client.

***Figure 2–2 Sample GUI Customization***


## 2.2 Repository Extension

The repository extension is the layer on the repository that the VueLink communicates with. It allows the VueLink to access the repository the same way the end-user accesses the repository through the GUI.

**Note:** If the repository already provides a programming interface that gives access to all documents and related data required by the VueLink, then there is no need to develop a custom extension on the repository side for an AutoVue integration.

If this interface is not available, then a custom extension must be created using a technology that the repository supports. The extension must support the different requests that come from the VueLink. That is, the VueLink requests are related to retrieving and storing documents and their related data inside the repository. The extension can be built as a Web service or a Java Application Programming Interface (API). A Java API is the preferred approach as the performance is generally better and the overhead is lower in a Java to Java integration than in a Web service integration. This is because once the interface is provided by the repository then a VueLink-type component should be implemented to connect and communicate with it. The Web services should be used when a Java API cannot be provided (such as when integrating with a .NET environment). For more details about the required interface refer to Chapter 4, "Implementation."

## 2.3 VueLink

The VueLink is the integration component that acts as the gateway between AutoVue and the repository. The name VueLink is reserved for these types of Oracle-developed gateway components. Third-party integrators and partners should choose their own trademarks or preferred name for this piece of integration. However, regardless of its name, VueLink-type components enable AutoVue to access documents that are stored inside the repository. It also enables AutoVue to retrieve any data related to these documents from the repository. In addition, any data generated by AutoVue (for example, markups and renditions) can be stored into the repository using this component.

The VueLink is the center piece of an AutoVue integration with a repository. It is able to communicate with AutoVue and with the repository, thereby acting as a translator for each end and isolating AutoVue and the repository from each other's complexity. It is a Java Web application and needs to be deployed on a Java Web application server (such as WebLogic, GlassFish, Tomcat, and so on). In case of a Web service-based integration, the application server must support Java Web service technology. For more information, refer to Chapter 5, "Deployment of ISDK-Based Integrations."

Since the interface between the VueLink and AutoVue is the same for all VueLinks (regardless of the repository they are built for), it is good practice to have an integration framework that has built-in communication with AutoVue and is ready to be used as a starting point for building new integrations with any repository. The AutoVue Integration SDK is designed to fulfill this requirement. For more information, refer to Chapter 3, "AutoVue Integration SDK."

## 2.4 Optional Components

Before discussing the AutoVue Integration SDK, the optional components to be used in conjunction with building an integration are presented.

### 2.4.1 CAD Connector

One of the characteristics of CAD models is that often they are not stored in one document. For example, an airplane CAD model consists of many parts (such as

wings, wheels, and so on) which in turn have their own subparts. Each part or subpart might be designed and stored in a separate document and referenced in the airplane CAD model document directly or hierarchically. In order to view that airplane CAD model, all parts must be loaded and put together. These separate parts and subparts files are known as external references (XRefs). AutoVue supports loading and viewing documents along with their XRefs documents. However, in an integration, the XRefs support should also be provided at the repository level since they are all stored inside the repository. If the repository provides a mechanism to link documents to each other as references, then this mechanism can be used to provide XRefs support.

The storing and linking of references in a repository should not be done manually. In order to properly support the XRefs, some software tools should be provided that can import related documents from the CAD authoring software into the repository. An example is a CAD connector. A CAD connector is a software tool that integrates the repository with a CAD authoring software package (such as AutoCAD). It can check-in/check-out a set of related CAD files into/out-of the repository while preserving their relations and linkage.

**Note:** This software tool is not an AutoVue integration requirement. It is a facilitator for the repository to organize the XRefs.

# 3

# AutoVue Integration SDK

The AutoVue Integration SDK (ISDK) is a framework designed to help third-party integrators to develop and implement an integration between AutoVue and their repository. It saves time and effort in developing a new integration since it already has the necessary code to talk to AutoVue. It defines a data model and an interface for communicating with the repository extension. Integrators must understand this data model and implement the code for communication between the ISDK and the repository.

## 3.1 ISDK Data Model

This section provides an overview of the ISDK data model. More details can be found in the *AutoVue Integration SDK Technical Guide*.

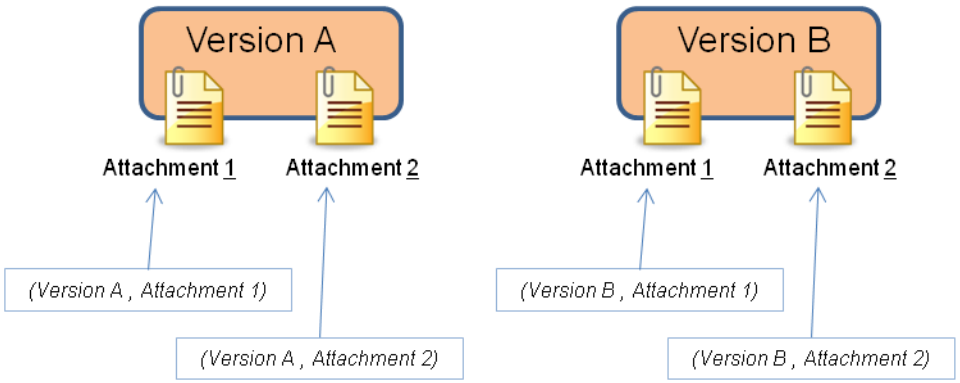
### 3.1.1 Document ID

In the ISDK every document is represented with a document ID (DocID) object which is a unique identifier for each file. This representation is chosen because unlike files in a file system (represented by path) or internet resources (represented by URI), objects in a repository are addressed by IDs. Since the architecture and data structure of each repository is different, no universal structure for a DocID can be defined. For this reason, the ISDK provides a flexible data model through an abstract class that allows integrators to define their own custom structure for DocIDs and register it into the ISDK framework. An important factor to consider is that the DocID should be defined in a way that it can uniquely identify each version (revision), attachment or any other entity in the repository that holds a file. For example, if there are multiple versions of a repository object, and each version holds multiple files as attachments, then the attachment of each version should have a unique DocID. This is displayed in the following figure.

**Note:** It is recommended that the DocID size should be less than 512 bytes and should not contain any variables. That is, it should not have any information that changes when multiple calls are made (for example, a session object).

**Note:** It is recommended that the size of the items listed in the DMSARGS should not exceed 512 bytes.

***Figure 3–1 Attachments in multiple versions of a repository object***




It is good practice to begin an integration by first defining a DocID structure. It is recommended to have a field in the DocID structure that displays the version of the document this ID is referring to.

---

**Note:** This is useful in certain AutoVue functions that deal with all versions of a document.

---

If your repository has a hierarchical structure (for example, folders/directories) it is recommended to extend the DocID structure to cover them as well (that is, each folder or directory should have its own unique DocID). In this case, it is helpful to add another field in the DocID structure that displays its type (for example, whether it is a file, folder, list, and so on).

---

**Note:** You should also consider that every AutoVue markup (annotation) or rendition that is being stored in the repository should have a unique DocID.

---

In a simple case, the DocID can be just a number that represents a single document in the repository. In other cases it can be a combination of some parameters that together locate the document inside the repository (for example, *site:1,list:3,item:25,version:2* ).

## 3.1.2 Document Attributes

The ISDK provides a data structure for holding attributes of documents. It is used for any attribute that the repository assigns to a document. Attributes can be single-value or multi-value (for example, when multiple options are selected from a list). They can also be associated with a list of pre-defined values (for example, a drop-down list). Some examples of attributes are document's title, size, last modified date, status, owner, and so on.

## 3.1.3 Actions on a Document

In the ISDK, a set of actions are defined that are to be performed on documents. The actions in the ISDK are Open, Download, GetProperties, SetProperties, Save and Delete. These actions are described in detail in the *AutoVue Integration SDK Technical Guide*. Each action requires a handler class that has to be registered in the ISDK framework. The GetProperties action is divided into a set of smaller actions (sub

actions) that each has its own handler class (examples of properties to get are: document size, last modified date, and so on).

All action handler classes register themselves into the framework. The ISDK defines an interface that all actions should implement in order to register themselves into the framework. Once the ISDK initializes, it instantiates all registered actions.

### 3.1.4 Security

To perform their tasks, action objects have to communicate with the repository. Since most of the repositories are controlled by some authentication and authorization mechanism, the integration needs to provide the authentication/authorization information to the action.

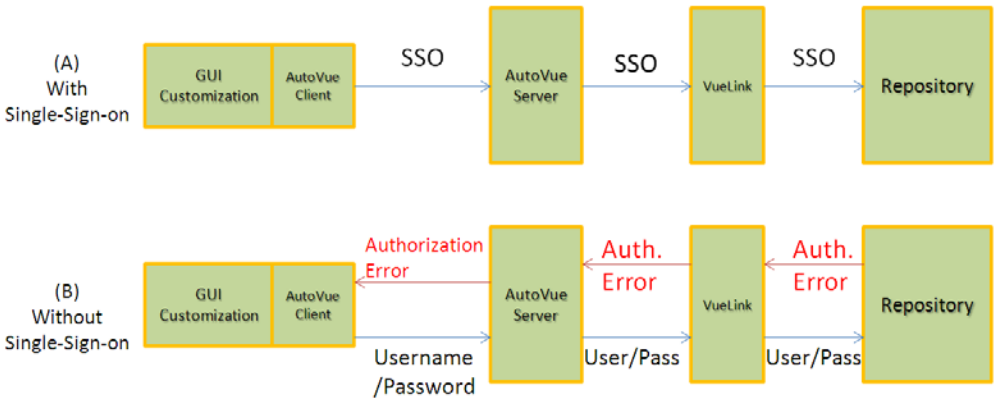
The ISDK is designed to accommodate any security mechanism that the repository has in place. By default, no particular security mechanism is enforced in the ISDK framework. It is up to the repository to define how the ISDK-based VueLink should communicate with it in a secure fashion. Using security credentials, there are two ways for the VueLink to communicate with the repository:

- Have a persistent connection
- Connect/disconnect when performing each action

In either case, the VueLink holds a session object for each user and the security credentials are stored in this session object for later use as long as the session is valid.

Another security related issue is the Single-Sign-On (SSO) versus the non-SSO, as demonstrated in the following figure.

***Figure 3–2 SSO vs non-SSO***




If the repository supports SSO (either through an external Identity/Access Management System or on its own) then it would be possible to use it. In this case, the SSO information can be passed from the AutoVue client to the ISDK-based VueLink so that it can log in to the repository automatically. In a non-SSO environment the repository will block VueLink from logging in by returning an authorization error (as shown in section B in figure above). In this case, VueLink propagates the error back to the AutoVue client and the AutoVue client asks the user to provide the credentials. These credentials are then passed to the VueLink in order to log in to the repository. Once the VueLink connects to the repository, the action can be completed.

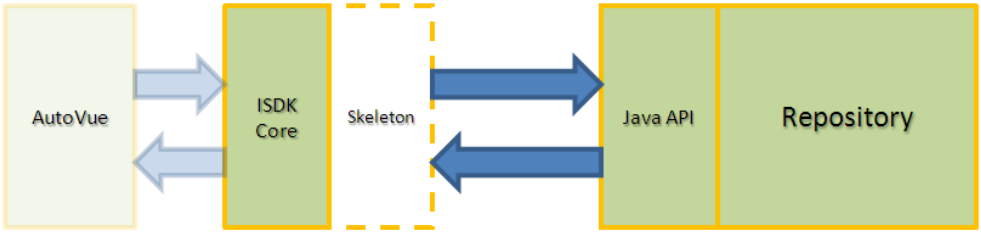
# 4

# Implementation

To speed up the integration and provide the integrators with a starting point, the ISDK includes a skeleton package and a Web service package.

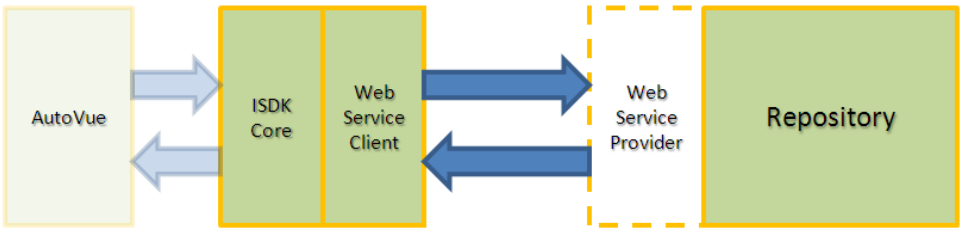
The ISDK Java skeleton package has the structure for building a new VueLink. The skeleton comes with a set of *TODO* comments in places where the integrators need to add their code. The ISDK Java skeleton implementation means adding code to the skeleton codebase so that it can communicate with the repository's Java API as shown in the following figure.

***Figure 4–1 Adding code to the skeleton codebase***




The Web service package includes a Web Services Description Language (WSDL) file that describes an interface for a Web service to be implemented by the repository. The package includes a client-side implementation of this WSDL. This client package itself is built using the ISDK Java skeleton. With the ISDK Web service package, the implementation means building a proper Web service provider based on the defined WSDL on the repository as shown in the following figure. This means more flexibility since the Web service provider can be implemented on any platform and with any programming language.

***Figure 4–2 Web service provider based on the defined WSDL***



The ISDK Java skeleton should be used when a Java API is available in the repository. The reason is the ISDK is written in Java and a Java-to-Java integration with a repository (if possible) performs better with less overhead than a Web service-based integration.

On the other hand, if Java API is not available on the repository (or the flexibility in implementation is more important and/or all other parties in the enterprise are using Web services to communicate), then the ISDK Web service package is more suitable for building the AutoVue integration.

The implementation steps are dependent on whether the ISDK Java skeleton or the Web service package is being used. However, the expected functionality of the integration can be understood in three phases that range from the most basic (phase one) to the more advanced (phase three) capabilities. The following sections discuss these integration phases.

## 4.1 Phase One

The requirement for phase one is viewing the document. To view the document, the integration should cover the Open and Download actions and a subset of GetProperties (get name, size, last modified date and multi-content values) actions.

## 4.2 Phase Two

Phase two of the integration adds the following capabilities:

- Save, update and delete markups (annotations) inside the repository
- Compare a document with other versions of the same document
- Download the external references (XRefs) of a document from the repository (if applicable)
- Save (and reuse) the renditions of a document into the repository
- Add the repository attributes to the print output in the header/footer sections

For this to happen, the integrators should add implementation for the Save and Delete and a subset of GetProperties related to listing versions, listing markups, listing XRefs, listing renditions and listing all attributes of a document.

As mentioned in Section 2.4, "Optional Components" the repository should support XRefs and the development of a CAD connector for the repository may be required.

## 4.3 Phase Three

Phase three of the integration adds the following capabilities:

- Search and browse the repository through the AutoVue client UI
- Use the AutoVue Intellistamp with the repository attributes

AutoVue Intellistamp is one of the AutoVue advanced markup features. For more information, refer to the *Oracle AutoVue User's Manual*.

For these features, integrators must add an implementation for SetProperties and the remaining subset of GetProperties that are defined to retrieve these information: search/browse UI, search/browse query results, and the collaboration-related data from the repository.

# 5

# Deployment of ISDK-Based Integrations

Once the development of an ISDK-based component is complete, it should be deployed on a Java Web application server. If the integration is done using the ISDK Web services package, then deployment should be done on an application server that supports Java Web services (that is, Java EE5 or higher).

The deployment may involve some configuration depending on its complexity. For example, if multiple instances of integrations are being used in a server farm, the deployment must be scaled for high usage. For more information, refer to Section 5.1, "Scaling for High Usage over Distributed Environments." Additionally, it may be required to support proper failover when deploying in a distributed environment.

For technical information on deploying the ISDK and supported Web application servers, refer to the *AutoVue Integration SDK Technical Guide.*

## 5.1 Scaling for High Usage over Distributed Environments

Depending on the number of concurrent users, the type and size of documents that users typically view, and whether files are to be loaded natively or from streaming files, it may be required to deploy your ISDK-based integration in a server farm. Additionally, it may be required to deploy it in distributed environments. In order to support proper failover in a distributed environment, the HTTP session needs to be replicated across all cluster nodes. In the event that a node fails, a second cluster node takes over and continues to process the requests. Seamless failover is when the user's actions are not disrupted and no authorization dialog is requested during this process.

Depending on the type of DMS integration and connection, login or session information may need to be remembered. For the information to be replicated across nodes, the objects attached to the HTTP session need to be serializable.

For example, in the FileSys sample integration, the com.cimmetry.vuelink.filesys.FilesysContext class manages this aspect. The back-end session objects may not always be serializable. The FilesysContext stores the username and password strings in the session. This allows the node that is taking over another session to reinitialize the connection with the back-end. The DMSSession class is provided by the ISDK to wrap the HTTP session variable. It provides the setAttribute() and getAttribute() methods to handle the storing of serializable objects to be saved and replicated. For more information, refer to the *ISDK Technical Guide*.

Consider the following when serializing objects:

- The DMS provides a session ID: If the back-end DMS provides a session ID, this session ID can be serialized into the DMSSession object. This way when a cluster node fails, the new node can pick up the replicated DMSSession and use the stored session ID to continue communicating with the back-end DMS.

- The DMS connection object is not serializable: If the connection object cannot be serialized into the DMSSession, the information needed for recreating this object should be serialized and replicated. This way, when the active node fails, the second node retrieves this information and recreates the DMS connection object.

If seamless failover is not possible, an authorization exception can be thrown to request the user login information. This way, the user retains the ability to save any markups that they have created provided that they can enter a valid username and password.

Non-serializable objects should be added to the DMSSession to increase performance and allow caching of data between requests. However, they need to be declared as transient in order not to break the session replication during failover. These transient objects will need to be regenerated once the session was migrated to a different cluster node.

For information on scaling AutoVue servers for high usage and seamless failover, refer to the "Scaling AutoVue for High Usage" section of the *Oracle AutoVue Client/Server Deployment Planning Guide*.

# 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 |