Oracle AutoVue Web Services Release 20.2.1

Developer's Guide

ORACLE

September 2012

Contents

PREFACE 4
- **Audience** 4
- **Related Documents** 4
- **Conventions** 4

INTRODUCTION 5
SYSTEM REQUIREMENTS 7
ARCHITECTURE 7
- **How AutoVue Web Services Works** 7

HOW TO USE AUTOVUE WEB SERVICES 8
- **Java Client** 8
  - Generating Client Proxy Using WSimport 8
  - Importing and Using Client Proxy 9
- **.NET Client** 10
  - Generating Client Proxy using WSDL 10
  - Importing and Using Client Proxy in Microsoft Visual Studio 2005 10
  - Importing and Using Client Proxy in Microsoft Visual Studio 2008 13
- **HTTPS/SSL** 16

AUTOVUE WEB SERVICES AND DMS INTEGRATION 16
- **VueLink for Oracle UCM** 17
- **VueLink for Documentum** 17
- **Third-Party Integration** 18
  - AutoVue ISDK Integration Example 18

ORACLE WEB SERVICES MANAGER 19
TESTING AUTOVUE WEB SERVICES 20
- **AutoVue Web Services Methods** 22

AUTOVUE WEB SERVICE API 26
APPENDIX A - SAMPLE CLIENT CODE IN JAVA 27
- **Web Services Sample Client Code for Printing** 29
  - Packet Printing 29

FEEDBACK 30
- **General AutoVue Information** 30
- **Oracle Customer Support** 30
- **My Oracle Support AutoVue Community** 30
- **Sales Inquiries** 30

Preface

The *AutoVue Web Services Developer's Guide* describes how to create a Web service client stub for the AutoVue Web Services package, how to use the generated code inside your application, and how to call AutoVue Web Services methods from inside your code.

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

The *AutoVue Web Services Developer's Guide* is intended for third-party developers (for example, integrators) who want to implement SOAP-based integration with AutoVue.

Related Documents

For more information, refer to the following documents:

- *Installation and Configuration Manual*
- *Deployment Guide*
- *Overview*
- *Release Notes*

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 the text.
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.

Introduction

Note: It is recommended that you first review the *Oracle AutoVue Web Services Overview* and *Oracle AutoVue Web Services Installation and Configuration Manual*. These manuals are located in the **docs** directory. Additionally, you can access them from the readme file, *readme.html*, located in the root folder where you installed AutoVue Web Services.

The AutoVue Web Services package provides a standard interface for developers to take advantage of AutoVue functionalities in the environment and programming language of their choice, as long as Simple Object Access Protocol (SOAP)[1] is supported by their platform.

AutoVue Web Services represents many AutoVue functionalities such as print, convert, text extraction, and more in the structure defined by Web Service Definition Language (WSDL). These functionalities are discussed in more detail later in the manual.

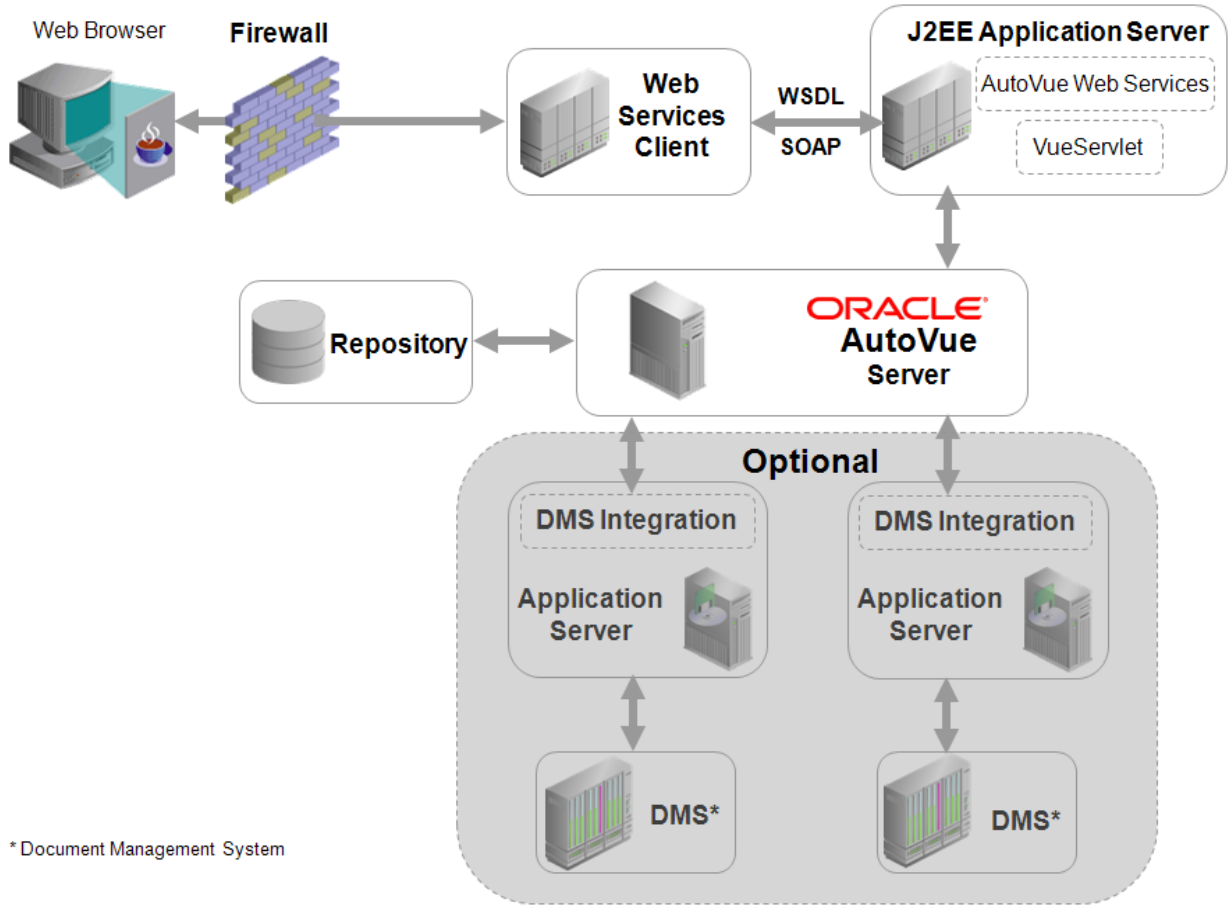
This document is intended for systems integrators or developers who want to implement SOAP-based integration with AutoVue. This manual also serves as a good starting point for developers and professional services to become more familiar with the technical details of this package; it shows you how to create a Web service client stub for the AutoVue Web Services package, how to use the generated code inside your application, and how to call AutoVue Web Services methods from inside your code.

The AutoVue Web Services package is designed to work seamlessly with Document Management Systems (DMS)[2] through various DMS integrations. It can also work with local files and Uniform Resource Indentifiers (URIs) that are accessible to the host machine.

Note: Not all of AutoVue's functionalities are represented in the AutoVue Web Services package. This is because many of the functionalities require user interaction (for example, online collaboration, digital mockup, and so on) and are not suitable for application-to-application communication; which is the main objective of AutoVue Web Services

1. SOAP is a standard protocol that is governed by the World Wide Web Consortium (**http://www.w3.org/TR/soap**).
2. In this document, a DMS/PLM/ERP/UCM system is referred to as DMS.

The following diagram displays the communication process for AutoVue Web Services:



System Requirements

AutoVue Web Services uses Java annotation and other features introduced in Java EE 5. As a result, it must be deployed on a Java EE 5 (or later) certified application server.

For a complete list of system requirements specific to your platform, refer to the *Oracle AutoVue Web Services Installation and Configuration Manual.*

Architecture

The AutoVue Web Services package acts as a layer around Oracle AutoVue. It exposes certain AutoVue functionalities as Web methods, and translates AutoVue Web Services requests to and from AutoVue (for example, AutoVue messages to AutoVue Web Services responses). Additionally, AutoVue Web Services enables AutoVue to communicate with any third-party application that wants to invoke AutoVue in a Service Oriented Environment (SOE).

How AutoVue Web Services Works

Once the AutoVue Web Services package is deployed, its Web Service Definition Language (WSDL) interface (VueBeanWS?wsdl) provides the gateway to client applications. Client applications can detect available Web methods and their input/output parameters through the WSDL, and generate a proper communication proxy. Since most programming languages have tools to automatically generate Web service client stubs, it does not take much effort to create one for AutoVue Web Services (all AutoVue Web Services methods are defined in a single WSDL).

Once the client stub is created in a specific programming language it can be reused by applications in that language and a few lines of code are needed to call any AutoVue Web Services method through the client proxy.

Several AutoVue Web Services methods accept options (for example, print and convert) as an optional input parameter. The structures of these options are defined in a XML Schema Definition (XSD) that is linked to the WSDL and are generated in the client stub code automatically. The client application instantiates these options and sets their variables to desired values and invoke the AutoVue Web Services method.

After a successful call to an AutoVue Web Services method, it either returns the desired output, or in the event there is an issue with the input parameters, it returns an error message.

Note: The output types of AutoVue Web Services methods vary from one to another. Regardless, all custom output structures are defined in the XSD and are generated automatically once the client stub is generated.

How to use AutoVue Web Services

The first step in using AutoVue Web Services is to create a client proxy in your desired language. Then, after installation and deployment of AutoVue Web Services, look for the URL that points to the WSDL (for example, http://host:port/AutoVueWS/VueBeanWS?wsdl). This URL is needed for any utility that you use to create your client stub.

Java Client

There are two steps in generating a Java client proxy:

1 Generating Client Proxy Using WSimport

2 Importing and Using Client Proxy

Generating Client Proxy Using WSimport

To generate the Java client proxy, you can simply call **wsimport,** which is bundled with Java Standard Edition 6, from the command line with the **-keep** option and pass the WSDL's URL.

For example:

```
wsimport -keep http://host:port/AutVueWS/VueBeanWS?wsdl
```

This will provide the following output:

```
parsing WSDL...
generating code...
compiling code...
```

After returning back to the command line, a new Java package is created in the current location.

The directory structure of the package should be *com\oracle\autovue\services.* All client proxy codes are generated inside this directory.

For detailed information regarding available wsimport options refer to the following link:

http://java.sun.com/javase/6/docs/technotes/tools/share/wsimport.html

Importing and Using Client Proxy

The next step is to import and instantiate the generated package inside your client code.The following code demonstrates how to call an AutoVue Web Services method:

```
import com.oracle.autovue.services.*;


public class AutoVueWSClient {


   public static void main(String[] args) throws Exception  {


      //create service
      VueBeanWS_Service service = new VueBeanWS_Service();


      //create proxy
      VueBeanWS proxy = service.getVueBeanWSPort();


      //call autovue ping Web method
      System.out.print (proxy.ping("Hello from Java") );
 }
}
```

The first line, `import com.oracle.autovue.services.*;` imports the AutoVue client stub package generated by the wsimport tool. To call a Web method, you need to first instantiate the *VueBeanWS_Service* object. From this object, instantiate the *VueBeanWS* class which is the proxy for calling all AutoVue Web Services methods.

The simplest AutoVue Web Services method to run—which is also a good method for testing—is the *ping* method. It verifies that AutoVue Web Services is running and responding correctly.

After running the above code, you should receive an output similar to the following:

```
Server Date/Time: //some number showing current time at the server side
Pool, Size: //some number showing the size of the Web service pool
Max waiting time:// some number showing maximum timeout (in seconds) while waiting to
borrow VueBean from VueBean pool
Pool, # of active: //number of active objects in the pool
Pool, # of idle: //number of idle objects in the pool
initialJVueServer: //URL for VueServlet used to connect to JVue server
vuelink Protocol(s): //some vuelink protocols e.g. vuelink1;vuelink2;vuelink3
vuelink PropsDir: //some local path to vuelink properties file
destination DIR: //some local path to output generated files
log4j config file: //some local path to log4j properties file
```

Note: Optionally, you can pass a string value to this method.

For a detailed description on calling AutoVue Web Services methods, refer to **Testing AutoVue Web Services.**

.NET Client

Generating Client Proxy using WSDL

The .NET environment also provides a tool for creating an AutoVue Web Services client proxy, *wsdl.exe*.
To generate the AutoVue Web Services client proxy, from the command line, you can simply pass WSDL's URL to wsdl.exe.

For example:

```
wsdl.exe    http://host:port/AutoVueWS/VueBeanWS?wsdl
```

The tool generates a file, *VueBeanWS.cs,* in the same location as *VueBeanWS?wsdl.*

Note: If you want the information being sent between a client and AutoVue Web Services to be secure, you should enter the HTTPS protocol in the URL instead of HTTP. For more information, refer to **HTTPS/SSL**.

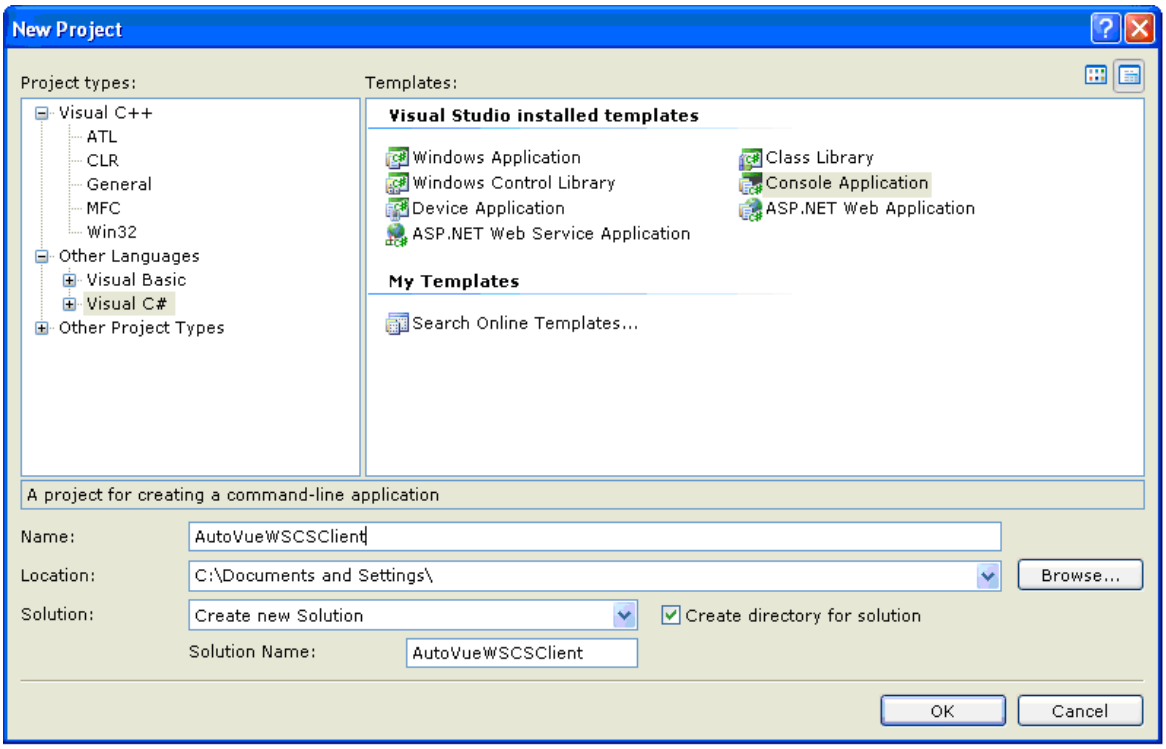
Since Microsoft Visual Studio is the primary IDE for .NET development, you can also use it to create and use the AutoVue Web Services client proxy. For more information on using Microsoft Visual Studio for generating a client proxy, depending on your environment, refer to **Importing and Using Client Proxy in Microsoft Visual Studio 2005** or **Importing and Using Client Proxy in Microsoft Visual Studio 2008**.

Importing and Using Client Proxy in Microsoft Visual Studio 2005

By using Microsoft Visual Studio 2005, you can generate the AutoVue Web Services client proxy without using the command line.

Note: Make sure you install Microsoft Web Service Extension (WSE 3.0) before proceeding with the client proxy generation.

1 After starting Visual Studio, create a new console application in a C# project (as shown in the following figure).

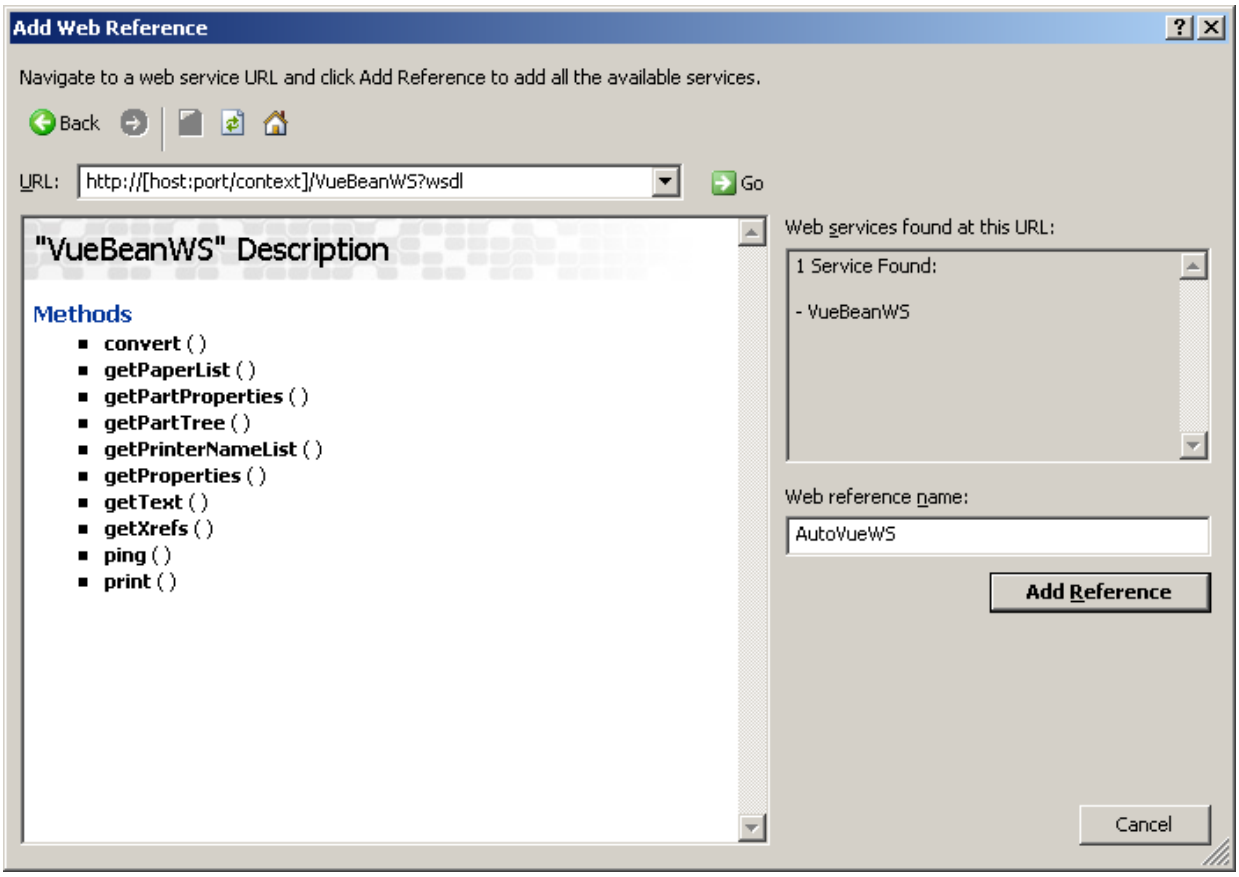


2 In the newly created project, from the **Solutions Explorer** tree, right-click **References**, and then select **Add References.**
The Add Reference window appears.

3 Click the **Browse** tab and navigate to the WSE 3.0 installation directory. Select the **Microsoft.Web.Services3.dll** file and then click **OK.**

4 From the **Solutions Explorer** tree, right-click **References**, and then select **Add Web References.**
The Add Web Reference window appears.

5 Enter the AutoVue WSDL's URL in the **URL** field, and then click **Go.**
The VueBeanWS Web service, along with all of its Web methods, are displayed in the Add Web Reference window.



6 Configure the project configuration file with the WSE 3.0 configuration tool:

a. From the **Start** menu, select **Program Files, Microsoft WSE 3.0,** and then select **Configuration Tool.**
The WSE 3.0 configuration tool starts.

b. From the **File** menu, select **Open,** and then select the project configuration file.

c. From the **General** tab, select the **Enable this project for Web Services Enhancements** check box.

d. From the **Messaging** tab, select **On** for the **Client Mode** option.

7 Optionally, provide a name for the Web Reference (for example, AutoVueWS), and then click **Add Reference.**
The proxy code is generated and added to your project.

At this point, you can import the proxy to your application (for example, Program.cs) and call the AutoVue Web Services methods.

The following code demonstrates a sample C# code that calls the Ping Web method:

```
using System;
using AutoVueWSCSClient.AutoVueWS;

namespace AutoVueWSCSClient
{
    class Program
    {
        static void Main(string[] args) {

            try {
                VueBeanWSWse vuebean = new VueBeanWSWse();
                Console.Write(vuebean.ping("Hello from C#"));
            }   catch (Exception e){
                Console.Write(e);
            }
        }
    }
}
```

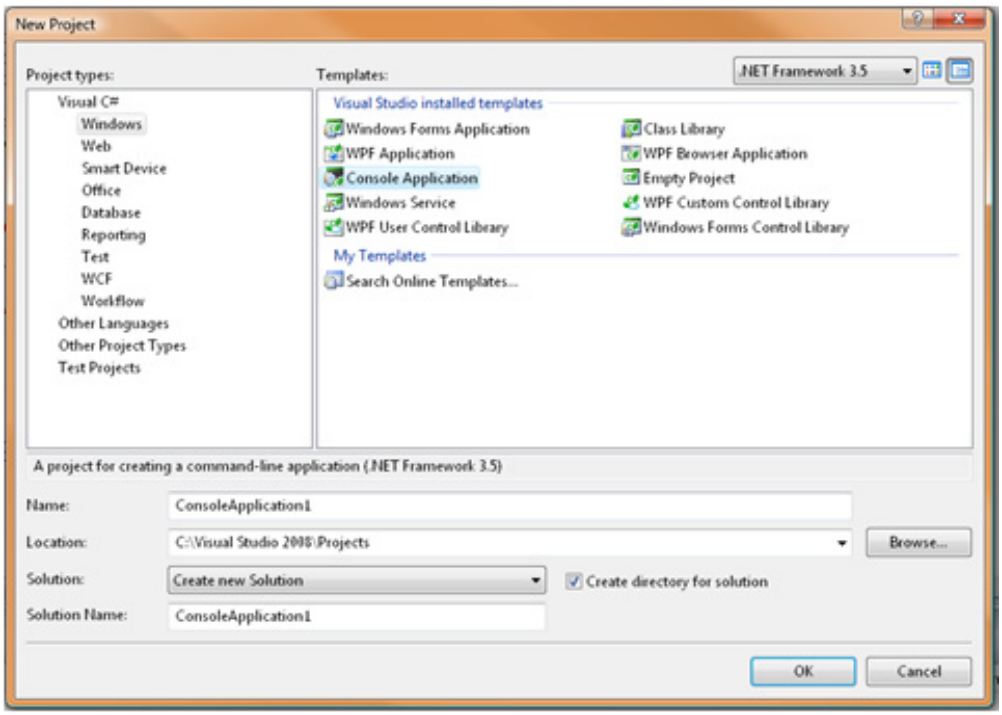
As with the ping Web method, you can call all other VueBeanWS Web methods by passing them input parameters. For more information on input/output parameter, refer to the AutoVue Web Services methods descriptions in **Testing AutoVue Web Services**.

Importing and Using Client Proxy in Microsoft Visual Studio 2008

As with Microsoft Visual Studio 2005, you can generate the AutoVue Web Services client proxy without using the command line in Visual Studio 2008.

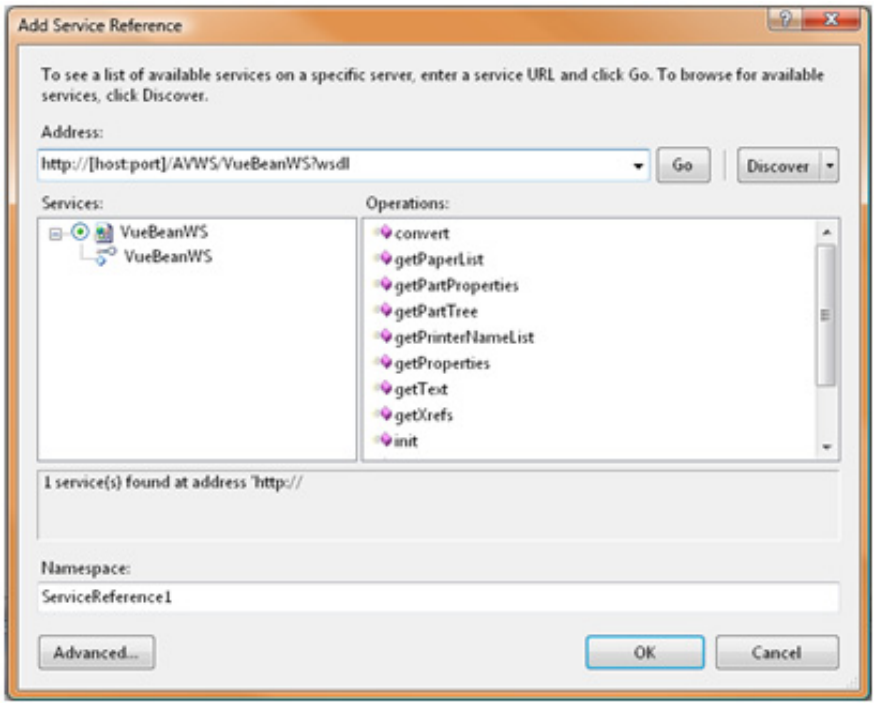
Note: There is no need for WSE as was required in VS 2005.

1 After starting Visual Studio, create a new console application (optionally choose the name AutoVueWSClient). As shown in the following figure, C# is the preferred coding language.



2 In the newly created project, from the Solutions Explorer window, right-click on **References**, and then select **Add Service References**.
The Add Service Reference window appears.

3 Enter the AutoVue WSDL's URL in the URL field, then click **Go**.
The VueBeanWS Web service and its Web methods are displayed in the Add Service Reference window.



4 Optionally, provide a new Namespace (for example, AutoVueWS), and then click **OK**.
The proxy code is generated and added to your project.

5 On the Solution Explorer window double-click on the **app.config** file to be opened for editing.

6 Inside the file, locate `messageEncoding="Text"` which is part of the attributes for this binding: `<binding name="VueBeanWSPortBinding"` under the `<basicHttpBinding>` .

7 Change the value of the attribute to `Mtom`. That is, `messageEncoding="Mtom"`

8 Optionally, increase the values of the `maxBufferSize` and the `maxReceivedMessageSize`. This is useful when using the `convert()` method for converting large files, because the conversion result is returned in binary format attached to the response.
At this point, you can import the proxy to your application (for example, Program.cs) and call the AutoVue Web Services methods.
The following code demonstrates a sample C# code that calls the ping Web method:

```
using System;
using AutoVueWSCSClient.AutoVueWS;
namespace AutoVueWSCSClient
{
 class Program
 {
  static void Main(string[] args)
    {
      try
      {
        VueBeanWSClient vuebean = new VueBeanWSClient();
        Console.Write(vuebean.ping("Hello from C#"));
      }
      catch (Exception e)
      {
        Console.Write(e);
      }
    }
  }
}
```

As with the ping Web method, you can call all other VueBeanWS Web methods by passing them input parameters.

For more information on input/output parameter, refer to the AutoVue Web Services methods descriptions in **Testing AutoVue Web Services**.

HTTPS/SSL

Security plays an important role in communication between applications. When it comes to Web services, this issue is even more critical. As a result, it is highly recommend to only use HTTPS protocols to call AutoVue Web Services.

To run and use AutoVue Web Services over SSL, you must first deploy AutoVue Web Services on a secure server, import the server certificate to your client environment, and then generate and use the client proxy in the same manner as described in **Importing and Using Client Proxy**. Additionally, for SSL, you must use a secure connection over HTTPS to generate and use the client code (for example, https://host:port/AutVueWS/VueBeanWS?wsdl).

If you are using Oracle Weblogic 11g Release 1 to deploy AutoVue Web Services, you can use the self-signed certificate that comes with the application server out of the box:

1 Export the certificate from Oracle Weblogic 11g Release 1 into a file.
You can do so through a Web browser.

2 Import the certificate into your client machine.

3 Follow the instructions in **Importing and Using Client Proxy** to generate and use the client proxy.

Note: Make sure you provide the HTTPS address of the WSDL.

AutoVue Web Services and DMS Integration

In addition to standard protocols supported by AutoVue Server (such as http:// and ftp://), and some custom protocols defined by AutoVue Server (for example, server://), AutoVue Web Services architecture allows flexible communication with DMS integrations in the same way as passing a URI. As a result, the client can send information about a document that is inside a DMS repository to AutoVue Web Services. Additionally, if an existing DMS integration is already set up, AutoVue Web Services can communicate with the DMS integration and access the document in order to process the client's request.

As with standard protocols such as http and ftp, the AutoVue Web Services administrator defines a custom protocol for each DMS integration and assigns a properties file on the AutoVue Web Services server that contains connection information for that specific DMS integration.

For example, if a DMS integration protocol is defined with the name *DMS_Integration_1*, then a *DMS_Integration_1.properties* file contains the location information and any other static data that is needed to communicate with an existing DMS instance. Client code can easily call AutoVue Web Services and pass a valid DMS document ID, as well as use the term *DMS_Integration_1* as prefix (for example, DMS_Integration_1:// dID=12345). Once AutoVue Web Services finds a match between the DMS integration protocol name in the request and a defined custom protocol (in this case, DMS_Integration_1), it treats the rest of the string as a document ID and passes it to the DMS instance.

Note: The name of the DMS integration protocol is arbitrary and can be configured in AutoVue Web Services. However, both associated properties files on the server and client code must use the same name.

One way for the client to find out whether any DMS integration protocols are defined on the server, is to call the *ping* Web method. One of the outputs of the ping Web method is *vuelinkProtocol*, and its value is a semi-colon (;) separated list of DMS integration protocols that are defined by the AutoVue Web Services administrator.

Refer to the *Oracle AutoVue Web Services Installation and Configuration Manual* for information on configuring vuelinkProtocol.

Note: It is important for the administrator to use meaningful names for DMS Integration protocols to avoid any confusion on the client side. For example, *vuelinkUCM://,* and so on. Additionally, if more than one instance of the same DMS integration is setup with AutoVue Web Services, a numbering scheme is suggested. For example, vuelinkUCM1://, vuelinkUCM2://, and so on.

Because each DMS integration and related DMS repository follow different standards of addressing documents, the structure of the document ID varies from one DMS integration to another. It is important to follow the string representation of document IDs that are defined in this section.

The following sections demonstrate string representations of the document ID for these supported DMS integrations (assuming a custom DMS integration protocol is setup and registered with AutoVue Web Services by the server administrator):

- VueLink for Oracle UCM
- VueLink for Documentum
- Third-Party Integration
- AutoVue ISDK Integration Example

VueLink for Oracle UCM

Note: Oracle UCM has been renamed as Oracle WebCenter Content (WCC).

The string representation of a document ID in VueLink for Oracle UCM is as follows:

```
dID=some_id_number[&Markup_BasedID=some_id_number][&Format=some_format]
[&Extension=some file ext]
```

Where:

dID: The valid document ID of the desired document.

Markup_BasedID: The valid document ID of the base document (only meaningful and needed when the document ID belongs to an AutoVue Markup).

Format: The format of the document according to what is defined inside the Oracle UCM (optional, but it is needed when the document ID belongs to an XRef folio).

Extension: The filename extension of the document according to what is defined inside the Oracle UCM (optional, but needed when the document ID belongs to an XRef folio and Format is not included).

The following are examples of URI values when invoking AutoVue Web Services for an Oracle UCM document (assuming protocol name is vuelinkUCM):

```
vuelinkUCM://dID=227&Markup_BasedID=228
vuelinkUCM://dID=350Extension=slddrw
vuelinkUCM://dID=270&Format=Application/dwg&Extension=dwg
vuelinkUCM://dID=253&Extension=xcsr
```

VueLink for Documentum

The string representation of a document ID in VueLink for Documentum is as follows:

```
WebTopURL?userName=some_name&docbase=some_docbase_name&sessionid=webtop_seession_id&
objectid=some_object_id&rendition=some_file_format
```

WebTopURL: The URL for webtop.

userName: A valid webtop UserName.

docbase: A valid docbase name.

sessionid: A valid webtop session ID.

objectid: A valid ID of an object in the above docbase.

rendition: A valid Documentum format.

The following are examples of URI values when invoking AutoVue Web Services for a Documentum document (assuming protocol name is vuelinkDocumentum):

```
vuelinkDocumentum://http://[host:name]/Webtop6?userName=Administra-
tor&docbase=demo&sessionid=s7&objectid=0901869f80002565&rendition=unknown
```

Third-Party Integration

You must construct a document ID for a file stored inside Third_Party_Name DMS using the Third_Party_Name protocol.

For example: Third_Party_Name://Third_Party_NameDocID=123&Format-dwg

Note:

- The prefix used in your document ID must match your properties filename (in this case, Third_Party_Name).
- To properly access files stored inside your Third_Party_Name DMS repository, the syntax of your document ID should match one that is understood by your DMS integration servlet.
- Invoke AutoVue Web Services on the document ID.
  For example: getXrefs()/getText()

AutoVue ISDK Integration Example

The string representation of document ID in AutoVue ISDK (filesys) is as follows:

```
RootURL/some_repository/some_file_name/some_file_name(some_version)/ some_file_name
```

RootURL: The value defined for parameter RootURL in web.xml of ISDK (filesys).

some_repository: A valid repository name which the file belongs to.

some_file_name: A valid file name that exists in the repository.

some_version: A valid version number for the file.

The following are examples of a URI value when invoking AutoVue Web Services for an ISDK document (assuming that the protocol name is vuelinkISDK):

```
vuelinkISDK://http://localhost/filesysRepository/2D/AutoCAD.dwg/AutoCAD.dwg(1)/Auto-
CAD.dwg
```

```
vuelinkISDK://http://localhost/filesysRepository/3D/Hard Drive.CATProduct/Hard
Drive.CATProduct(1)/Hard Drive.CATProduct
```

Note:

- It is important that the prefix used in your document ID matches your properties filename (in this case, vuelinkISDK).
- No authentication is required in order to access files in AutoVue ISDK (filesys).
- Invoke AutoVue Web Services on the document ID.
  For example: getXrefs()/getText()

Oracle Web Services Manager

Oracle Web Services Manager (OWSM) is a component of the Oracle SOA suite. It can be used as a proxy for your AutoVue Web Services, and you can assign it different policies and rules to access AutoVue Web Services. Additionally, you can monitor accesses to your AutoVue Web Services and review different statistics and logs that are provided by OWSM.

OWSM can also be used to perform a simple test of AutoVue Web Services. It can generate an input form for any AutoVue Web Services methods and invoke them through a Web browser. AutoVue Web Services can be easily tested using the Oracle Web Service Manager Test tool.

For more information about Oracle Web Service Manager, refer to the following URL:

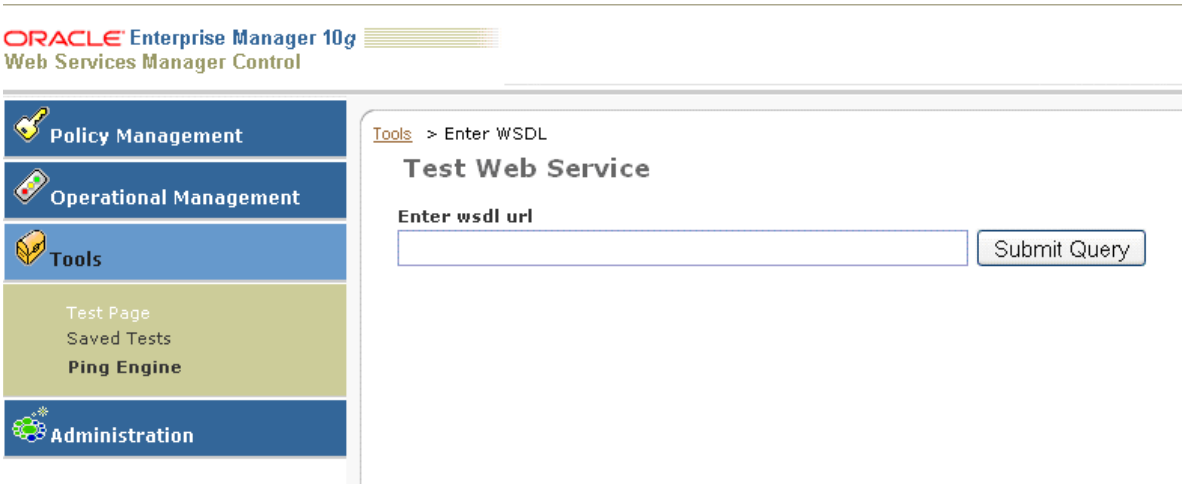
http://www.oracle.com/technology/products/webservices_manager/index.html

Testing AutoVue Web Services

For this manual, Oracle SOA Suite 10g is used to test AutoVue Web Services.

1 Start Oracle SOA Suite 10g.

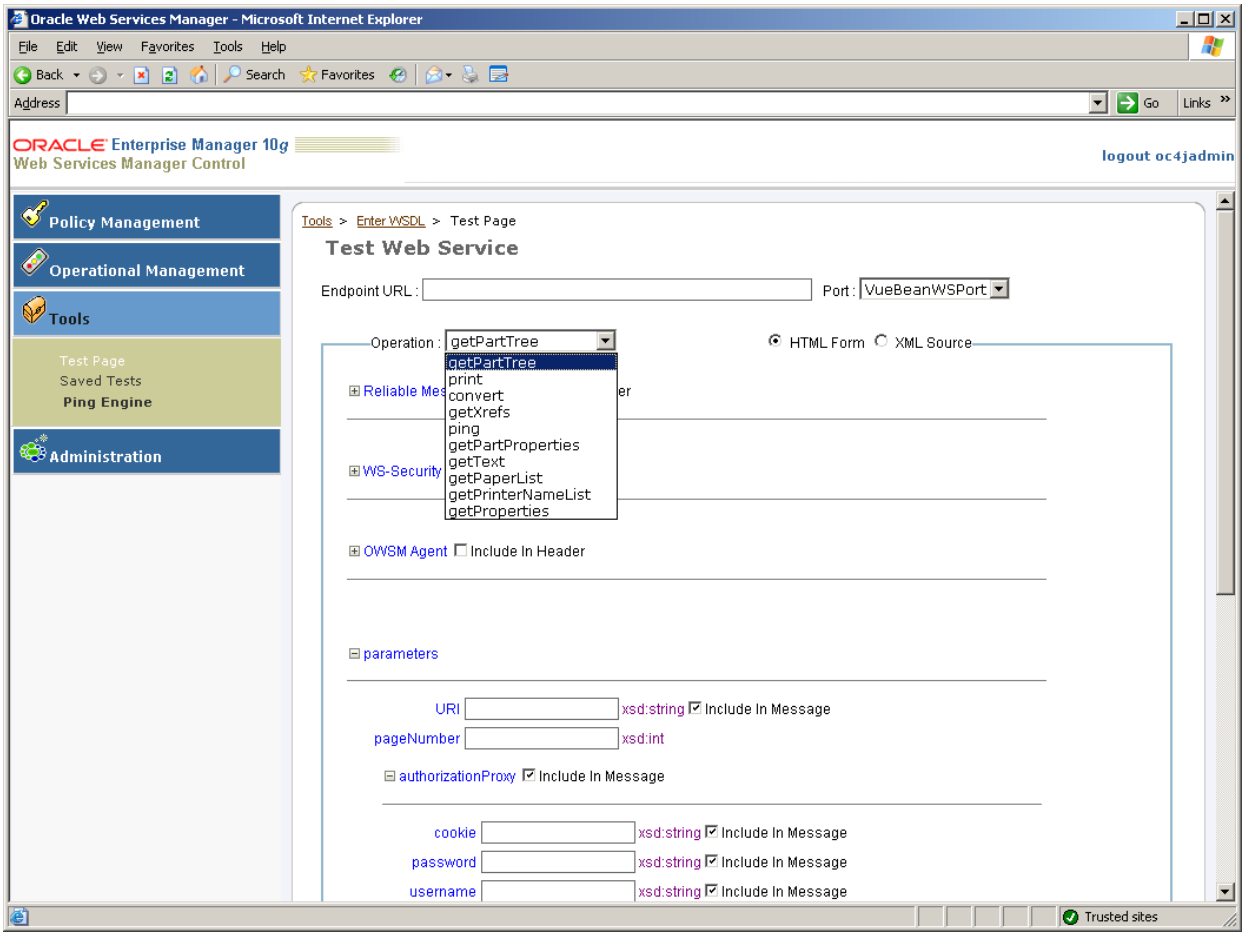
2 As shown in the following screen shot, from the Web Service Manager Control page, click **Tools,** and then click **Test Page.**



3 Enter the AutoVue Web Services WSDL's URL in the **Enter wsdl url** text box. For example, `http://AVWSHost:7011/AutoVueWS/VueBeanWS?wsdl`

4 Click **Submit Query.**

As shown below, the Test Web Service page reloads with an input form that is ready to invoke one of the AutoVue Web Services methods.



5 From the **Operations** list, select a Web method.

For information on the available Web methods, refer to **AutoVue Web Services Methods**.

AutoVue Web Services Methods

The following table provides a summary of the available AutoVue Web Services methods. After selecting the method and entering the required information, click **Invoke** to send the request to the Web Services provider.

Note: If the method calls a file from inside the DMS repository that requires authentication, you must provide the required credentials.

Web Method	Description
getPartTree	This part tree extraction Web method returns a list of parts contained in a given file. From the Operation list, select **getPartTree** and wait for the page to refresh. 1 To invoke this service, enter a valid URI in the **URI** text box. 2 If the URI is a VueLink DocID, or an address that needs authentication, you should also enter the username/password and/or cookie depending on what is required. 3 In the **pageNumber** field, enter a value less or equal to the number returned by getProperties. **Note:** The dmsArguments section is optional and is only needed if required by a VueLink. To add more DMS arguments, click ✚.
print	This printing Web method sends a given file to a printer for printing. From the Operation list, select **print** and wait for the page to refresh. The print options for the print Web method are divided into three groups: **WSPrintOptions** This option provides the following options: • Specify page range. • Choose Java Printing. This parameter has to be set to TRUE. • Choose one of the available paper sizes on the target printer. These values can be retrieved by calling getPaperList and passing the printer name. • Select printOrientation {ORIENTATION_LANDSCAPE, ORIENTATION_AUTO} • Select printPageType {PAGES_ALL, PAGES_CURRENT, PAGES_RANGE} • Specify printer name. The available values can be retrieved by calling getPrinterNameList. • Flag indicating whether the blank pages should be skipped. • Flag indicating whether force all colors to black. • Specify the layers to print for specified pages. Page number, layer id and layer name are mandatory. If no layer information is provided, then the file's default layer settings define which layers to print. **WSPrintHeader** This option allows you to specify the text to be added to the header and footer of the printed page (left, right, and/or center). **WSPrintWaterMark** This option provides the following options: • Specify the text to be added as watermark to the printed page. • Select the orientation of the watermark {DIAGONAL, HORIZONTAL, VERTICAL}

Web Method	Description
packetPrint	Prints a group of documents (known as packets) one at a time, along with the auto-generated cover page and summary page. From the Operation list, select **packetPrint** and wait for the page to refresh. The following is the list of packetPrint input parameters: **URIs** A list of URIs that belong to same packet.(mandatory) **PacketID** A string representing the ID of the packet. (mandatory) **PacketIDLocation** One of the six possible locations for packetID to appear on every page. From the printoutLocation enum, it is a combination of (top/bottom) + (left/center/right). It is an optional input. The PacketID is printed on the cover page and summary page regardless. **FileIDLocation** One of the six possible locations for File ID (file number in the packet) to appear on every page. From the printoutLocation enum, it is a combination of (top/bottom) + (left/center/right). It is an optional input. **WSPacketPrintOptions** This option applies to all documents in the packet. It is an optional parameter and provides the following: • Choose Java Printing. If this option is not specified, then it is set to TRUE. Do not set this parameter to FALSE. • Choose one of the available paper sizes on the target printer. These values can be retrieved by calling getPaperList and passing the printer name. • Select printOrientation {ORIENTATION_LANDSCAPE, ORIENTATION_AUTO} • Specify printer name. The available values can be retrieved by calling getPrinterNameList. If this option is not specified, then the default printer is used. • Flag indicating whether force all colors to black (grayscaled). • Specify the layers to print for specified pages. Page number, layer id and layer name are mandatory. **WSPrintHeader** This option allows you to specify the text to be added to the header and footer of the printed page (left, right, and/or center). **WSPrintWaterMark** This option provides the following options: • Specify the text to be added as watermark to the printed page. • Select the orientation of the watermark {DIAGONAL, HORIZONTAL, VERTICAL} **openAllMarkups** A boolean flag that indicates if the markups of the document must be printed with the document.
getXrefs	This External References (XRefs) Web method returns a list of XRefs associated to a given file. From the Operation list, select **getXrefs** and wait for the page to refresh. This method only requires a valid URI. Authorization is needed only if the URI cannot be accessed without it.

Web Method	Description
getLayerInfo	This Web service which allows a user to obtain information about layers for every page of a document. It returns a list of pages and a list of layers for each page. From the Operation list, select **getLayerInfo** and wait for the page to refresh. This method only needs a valid URI. Authorization is needed only if the URI cannot be accessed without it.
getPartProperties	This part level metadata extraction Web method returns metadata for a given part in a given file. For example, in the case of a 3D assembly, this Web method returns properties of a particular part referenced by the 3D assembly. 1 From the Operation list, select **getPartProperties** and wait for the page to refresh. 2 This method needs a valid URI and a valid entityID. The valid entityIDs are retrieved by calling the getPartTree method and passing the same URI. Authorization is needed only if the URI cannot be accessed without it. 3 In the **pageNumber** field, enter a value less or equal to the number returned by getProperties.
getText	This text extraction Web method returns text contained in a given file. This method extracts all text present in 2D/EDA/Office files. Attributes are not extracted (for example, attributes in EDA designs). If a drawing contains XRefs, text is extracted from the XRefs as well. Text is extracted from visible layers. Markup text is not extracted. **Note:** Text extraction is not support on 3D pages. Additionally, the 3D page number is not displayed in the return value of getText. From the Operation list, select **getText** and wait for the page to refresh. This method only needs a valid URI. Authorization is needed only if the URI cannot be accessed without it.
getPaperList	This utility Web method returns the paper sizes for a given printer that are available to AutoVue. From the Operation list, select **getPaperList** and wait for the page to refresh. This method only needs a valid printer name. Valid printer names can be retrieved by calling getPrinterNameList.
getPrinterNameList	This utility Web method returns a list of available printers. From the Operation list, select **getPrinterNameList** and wait for the page to refresh. This method does not need an input parameter.

Web Method	Description
convert	This conversion Web method converts a given file into another format such as JPEG, PNG, PDF, or TIFF. It only supports one page at a time. From the Operation list, select **convert** and wait for the page to refresh. This method can be called without including the option section. In this case, the default options use the bitmap version of the document in its original size. If you set *openAllMarkups* to TRUE, AutoVue Web Services retrieves and includes all existing markups into the convert output. If you include **convertOption**, you can: • Specify the color depth value. • Select the output format {BMP, TIF, PDF, JPG, PNG} • Specify the page (only one page at a time is supported). Note that with PDF format, regardless of the page setting, all pages are converted together. • Select the *convert* scale {TYPE_SIZE, TYPE_SCALE} • Specify the *height* and *width* in pixels (if TYPE_SIZE Scale is selected). • Specify the *scaleFactor* and *stepsPerInch* (if TYPE_SCALE is selected). • Specify if it is a rendition to be saved back to the repository. If set to TRUE, then no convert data is returned to the caller and it is sent to the repository. • Select the *cameraView* {NONE, ISOMETRIC, TOP, BOTTOM, FRONT, BACK, LEFT, RIGHT}. A few notes to consider when using any **cameraView** other than NONE: • It only applies to 3D documents. • An illegal argument is thrown if: - An output format other than PNG is selected. - openAllMarkups is set to TRUE. - TYPE_SCALE is selected.
getProperties	This file level metadata extraction Web method returns metadata and properties for a given file. From the Operation list, select **getProperties** and wait for the page to refresh. This method only needs a valid URI. Authorization is needed only if the URI cannot be accessed without it.

AutoVue Web Service API

The **JavaDoc** index provides a complete reference to all classes and APIs inside the AutoVue Web Service package. The **com.oracle.autovue.services** package contains all classes and sub-packages of AutoVue Web Services. All the AutoVue Web methods are defined inside the **VueBeanWS** class of this package.

The sub-package **com.oracle.autovue.services.options** includes all classes that represent custom input options for different AutoVue Web methods such as convert and print.

The sub-package **com.oracle.autovue.services.types** includes all classes that represent custom outputs for different AutoVue Web methods such as getText, getXrefs, and so on.

The sub-package **com.oracle.autovue.services.pool** includes pooling mechanisms used inside the AutoVue Web Services package.

Appendix A - Sample Client Code in Java

The following sample client code in Java calls all of the AutoVue Web methods with a predefined URL.

```
import java.io.FileOutputStream;
import java.util.List;
import com.oracle.autovue.services.*;

public class AutoVueWSClient
{
public static void main(String[] args) throws Exception{
   //Create Service
   VueBeanWS_Service service = new VueBeanWS_Service();

   //Create proxy
   VueBeanWS proxy = service.getVueBeanWSPort();

   //Call AutoVue ping Web method.
   System.out.print (proxy.ping("hello") );

   String URI = "http://www.oracle.com/applications/autovue/autovue-electro-
   mechanical-professional-data-sheet.pdf";

   //Call the convert Web method.
   try{
      VueBeanWS_Service service = new VueBeanWS_Service();
      VueBeanWS proxy = service.getVueBeanWSPort();

      ConvertOption option = new ConvertOption();

      option.setFormat(Format.BMP);
      option.setPage(1);
      option.setScaleType(ScaleType.TYPE_SIZE);
      option.setHeight(600);
      option.setWidth(800);
      String URI = "server://@1/2d/AutoCAD/……";

      byte[] file = proxy.convert(URI, option, null, false);
      FileOutputStream fos = new FileOutputStream("c:/temp/output1.bmp");
      fos.write(file);
      fos.close();
}
    catch(Exception e){
      e.printStackTrace();
    }
```

```
    //Call the getPrinterNameList Web method.
    List<String> printers = proxy.getPrinterNameList();
    for (String printer : printers) {
      System.out.println("Printer Name: "+printer);
      System.out.println("Available Papers on this Printer");
      //Call the getPaperList Web method
      List<String> papers = proxy.getPaperList(printer);
        for (String paper : papers) {
        System.out.println("Paper Name: "+paper);
        }
      //Call the getProperties Web method.
      List<MetaProperty> properties = proxy.getProperties(URI, null);
      for (MetaProperty prop : properties) {
        System.out.println( prop.getName() + "=" +prop.getValue());

      //Call the getText Web method.
      List<SearchText> texts = proxy.getText(URI, null);
      for (SearchText text : texts ) {
        System.out.println("\nPage Number:"+ text.getPageNumber());
        List<String> txts = text.getTexts();
        for (String txt : txts) {
          System.out.print(txt);
        }
      }

      //Call the getXrefs Web method
      List<XrefsInfo> xrefs = proxy.getXrefs(URI, null);
      for (XrefsInfo xref : xrefs ) {
        System.out.println("Name:"+xref.getDocName() + " " + "docID:" + xref.ge
        DocID());
      }

      //Assuming URI is a 3D document. Call the getPartTree Web method.
      int pageNum = 4;
      PartTreeResult parts = proxy.getPartTree(URI,pageNum,null);

      //Call getParts Web method.
      List<PartInfo> info = parts.getParts();
      for (PartInfo part : info) {
        System.out.println("Part Name :"+part.getName() + " - Part ID:" +
        part.getID()+" - Part Type:" + part.getType());

        List<PartMetaProperty> metaProps = proxy.getPartProperties(URI, pageNum,
          part.getID(), null);
        for (PartMetaProperty meta : metaProps) {
          System.out.println( meta.getName() + "=" meta.getValue());
        }
      }
}
```

Web Services Sample Client Code for Printing

AutoVue Web Services provides a sample Web Services client code, SampleClient.java, which demonstrates how to call Web Services' print() method. It is located under the <AutoVue Web Services Installation Directory>\autovue_webservices directory. You can make the following modification according to your needs:

- Specify the username and password if the file has restricted access. For example, this is needed when storing a file in DMS.
- Specify more print options, watermark options, and header/footer options.
- If an error message containing the string ERROR_00 appears when the client calls the Web Services print() method, then the Web Services cannot process the request due to following reasons:
  - Server is too busy. No VueBean is available to process the request.
  - Not enough memory is available for a VueBean to open a file.

  To resolve this issue, the client must call the `print()` method later. In the SampleClient.java file, the client waits for one minute (60000 milliseconds) to call again.
- The following error messages appear when layer information is invalid:
  - ERROR_005: The page specified in LayersInfo object does not have layer.
  - ERROR_006: The layer information specified in LayersInfo object is not correct.
  - ERROR_007: There is no layer information in LayersInfo object for the specified page number.

Packet Printing

AutoVue Web Services provides a sample Web Services client code, SamplePacketPrintClient.java, which demonstrates how to call Web Services' `packetPrint()` method. The sample client code is located under <AutoVue Web Services Installation Directory>\autovue_webservices directory.

Note: In the `packetPrint()` method only the list of the documents and the packet ID are mandatory. Other parameters are optional. If you want the packetID to be printed on every page of all documents, then you must specify a print out location. Same is true for file counter (the file ID of each document in the packet is printed if a location is specified).

If no packet print option is defined or if no printer is set in that object, then the default printer on the AutoVue Web Services machine is used automatically.

The output of the `packetPrint()` method includes an auto-generated cover page at the beginning of the packet print out and a summary page at the end.The summary page includes the success/fail status of each document in the packet. For this reason, the `packetPrint()` method does not return until the last document in the packet is processed.

Feedback

If you have any questions or require support for AutoVue please contact your system administrator.

If at any time you have questions or concerns regarding AutoVue, please contact us.

General AutoVue Information

Telephone	+1.514.905.8434 or 1.800.363.5805
Web Site:	http://www.oracle.com/us/products/applications/autovue/index.html
Blog:	http://blogs.oracle.com/enterprisevisualization/

Oracle Customer Support

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

My Oracle Support AutoVue Community

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

Sales Inquiries

E-mail:	autovuesales_ww@oracle.com