



KODAK Image Access Client SDK Version 2.0 Developer's Guide

Part No. 6B7231
March 1, 2002

Eastman Kodak Company
343 State Street
Rochester, NY 14650

Kodak is a trademark of Eastman Kodak Company.

©Eastman Kodak Company, 2002

Table of Contents

1. Introduction.....	1
Overview.....	1
Document Purpose.....	2
References and Applicable Documents.....	2
Document Organization.....	3
2. Automatic SDK Installation.....	5
Supported Configurations.....	5
Software development environments.....	5
Hardware and software requirements.....	5
Installing the Client SDK on Windows.....	6
Enabling Software Installation.....	6
Verify Enabling Software Installation.....	6
Client Installation.....	7
Verify KODAK Image Access Client Installation.....	7
Verify Order Submission.....	8
Remove.....	8
3. Manual KODAK Image Access Client Library Installation and Usage (WINDOWS).....	9
Java Library Installation.....	9
Building Your Java KODAK Image Access Application.....	10
Running Your Java KODAK Image Access Application.....	10
Installing and Running the Test Tool.....	11
Native library Installation.....	11
Building Your Native KODAK Image Access Application.....	12
4. Manual KODAK Image Access Client Library Installation and Usage (SOLARIS).....	13
Java Library Installation.....	13
Building Your Java KODAK Image Access Application.....	14
Running Your Java KODAK Image Access Application.....	14
Installing and Running the Test Tool.....	15
5. Organization of KODAK Image Access Client Class Hierarchy....	17
Class Hierarchy.....	17

6. KODAK Image Access Client Class Information	19
KIASSystem	19
Description	19
Inner Class Summary	19
Constructor Detail	19
Method Detail	20
Session	22
Description	22
Inner Class Summary	22
Constructor Detail	22
Method Detail	23
Order	38
Description	38
Method Detail	38
ImageInfo	43
Description	43
Method Detail	43
OrderInfo	44
Description	44
Method Detail	44
OrderStatus	45
Description	45
Method Detail	45
KIASLocale	46
Description	46
Method Detail	46
Product	47
Description	47
Method Summary	47
Output	48
Description	48
Method Summary	48
User	48
Description	48
Method Summary	48
SessionInfo	49
Description	49
Method Detail	49
VersionInfo	49
Description	49
Method Summary	49
ClientLibException (KODAK Image Access Client SDK Error Codes)	50
Description	50
Method	50
Valid Throws	51
ClientJNIException (KIASClient.dll Error Codes)	53

Description	53
Methods inherited from JAVA language throwable.....	53
Field Detail	53
ClientLibSubmitException	55
Description	55
Method Summary	55
GetImagesException	55
Description	55
Method Summary	55
7. KODAK Image Access Client Date Encoding.....	57
Time Format	57
8. Native version of KODAK Image Access Client API.....	59
Native library version	59
Memory Allocation.....	59
Error Handling	60
Methods of Global Scope	61
Methods of KIAS System Scope	61
Methods of Session Scope.....	61
Methods of KIAS Order scope.....	63
Native library data types	65
9. Sample Code.....	69
JAVA Sample Code	69
Native Sample Code.....	74
10. Obtaining an X509 SSL Certificate for Your Kodak Image Access Client	83
Install Sun JSSE 1.0.2.....	83
Adding a trusted root certificate to your JRE	84
Generate a private / public key-pair for your server.....	85
Generate a CSR (Certificate Signing Request) for your CA (Certificate Authority).....	85
Submit your CSR and get back your certificate	85
Import the new certificate into your Keystore.....	86
Configure your Web Server	87
Parameter Key for Common Parameters	88
11. Server Status Codes.....	89
12. Glossary	93

1. Introduction

Overview

The KODAK Image Access Standard is a suite of API's that lets a client machine communicate with a KODAK Digital Lab System (DLS) server. A client submits photofinishing orders to the server as COS (Customer Order Specifications) files which may be constructed with the KODAK COS library. COS files contain product information, such as 4x6 print and the actual images. For more information on COS files, refer to the *KODAK Digital Lab System Software Developer's User Guide for COS File Creation for DLS, Customer Order Specification v1.11*, and *COS, SDK User's Guide v2.3*.

A client may also request images from orders that have been archived on the server. An archived order is simply a related group of images without product information.

The KODAK Image Access Client SDK is intended to let external clients send images and orders to the KODAK Digital Lab System (DLS), and for clients to query the DLS for product capabilities and order status using the KODAK Image Access Client application programming interfaces (APIs).

The KODAK Image Access Client SDK is intended for use by programmers developing either JAVA or Native (C language) custom applications that will interface with the KODAK DLS or derivative KODAK DLS products.

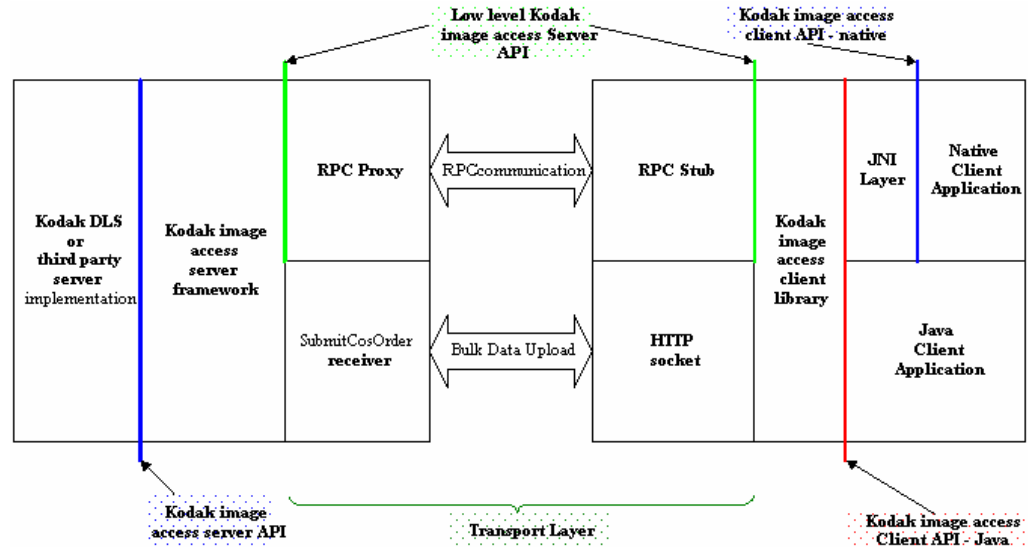


Figure 1: KODAK Image Access Standard Architecture Diagram

This document focuses on the Client API. The intended audience is a developer writing a JAVA or Native KODAK Image Access Client application. The Client Library talks to the server via the transport layer. The transport layer consists of two parts: the Remote Procedure Call (RPC) section and the bulk upload section. The bulk upload section consists of the Client Library directly performing an HTTP POST upon the server. The RPC section exposes the Server API to the Client Library via an RPC stub. The Client Library must be configured to use a specific RPC stub. Calling the RPC stub is equivalent to calling the Server directly - the two server API lines are the exact same interface. The RPC Proxy - RPC Stub pair are a middleware layer that hides the network communication details.

Document Purpose

The purpose of this document is to provide DLS external client developers with guidelines regarding the construction and transmission of orders for fulfillment on a DLS using the KODAK Image Access Standard.

References and Applicable Documents

The following documents are either referenced within this document or may prove useful during KIAS development:

- *Customer Order Specification*, Version 1.11, March 25, 1999.
- *Customer Order Specification SDK User's Guide*, Version 2.3, June 1999.
- *KODAK Digital Lab System Software Developer's User Guide for COS File Creation for DLS*.

Document Organization

This document is organized as follows:

Introduction: General information regarding KODAK Image Access Standard, document purpose, key references, and document organization.

SDK Installation: This section provides detailed information for JAVA Library installation as well as Native C/C++ installation.

Manual KODAK Image Access Client Library Installation: These installation instructions detail which files that are part of the KODAK Image Access Client library and how to set them up for a KODAK Image Access Client application. These instructions are intended for a developer writing an install program for a custom KODAK Image Access Client application.

Organization of KODAK Image Access Client Class Hierarchy: This section provides a grouping of KODAK Image Access Client methods by class hierarchy.

KODAK Image Access Client Class Information: This section describes the KODAK Image Access Client and its methods for JAVA developers. This is the library that provides methods for connecting to the DLS, uploading orders to the DLS, getting upload status, and getting archived images from the DLS.

KODAK Image Access Client Date Encoding: This section details how dates are used by the KODAK Image Access Client API.

Native Version of KODAK Image Access Client API: This section provides native library information for support of a client application using C++.

Sample Code: This section contains the JAVA Sample Code and Native Sample Code.

Obtaining an X509 SSL Certificate for your KODAK Image Access Client: This section provides information on enabling the Secure Socket Layer (SSL) to gain encrypted communications for your server.

Server Error Messages: This section lists the various status codes and messages that are returned by a server to its clients.

Glossary: This section contains the definitions of many terms used within this document.

2. Automatic SDK Installation

Supported Configurations

Software development environments

To begin developing your KODAK Image Access Client application, you must have knowledge of either JAVA or C/C++. You must also understand the DLS user interface (UI).

Hardware and software requirements

To install the KODAK Image Access Client SDK and KODAK Image Access SDK Test Tool, you must have the following minimal configuration:

- 64 MB RAM for WINDOWS (128 minimum to run sample application)
- 256 MB RAM for SUN SOLARIS 8 (SPARC Platform Edition)
- 16 MB of available disk space for JAVA Runtime Environment (JRE)
- 12 MB of available disk space for KODAK Image Access Client library installation
- Sufficient disk space available to store retrieved images of various sizes
- JRE 1.3.0
- CD-ROM drive
- WINDOWS 95 OSR25, WINDOWS 98 Second Edition, NT Workstation 4.0 with Service Pack 5, WINDOWS 2000 Professional with Service Pack 1, or SUN SOLARIS 8 (SPARC Platform Edition)

To develop and run your KODAK Image Access Client application, you must have these additional items:

- COS SDK version 2.6. This file can be found in the COS directory on the KODAK Image Access Client SDK Version 2.0 CD.
- Internet/LAN access to a server and DLS 2.0 or DLS Simulator.
- A JAVA development environment, such as JDK 1.3.0, or
or
A C++ development environment, such as Visual C++ 6.0

Installing the Client SDK on Windows

Note: For manual installation instructions, refer to Section 3 of this document.

Enabling Software Installation

Important: If a previous version of KODAK Image Access Enabling Software is already installed on your system, use the Add/Remove Program Utility from your WINDOWS Control Panel to remove it before continuing with this procedure. It is also possible to remove previous versions by running Setup.exe from the KODAK Image Access Enabling Software Version 2.0 CD.

1. Install JRE 1.3.0 from the KODAK Image Access Enabling Software for DLS Version 2.0 CD by double-clicking J2RE1_3_0-WIN.EXE.
2. Follow the instructions and complete the JRE installation.
3. Open the Disk1 folder from the KODAK Image Access Enabling Software for DLS Version 2.0 CD and double-click setup.exe to install the enabling software.

A message box displays saying the KODAK Image Access Enabling software installation is complete.

4. Restart the computer.

Verify Enabling Software Installation

Connect to the DLS Server with a Web browser. Use the URL http://your_server_IP:8080/soap/servlet/rpcrouter or <http://localhost:8080/soap/servlet/rpcrouter> if you are using a browser on the server. You should get back a page that says "Sorry, I don't speak via HTTP GET..."

Client Installation

Use the following procedure for automatic installation.

Important: If a previous version of KODAK Image Access Client SDK is already installed on your system, use the Add/Remove Program Utility from your WINDOWS Control Panel to remove it before continuing with this procedure. It is also possible to remove previous versions by running Setup.exe from the KODAK Image Access Client SDK Version 2.0 CD.

1. Install JRE 1.3.0 from the KODAK Image Access Client SDK Version 2.0 CD by double-clicking J2RE1_3_0-WIN.EXE.
2. Follow the instructions and complete the JRE installation.
3. Open the Disk1 folder on the Image Access Client SDK Version 2.0 CD. Select and double-click setup.exe to install the KODAK Image Access Client.
4. Verify that a message box is displayed stating the KODAK Image Access Client installation is complete.
5. Install the COS Library version 2.6 onto client computers using the KODAK Image Access Client Version 2.0 CD. The file path is D:\COS\cos_lib_opt.zip where D is the CD drive.

Verify KODAK Image Access Client Installation

Note: For more information on the KODAK Image Access SDK Test Tool, refer to the *KODAK Image Access SDK Version 2.0 Test Tool Guide*.

1. Double-click the file C:\Program Files\KODAK\KIAS\testtool.bat or use Start->Programs->KIAS->Start KIAS Test Driver. This launches the KODAK Image Access SDK Test Tool.
2. To make the KODAK Image Access SDK Test Tool is aware of the server, click **Settings** and add a line to the large edit box at the center of the dialog box. You may copy the line that is already present, replacing the string 'localhost' with the IP address of your server.
3. Within the KODAK Image Access SDK Test Tool, select the URL of your server from the pull-down menu.
4. Click **Update** under Server Version.

Verify Order Submission

Use the following steps to verify that orders can be submitted:

1. From your working KODAK Image Access SDK Test Tool, click **Connect**.
2. Enter a KODAK Image Access Client name / password. Your server should have a default account of pns / pns123. Use the name / password combination of KIASSU / KIMYXYZ to add the ability to call GetOrdersAdmin.
When you successfully connect, a new line is added to the KIAS Sessions table.
3. Select the Submissions tab.
4. Click **Submit** on the Submissions tab.
5. In the file selection dialog that appears, select a sample COS file. These files are found in the SampleCosfiles directory on the KODAK Image Access Client SDK Version 2.0 CD.
Your submission appears as a new entry in the KODAK Image Access Orders table on the Submissions tab.
6. Examine the Test Tool Order Status field for your order. The status changes from Submitting to Sent after the order has been processed.
7. On the server side, check the DLS WIP screen to see if the order arrived.

Remove

Option 1: Use the Add/Remove Program Utility from your WINDOWS Control Panel to remove KODAK Image Access Client SDK.

Option 2: Use the Setup.exe file.

1. Open the Disk1 folder from the KODAK Image Access Client SDK Version 2.0 CD and double-click setup.exe to remove the enabling software.
A message box displays saying that the KODAK Image Access Client is already installed.
2. Click **Yes** to remove the software.
A message box displays saying that the KODAK Image Access Client has been removed.
3. Click **OK** to complete.

3. Manual KODAK Image Access Client Library Installation and Usage (WINDOWS)

These instructions are intended for developers writing an installation program for a custom Image Access Client application using WINDOWS.

The main body of the KODAK Image Access Client library is a Java .jar file, KIASClientLib.jar. Whether development will be done in Java or using the Native Image Access Client API, the Java runtime must be installed before development can begin.

All users of the KODAK Image Access Client Library must follow the Java installation steps. Only those choosing to use the Native API need to follow the Native Installation steps.

Java Library Installation

1. Install the JRE (for a user) or JDK (for a developer)
 - a. If you are a developer, the JDK may have already been installed as part of your development environment. Verify the version by typing `java -version` at the command line of an MS-DOS window. Version 1.3.0 should be returned. If you already have version 1.3.0 installed, skip to step 2.
 - b. Install JRE 1.3.0 from the KODAK Image Access Client SDK CD by double-clicking J2RE1_3_0-WIN.EXE.
 - c. Follow the instructions and complete the JRE installation.
 - d. Type `java -version` at the command line of an MS-DOS window and press **Enter**. The system should return 1.3.0 if the installation was successful.
2. Install the KODAK Image Access Client Library
 - a. Create a new directory to hold the .jar files. For example: `C:\kias\jars`.
 - b. Copy the .jar files: `KIASClientLib.jar`, `xerces.jar`, `soap.jar`, `mail.jar` and `activation.jar` to the directory just created. The file path is `D:\BINARY` where D is the CD drive.
 - c. Remember the directory, as you will have to specify it as part of the classpath when running an application.
3. Install the COS Library version 2.6 onto client computers using the KODAK Image Access Client CD. The file path is `D:\COS\cos_lib_opt.zip` where D is the CD drive.

Building Your Java KODAK Image Access Application

1. Verify that your development environment uses JDK 1.3.0.
2. Add each of the .jar files installed in the prior section to your development environment's library / classpath.
3. At the top of your Java source files insert the line:
`import com.kodak.KIAS.clientlib.*;`
to import the KODAK Image Access Client API definitions.

Running Your Java KODAK Image Access Application

1. Create a batch file to start Java executing the application. Refer to the following command line as an example:

```
java {defines} {classpath} {class}
```

- a. If your application has to work over SSL, the {defines} section of the command line should be:

```
-Djava.protocol.handler.pkgs=com.sun.net.ssl.internal.www.protocol
```

- b. The {classpath} section of the java command should be:

```
-classpath path1;path2;...pathN (WINDOWS)
```

or (replace separating semicolons with colons)

```
-classpath path1:path2:...pathN (SOLARIS)
```

You must specify the path of each library used by the application and the path of the application itself. Your KODAK Image Access Application will require the paths to KIASClientLib.jar, soap.jar, mail.jar, activation.jar, xerces.jar, as well as that of your application, to be specified. For example, path1 would be C:\kias\jars\KIASClientLib.jar.

- c. The {class} section specifies the class name containing the 'main' method that needs to be executed. For example:
`com.mycompany.mypackage.myclass` Save your Java command line and then run it to start your application.

NOTE: For an example of a full command line, see the testtool.bat file included with the KODAK Image Access Client SDK.

2. For more details on the usage of the 'java' command, see the online help that comes with the JDK.

Installing and Running the Test Tool

The KODAK Image Access Client SDK comes with a Test Tool to exercise each API method. To install and run it:

1. Complete installation of the Java KODAK Image Access library.
2. Copy KiasTestTool.jar to C:\kias\jars. The file path is D:\BINARY\KiasTestTool.jar where D is the CD drive.
3. Copy the startup script, testtool.bat to a directory of your choice. The file path is D:\BINARY\testtool.bat where D is the CD drive.

Note: If you have installed your .jar files in locations other than the example, modify our startup script or write your own. Refer to the prior section for instructions on how to write a startup script.

4. Change to the directory specified in step 3 above, or make sure that it is in your path.
5. Execute the startup script.
6. Consult the *KODAK Image Access SDK Version 2.0 Test Tool Guide* for usage details.

Native library Installation

Be sure to complete the JAVA install before continuing with the native install.

NOTE: The native install is only required if you are not developing in JAVA.

1. Establish environment variables:

- a. Set KIASJVM to the location of your JRE shared library.

On the WINDOWS platform, this is most likely C:\jdk1.3\jre\bin\hotspot\jvm.dll. On WINDOWS, you may check your registry at HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Java Runtime Environment\1.3.0. The RuntimeLib key should hold the location of the shared library.

For example:

```
set KIASJVM=c:\jdk1.3\jre\bin\hotspot\jvm.dll
```

- b. Set KIAS_JARS_HOME to the location where you placed the .jar files from the Java install. Add a terminating directory separator.

For example:

```
set KIAS_JARS_HOME=c:\kias\jars\
```

2. Place the KODAK Image Access Client shared library in your path. This could be either the same directory as your application or the system directory. On WINDOWS, the KODAK Image Access Client shared library is named KIASClient.dll.

Building Your Native KODAK Image Access Application

1. At the top of your C language source code, be sure to put to put the line:

```
#include KIASClient.h
```

to import the KODAK Image Access Client API definitions.

2. Link your C language KODAK Image Access Client application against KIASClient.lib.

Note: When you distribute your Native application, be sure to install KIASClient.dll. The .h and the .lib files are not to be redistributed.

4. **Manual KODAK Image Access Client Library Installation and Usage (SOLARIS)**

These instructions are intended for developers writing an installation program for a custom Image Access Client application using SOLARIS.

The main body of the KODAK Image Access Client library is a Java .jar file, KIASClientLib.jar. Whether development will be done in Java or using the Native Image Access Client API, the Java runtime must be installed before development can begin.

All users of the KODAK Image Access Client Library must follow the Java installation steps.

Java Library Installation

1. Install the JRE (for a user) or JDK (for a developer)
 - a. If you are a developer, the JDK may have already been installed as part of your development environment. Verify the version by typing `java -version` at the command line of a terminal window. Version 1.3.0 should be returned. If you already have version 1.3.0 installed, skip to step 2.
 - b. Install JRE 1.3.0 from the KODAK Image Access Client SDK CD using the `README.sparc` and `j2re1_3_0-solsparc.bin` files in the UNIX directory.
 - c. Copy the `j2re1_3_0-solsparc.bin` file from the CD. Make sure the file is named `j2re1_3_0-solsparc.bin`, once copied from the CD.
 - d. Type `java -version` at the command line of a terminal window and press **Enter**. The system should return 1.3.0 if the installation was successful.
2. Install the KODAK Image Access Client Library
 - a. Create a new directory structure to hold the .jar files. For example:
`/export/home/software/kodak/kias/jars.`
 - b. Unpack the UNIX .tar file from the UNIX directory on the CD to the new directory created in Step 2a above. Use the command, `tar xvf /cdrom/<CDVolumenumber>/UNIX/install.tar`, where `CDVolumenumber` is the volume ID of the CD, for example `cdrom0`.

- c. Move all of the unpacked files from the binary directory to the new directory created in Step 2a above.
- d. Remember the directory, as you will have to specify it as part of the classpath when running an application.
- e. Unpack the COS Library from the UNIX directory of the CD. Use the command, `tar xvf /cdrom/<CDVolumenumber>/UNIX/cos_lib_opt.tar`, where CDVolumenumber is the volume ID of the CD, for example `cdrom0`.

Building Your Java KODAK Image Access Application

1. Verify that your development environment uses JDK 1.3.0.
2. Add each of the .jar files installed in the prior section to your development environment's library / classpath.
3. At the top of your Java source files insert the line:

```
import com.kodak.KIAS.clientlib.*;
```

to import the KODAK Image Access Client API definitions.

Running Your Java KODAK Image Access Application

1. Create a batch / shell script to start Java executing the application. Refer to the following command line as an example:

```
java {defines} {classpath} {class}
```

- a. If your application has to work over SSL, the {defines} section of the command line should look like:

```
-Djava.protocol.handler.pkgs=com.sun.net.ssl.internal.www.protocol
```

- b. The {classpath} section of the java command should be:

```
-classpath path1;path2;...pathN (WINDOWS)
```

or (replace separating semicolons with colons)

```
-classpath path1:path2:...pathN (SOLARIS)
```

You must specify the path of each library used by the application and the path of the application itself. Your KODAK Image Access Application will require the paths to KIASClientLib.jar, soap.jar, mail.jar, activation.jar, xerces.jar, as well as that of your application, to be specified. For example, path1 would be `c:\kias\jars\KIASClientLib.jar` (WINDOWS) or `/export/home/software/kodak/kias/jars/KIASClientLib.jar` (SOLARIS).

- c. The {class} section specifies the class name containing the 'main' method that needs to be executed. For example:
`com.mycompany.mypackage.myclass`

2. Save your Java command line and then run it to start your application. On Solaris you may wish to make the script executable. (chmod +x).

For an example of a full command line, see the testtool.bat and testtool.sh files included with the KODAK Image Access Client SDK. They start the KODAK Test Tool on WINDOWS and SOLARIS, respectively.

3. For more details on the usage of the 'java' command, see the online help that comes with the JDK.

Installing and Running the Test Tool

The KODAK Image Access Client SDK comes with a Test Tool to exercise each API method. To install and run it:

1. Complete installation of the Java KODAK Image Access library.
2. If you have installed your .jar files in locations other than the example, modify our startup script, testtool.sh, or write your own. Refer to the prior section for instructions on how to write a startup script.
3. Make sure that the directory where you have installed you .jar files is in your path.
4. Make the script executable (chmod +x testtool.sh).
5. Execute the startup script.
6. Consult the *KODAK Image Access SDK Version 2.0 Test Tool Guide* for usage details.

5. Organization of KODAK Image Access Client Class Hierarchy

Class Hierarchy

- class java.lang.Object
 - class com.kodak.kias.clientlib.[ImageInfo](#)
 - class com.kodak.kias.clientlib.[KIASLocale](#)
 - class com.kodak.kias.clientlib.[KIASSystem](#)
 - class com.kodak.kias.clientlib.[KIASSystem.Session](#)
 - class com.kodak.kias.clientlib.[KIASSystem.Session.Order](#) (implements java.lang.Comparable)
 - class com.kodak.kias.clientlib.[OrderInfo](#) (implements java.lang.Comparable)
 - class com.kodak.kias.clientlib.[OrderStatus](#)
 - class com.kodak.kias.clientlib.[Product](#)
 - class com.kodak.kias.clientlib.[SessionInfo](#)
 - class java.lang.Throwable (implements java.io.Serializable)
 - class java.lang.Exception
 - class com.kodak.kias.clientlib.[ClientLibException](#)
 - class com.kodak.kias.clientlib.[ClientJNIException](#)
 - class com.kodak.kias.clientlib.[ClientLibSubmitException](#)
 - class com.kodak.kias.clientlib.[GetImagesException](#)
 - class com.kodak.kias.clientlib.[VersionInfo](#)

6. **KODAK Image Access Client Class Information**

KIASSystem

Description

KIASSystem is the main class implementing the client side KODAK Image Access Client API. A KIASSystem represents a given KODAK Image Access Client system, identified by its URL. The KODAK Image Access Client methods contained in the top KIASSystem level are those that do not require any authentication. Please see the client JAVA sample source code provided in Section 8 of this document for complete examples.

Inner Class Summary

KIASSystem.Session

Session is an inner class representing a login session with the KIASSystem.

Constructor Detail

KIASSystem

```
public KIASSystem(java.lang.String url, java.lang.String dummy)
throws ClientLibException
```

The constructor initializes URL information for KODAK Image Access Client.

```
// Example
```

```
...
```

```
KIASSystem clientlib = new KIASSystem("url", "");
```

```
...
```

Parameters	Throws
<p>URL- The URL of the Server. This URL should refer to the RPC acceptor of the target server. For example, http://localhost/soap/servlet/rpcrouter.</p> <p>dummy - this parameter is not used, please pass a blank string, "".</p>	<p>ClientLibException - Possible failure codes are:</p> <p>C000100B - SOAPERR - SOAP error, check network connection to server.</p> <p>C0001011 - NOSESSION - invalid session data returned by server.</p>

Method Detail

AreICCProfilesSupported

public boolean AreICCProfilesSupported()
 throws ClientLibException

Retrieves the ICCProfile capability of the fulfillment system. AreICCProfilesSupported is one of two functions that do not require establishing a Session prior to being called. The boolean returned by this method indicates whether the server honors ICC profiles embedded within a submitted COS file.

```
// Example
...
KIASSystem clientlib = new KIASSystem("url", "");
...
boolean icc = clientlib.AreICCProfilesSupported();
...
```

Returns	Throws
<p>A boolean indicating ICC support.</p>	<p>ClientLibException - Possible failure codes are:</p> <p>C000100B - SOAPERR - SOAP error, check network connection to server.</p>

GetServerVersion

```
public VersionInfo GetServerVersion()
```

```
throws ClientLibException
```

Retrieves the version of the server. GetServerVersion is one of two functions that do not require establishing a session prior to being called.

```
// Example
```

```
...
```

```
KIASSystem clientlib = new KIASSystem("url", "");
```

```
...
```

```
VersionInfo vi = clientlib.GetServerVersion();
```

```
...
```

Returns	Throws
The version of the server.	ClientLibException - Possible failure codes are: C000100B - SOAPERR - SOAP error, check network connection to server.

Session

Description

Session is an inner class representing a login session with the KIASSystem. A Session represents the acceptance of a name/password pair by KODAK Image Access Client. The Disconnect() method must be called to destroy the session. If disconnect() is not called, the server will disconnect the session after a server-configurable time-out period. If a KODAK Image Access Client method is called upon a disconnected session, the server returns the error code 0xc0002102 - no such session.

Inner Class Summary

KIASSystem.Session.Order

Order is an inner class representing a specific order on the KODAK Image Access / fulfillment system.

Constructor Detail

KIASSystem.Session

```
public KIASSystem.Session(java.lang.String userName, java.lang.String
userPassword)
```

throws ClientLibException

This constructor is the 'Connect()' call and creates a session. A session is a cookie that indicates that the username and password have been authenticated.

// Example

```
...
KIASSystem clientlib = new KIASSystem("url", "");
...
Session session = clientlib.new Session("name", "pass");
...
```

Parameters	Throws
<p>userName - the user to be authenticated. It should be less than 128 characters.</p> <p>userPassword - the password for the user account. It should be less than 128 characters.</p>	<p>ClientLibException - Possible failure codes are:</p> <p>C000100B - SOAPERR - SOAP error, check network connection to server.</p> <p>C0001011 - NOSESSION - invalid session data returned by server.</p>

Method Detail**Disconnect**

```
public void Disconnect()
```

```
throws ClientLibException
```

Invalidates the internal sessionID. A new call to Connect() or Reconnect() will now be required to make further method calls. Calling Disconnect() during a file transfer to/from KODAK Image Access Client is permitted and halts the transfer. If the client application retains the ID for the file, the transfer may be resumed later.

```
// Example
```

```
...
```

```
KIASSystem clientlib = new KIASSystem("url", "");
```

```
...
```

```
Session session = clientlib.new Session("name", "pass");
```

```
...
```

```
session.Disconnect();
```

```
...
```

Returns	Throws
None	ClientLibException - Possible failure codes are: C000100B - SOAPERR - SOAP error, check network connection to server. C0001011 - NOSESSION - invalid session data returned by server.

GetAvailableLocales

public KIASLocale[] GetAvailableLocales
throws ClientLibException

Gets all of the locales directly supported by KODAK Image Access Client.

// Example

```
...  
KIASSystem clientlib = new KIASSystem("url", "");  
...  
Session session = clientlib.new Session("name", "pass");  
...  
KIASLocale[] locales = session.GetAvailableLocales();  
...
```

Returns	Throws
An array of supported locales. There is no sort order. This method should be called prior to calling SetLocale.	ClientLibException - Possible failure codes are: C000100B - SOAPERR - SOAP error, check network connection to server. C0001011 - NOSESSION - invalid session data returned by server.

GetAvailableProducts

public Product() GetAvailableProducts(KIASLocale locale)

throws ClientLibException

Retrieves an array of print products supported by KODAK Image Access Client. If a locale is specified, the returned product descriptions may be localized (if server supports). If the blank locale is specified, GetAvailableProducts(new KIASLocale());, the product description is in the default locale of the server.

// Example

```

...
KIASSystem clientlib = new KIASSystem("url", "");
...
Session session = clientlib.new Session("name", "pass");
...
Product[] products = session.GetAvailableProducts(new KIASLocale());
...

```

Parameters	Returns	Throws
locale - desired locale of the returned product descriptions.	An array of print products in no sort order.	ClientLibException - Possible failure codes are: C000100B - SOAPERR - SOAP error, check network connection to server. C0001011 - NOSESSION - invalid session data returned by server.

GetAvailableOutputs

public Output() GetAvailableOutputs(KIASLocale locale)

throws ClientLibException

Retrieves an array of outputs supported by KODAK Image Access Client. If a locale is specified, the returned output descriptions may be localized (if server supports). If the blank locale is specified, GetAvailableOutputs(new KIASLocale());, the output description is in the default locale of the server.

// Example

```

...
KIASSystem clientlib = new KIASSystem("url", "");
...
Session session = clientlib.new Session("name", "pass");
...
Output[] outputs = session.GetAvailableOutputs(new KIASLocale());
...
    
```

Parameters	Returns	Throws
locale - desired locale of the returned output descriptions.	An array of outputs in no sort order.	ClientLibException - Possible failure codes are: C000100B - SOAPERR - SOAP error, check network connection to server. C0001011 - NOSESSION - invalid session data returned by server.

GetLocale

```
public KIASLocale GetLocale()throws ClientLibException
```

Gets the currently set KIASlocale.

```
// Example
```

```
...
```

```
KIASSystem clientlib = new KIASSystem("url", "");
```

```
...
```

```
Session session = clientlib.new Session("name", "pass");
```

```
...
```

```
KIASLocale currentLocale = session.GetLocale();
```

```
...
```

Returns	Throws
The current locale.	ClientLibException - Possible failure codes are: C000100B - SOAPERR - SOAP error, check network connection to server. C0001011 - NOSESSION - invalid session data returned by server.

GetNewOrderID

public JAVA.lang.String GetNewOrderID() throws ClientLibException

Retrieves a new unique order ID from KODAK Image Access Client. Calling this method is optional; SubmitCosOrder() will also return the unique ID. Call this method if the ID is needed prior to calling SubmitCosOrder(). Once a new order ID is returned, it must be passed to SubmitCosOrder to prevent an orderID leak.

// Example

```
...
KIASSystem clientlib = new KIASSystem("url", "");
...
Session session = clientlib.new Session("name", "pass");
...
String orderID = session.GetNewOrderID();
...
```

Returns	Throws
A new unique OrderID.	ClientLibException - Possible failure codes are: C000100B - SOAPERR - SOAP error, check network connection to server. C0001011 - NOSESSION - invalid session data returned by server.

Get Orders

```
public KIASSystem.Session.Order[] GetOrders(java.lang.String customerID,  
java.lang.String customerPassword, int startDate, int endDate,  
int orderStatus, java.lang.String orderID)
```

throws ClientLibException

Retrieves an array of existing orders from the fulfillment system. This method is the first step in performing an Image Access. Those orders matching the specified selection criteria are returned. Except for customerID, the criteria are all optional. If no additional criteria are specified, all orders for the given customerID are returned. To retrieve orders for multiple customers, use GetOrdersAdmin. The orders are returned in order of increasing age. The criteria have an implied 'AND' between them.

To decline to specify a String, pass "".

To decline to specify orderStatus, pass OrderStatus .NOTFOUND.

To decline to specify a time, pass 0.

// Example

...

```
KIASSystem clientlib = new KIASSystem("url", "");
```

...

```
Session session = clientlib.new Session("name", "pass");
```

...

```
Order[] orders = session.GetOrders(  
    "fsname", "fspass", 0, 0, OrderStatus.NOTFOUND, "");
```

...

Parameters	Returns	Throws
<p>customerID - a customer name, as known by the fulfillment system. This is NOT the same as a KODAK Image Access Client account name used when a session is created. This parameter is required.</p> <p>customerPassword - a password to match the above customerID.</p> <p>startDate - the beginning of the desired date range of acceptable orders. startDate should be a KODAK Image Access Client date / time.</p> <p>endDate - similar to startDate, a KODAK Image Access Client date / time.</p> <p>orderStatus - the status of the order.</p> <p>orderID - a specific orderID.</p>	<p>An array of Orders, with the newest orders listed first.</p>	<p>ClientLibException - Possible failure codes are:</p> <p>C000100B - SOAPERR - SOAP error, check network connection to server.</p> <p>C0001011 - NOSESSION - invalid session data returned by server.</p> <p>C0001012 - NULLPARAM - KIAS server error.</p>

GetOrdersAdmin

```
public KIASSystem.Session.Order[] GetOrdersAdmin(int startDate,int
endDate, int orderStatus, java.lang.String orderID))
throws ClientLibException
```

The same as GetOrders except that customer name and password are not parameters and thus all orders, irrespective of CustomerID, are returned. This method requires that your account has the 'view other customers orders' privilege enabled on the server.

GetOrdersByDateRange

```
public KIASSystem.Session.Order[]
GetOrdersByDateRange(java.lang.String customerID,java.lang.String
customerPassword,int startDate,int endDate)
throws ClientLibException
```

Retrieves orders for a specific range of dates. Implemented as:

```
return GetOrders(customerID, customerPassword , startDate, endDate,
OrderStatus.NOTFOUND, "");
```

GetOrdersByOrderID

```
public KIASSystem.Session.Order[] GetOrdersByOrderID(java.lang.String
customerID,java.lang.String customerPassword,java.lang.String orderID)
throws ClientLibException
```

Retrieves orders for a specific orderID. Implemented as:
return GetOrders(customerID, customerPassword , 0, 0,
OrderStatus.NOTFOUND, orderID);

GetOrdersByStatus

```
public KIASSystem.Session.Order[] GetOrdersByStatus(java.lang.String
customerID,java.lang.String customerPassword,int orderStatus)
throws ClientLibException
```

Retrieves orders with a specific status. Implemented as:
return GetOrders(customerID, customerPassword , 0, 0, orderStatus, "");

GetOrderStatus

```
public OrderStatus GetOrderStatus(java.lang.String orderID)
throws ClientLibException
```

Retrieves an OrderStatus from the Server.

```
// Example
...
KIASSystem clientlib = new KIASSystem("url", "");
...
Session session = clientlib.new Session("name", "pass");
...
String orderID = session.SubmitCosOrder("COSpath", "");
...
OrderStatus orderStatus = session.GetOrderStatus(orderID);
...
```

Parameters	Returns	Throws
orderID - the ID of the order for which the status is being retrieved. This should be an orderID previously returned by SubmitCosOrder().	The current order status.	C000100B - SOAPERR - SOAP error, check network connection to server. C0001011 - NOSESSION - The session has been disconnected. C0001012 - NULLPARAM - order ID may not be null.

GetSessionInfo

public SessionInfo getSessionInfo()

throws ClientLibException

Return the SessionInfo. This method is only used if the default locale of KODAK Image Access Client is needed.

// Example

...

```
KIASSystem clientlib = new KIASSystem("url", "");
```

...

```
Session session = clientlib.new Session("name", "pass");
```

...

```
SessionInfo sessionInfo = session.getSessionInfo();
```

...

Returns	Throws
The sessionInfo structure associated with this session.	ClientLibException - Possible failure codes are: C0001011 - NOSESSION - the session has been disconnected.

GetTimeToComplete

```
public int GetTimeToComplete(java.lang.String orderID)
```

```
throws ClientLibException
```

Retrieves the time, in seconds, expected to take before order completion.

```
// Example
```

```
...
```

```
KIASSystem clientlib = new KIASSystem("url", "");
```

```
...
```

```
Session session = clientlib.new Session("name", "pass");
```

```
...
```

```
String orderID = session.SubmitCosOrder("COSpath", "");
```

```
...
```

```
int timeToComplete = session.GetTimeToComplete(orderID);
```

```
...
```

Returns	Throws
Time, in seconds.	ClientLibException - Possible failure codes are: C000100B - SOAPERR - SOAP error, check network connection to server. C0001011 - NOSESSION - the session has been disconnected.

Reconnect

```
public void Reconnect()  
throws ClientLibException
```

This is a convenience function for reconnecting after a disconnect without having to create a new Session object. It behaves the same as the constructor, but with no parameters. Orders that refer to this reconnected session are still valid.

```
// Example
```

```
...  
KIASSystem clientlib = new KIASSystem("url", "");  
...  
Session session = clientlib.new Session("name", "pass");  
...  
session.Disconnect(); // or an involuntary disconnection  
...  
session.Reconnect();  
...
```

SetLocale

public KIASLocale SetLocale(KIASLocale locale)

throws ClientLibException

Sets the KIASLocale to the passed in locale. Call GetAvailableLocales() prior to calling this, and use one of the retrieved locales as the parameter to SetLocale().

```
// Example
...
KIASSystem clientlib = new KIASSystem("url", "");
...
Session session = clientlib.new Session("name", "pass");
...
KIASLocale[] locales = session.GetAvailableLocales();
...
KIASLocale locale = session.SetLocale(locales[0]);
...
```

Parameters	Returns	Throws
locale - the desired KIAS locale.	The locale that was actually set. This is the best match for the requested locale, not necessarily an exact match.	C000100B - SOAPERR - SOAP error, check network connection to server. C0001011 - NOSESSION - the session has been disconnected. C0001012 - NULLPARAM - orderID may not be null. C0001023 - BADLANGUAGE - the language code must be exactly 2 characters. C0001024 - BADCOUNTRY - the country code must be exactly 3 characters. C0001025 - BADCURRENCY - the currency code must be exactly 3 characters

SubmitCosOrder

```
public java.lang.String SubmitCosOrder(java.lang.String COSFilePath,
java.lang.String orderID)
```

throws ClientLibException

Uploads the designated file to KODAK Image Access Client. This method is synchronous; however calling Disconnect() from another thread causes it to terminate immediately with an exception. SubmitCosOrder is declared synchronized so any simultaneous calls are serialized. SubmitCosOrder() is able to recover from a premature disconnection. See the following scenario:

Call SubmitCosOrder("myfile", null);

A disconnect happens, either via an intentional call to Disconnect() or by a communications error. A ClientLibSubmitException is thrown. This exception contains an orderID that the client application should remember.

Call SubmitCosOrder("myfile", orderID); where orderID is the unique file identifier remembered from the exception in the prior step.

SubmitCosOrder() automatically resumes from the point in the file where it left off.

// Example

```
...
KIASSystem clientlib = new KIASSystem("url", "");
...
Session session = clientlib.new Session("name", "pass");
...
String orderID = session.SubmitCosOrder("COSpath", ""); ...
```

Parameters	Returns	Throws
<p>COSFilePath - the full path of the COS file to be uploaded. The COS file is not examined by the client library. CosFilePath may be any valid. It should not be a URL.</p> <p>orderID - the orderID from a prior, partially sent upload. Set orderID to null on first attempt.</p>	<p>An orderID is returned that uniquely identifies the order that was submitted.</p>	<p>ClientLibException - Possible failure codes are:</p> <p>C0001006 - BADURL - Internal KIAS clientLib error.</p> <p>C000100A - CANTSEND - An order OrderID was obtained but the file transmission did not complete.</p> <p>C000100B - SOAPERR - SOAP error, check network connection to the server.</p> <p>C000100F - BADFILE - COSFilePath must be a readable file of nonzero size.</p> <p>C0001011 - NOSESSION - the session has been disconnected.</p> <p>C0001012 - NULLPARAM - orderID may not be null.</p>

ValidateFulfillmentSystemUser

public boolean ValidateFulfillmentSystemUser(User user)

throws ClientLibException

Return a boolean indicating whether the proposed user is acceptable to the server. This should be called prior to populating the order file with any user information. The user passed to this method represents the user information placed into the order file, it is distinct and separate from the customer login information that is passed to establish a session.

// Example

...

```
KIASSystem clientlib = new KIASSystem("url", "");
```

...

```
Session session = clientlib.new Session("name", "pass");
```

...

```
boolean userOK = session.ValidateFulfillmentSystemUser (
    new User("jdh3745", "billspass", "Bill Smith"
        , "301-123-4567", "123 3rd st. bowie md 20708",
        "bsmith@hotmail.com"));
```

...

Returns	Throws
a boolean indicating the acceptability of the passed in user.	<p>ClientLibException - Possible failure codes are:</p> <p>C000100B - SOAPERR - SOAP error, check network connection to server.</p> <p>C0001011 - NOSESSION - the session has been disconnected.</p>

Order

Description

Order is an inner class representing a specific order on the KODAK Image Access Client / fulfillment system. Orders may not be created directly (no public constructors are provided), but rather, they are returned by Session's GetOrders() method. An order is associated with a given session for all time. If a session that has associated orders becomes disconnected, do not call any of the order methods that require a connected session. A disconnection may be remedied by either calling the KIASSession's Reconnect() method, or by opening a new session and calling GetOrders() on the new session.

Method Detail

GetHighResImage

```
public void GetHighResImage(int imageID, java.lang.String destFile)
throws ClientLibException
```

Retrieves a full size image. Implemented as:

```
GetImage(0, destFile, imageID);
```

GetHighResImages

```
public void GetHighResImages(java.lang.String[] destFiles, int[] imageIDs)
throws ClientLibException
```

A convenience method for getting full size images. Implemented as:

```
GetImages(0, destFiles, imageIDs);
```

GetImage

public void GetImage(int resolution, java.lang.String destFile, int imageID)
throws ClientLibException

Downloads the single image specified. This method is similar to GetImages but does not overwrite the specified file. GetImage() examines the size of the target file and only downloads the portion of the image after this, appending it to the file. This is useful for resuming a download of a partially downloaded image. The partial download feature only works if the Server is HTTP 1.1 compliant, otherwise the entire file is downloaded.

// Example

...

```
KIASSystem clientlib = new KIASSystem("url", "");
```

...

```
Session session = clientlib.new Session("name", "pass");
```

...

```
Order[] orders = session.GetOrders(
    "fsname", "fspass", 0, 0, OrderStatus.NOTFOUND, "");
```

...

```
ImageInfo[] imageInfos = orders[0].GetImageRefs();
```

...

```
orders[0].GetImage(0, "temp.jpg", imageInfos[0].getImageID());
```

...

Parameters	Throws
<p>resolution - same as for GetImages(int, JAVA.lang.String[], int[]).</p> <p>destFile - the file where the retrieved image will be stored.</p> <p>imageID - identifies the image to be retrieved.</p>	<p>resolution - same as for GetImages(int, JAVA.lang.String[], int[])</p> <p>destFile - the file where the retrieved image will be stored.</p> <p>imageID - identifies the image to be retrieved.</p>

GetImageRefs

public ImageInfo[] GetImageRefs()

throws ClientLibException

Retrieves ImageInfo for each image in the order.

// Example

...

```
KIASSystem clientlib = new KIASSystem("url", "");
```

...

```
Session session = clientlib.new Session("name", "pass");
```

...

```
Order[] orders = session.GetOrders(  
    "fsname", "fspass", 0, 0, OrderStatus.NOTFOUND, "");
```

...

```
ImageInfo[] imageInfos = orders[0].GetImageRefs();
```

...

Returns	Throws
An array of ImageInfo's.	C000100B - SOAPERR - SOAP error, check network connection to server. C0001011 - NOSESSION - the session has been disconnected.

GetImages

public void GetImages(int resolution, java.lang.String[] destFiles,
int[] imageIDs)

throws ClientLibException

Downloads each image specified. If there is a problem midway through, a GetImagesException is thrown which contains the number of images successfully downloaded. Note: If the specified download file already exists, it is overwritten by the download.

```
// Example
...
KIASSystem clientlib = new KIASSystem("url", "");
...
Session session = clientlib.new Session("name", "pass");
...
Order[] orders = session.GetOrders(
    "fsname", "fspass", 0, 0, OrderStatus.NOTFOUND, "");
...
ImageInfo[] imageInfos = orders[0].GetImageRefs();
...
orders[0].GetImages(0, new String[] { "temp1.jpg" }
    , new int[] { imageInfos[0].getImageID() });
...
```

Parameters	Throws
resolution - the desired resolution of the largest dimension, in pixels. 0 denotes 'full size image' and -1 denotes 'thumbnail size image.' Negative values other than -1 are not permitted. Positive values will result in an image of the requested size being returned, up to the full size. destFiles - an array of the filenames where the images are saved. imageIDs - an array of the desired imageIDs; must be same size array as destFiles.	C0001006 - BADURL - KIAS Server returned invalid image URL. C000100B - SOAPERR - SOAP error, check network connection to server. C000100F - BADFILE - destFiles must be in an existing directory and be writable. C0001011 - NOSESSION - the session has been disconnected. C0001012 - NULLPARAM - destFiles and imageIDs may not be null. C000101A - IMAGENUM - KIAS Server error. C000101D - NOTRES - resolution must not be less than -1. C000101F - CANTGET - The image URL is inaccessible. C0001022 - ARRAYSIZE - destFiles and imageIDs must be of same nonzero length.

GetLowResImage

public void GetLowResImage(int imageID, java.lang.String destFile)

throws ClientLibException

Retrieves a thumbnail size image. Implemented as:

GetImage(-1, destFile, imageID);

GetLowResImages

public void GetLowResImages(java.lang.String[] destFiles, int[] imageIDs)

throws ClientLibException

Retrieves multiple thumbnail size images. Implemented as:

GetImages(-1, destFiles, imageIDs);

GetOrderInfo

public OrderInfo GetOrderInfo()

Returns the internal OrderInfo structure. The most important field of OrderInfo is OrderID.

// Example

...

KIASSystem clientlib = new KIASSystem("url", "");

...

Session session = clientlib.new Session("name", "pass");

...

Order[] orders = session.GetOrders(
 "fsname", "fspass", 0, 0, OrderStatus.NOTFOUND, "");

...

OrderInfoOrderInfo = orders[0].getOrderInfo();

...

Returns	Throws
An OrderInfo structure.	None

ImageInfo

Description

ImageInfo structures are returned by the `KIASSystem.Session.Order.GetImageRefs()` method. They contain the resolution and description of the image, along with an identifying imageID.

Method Detail

getDescription

```
public java.lang.String getDescription()
```

A human readable description of the frame. Most likely the per-frame back-print message, if any.

getImageID

```
public int getImageID()
```

A unique identifier relative to a specific order.

getNativeResolution

```
public int getNativeResolution()
```

The absolute size, in pixels, of the largest image dimension. May be 0 to denote unknown.

OrderInfo

Description

Each `KIASSystem.Session.Order` object contains an `OrderInfo` structure that stores basic information about the order. Call `KIASSystem.Session.Order.getOrderInfo()` to get an Order's `OrderInfo`.

Method Detail

getCustomerID

```
public java.lang.String getCustomerID( )
```

The customerID associated with this order.

getDateTime

```
public int getDateTime()
```

The returned orders are sorted in descending order based upon this KODAK Image Access Client date/time.

getDescription

```
public java.lang.String getDescription()
```

A human readable description of the order, if available. May be blank.

getImageCount

```
public int getImageCount()
```

The number of images associated with the order.

getOrderID

```
public java.lang.String getOrderID()
```

The unique order ID of the order. This order ID should be used when making calls to the `GetImage` family of methods rather than the `orderID` returned by `GetNewOrderID / SubmitOrder`. Only orders with `orderIDs` returned by `GetOrders` are guaranteed archived for later retrieval.

OrderStatus

Description

OrderStatus is a single integer with seven possible values or states.

State	Description
CANCELLED	The fulfillment system cancelled the order; possibly an operator cancellation.
COMPLETED	The fulfillment system successfully completed the order.
CREATED	The order is valid but nothing has happened to it yet. An order freshly returned by <code>KIASSystem.Session.GetNewOrderID()</code> should be in this state.
INPROCESS	The fulfillment system has the order and is not yet finished with it.
INVALID	The fulfillment system determined the order to be invalid. The order file was defective for some reason, perhaps a version issue between client and server.
NOTFOUND	Unable to locate the requested order. This most likely means that an invalid OrderID was used to query for status. It could also mean an older KIAS server that does not keep track of order status (DLS 1.1a).
RECEIVED	Received by the Server but not yet successfully forwarded to the order processing software.

Method Detail

GetOrderStatus

```
public int getOrderStatus( )
```

KIASLocale

Description

KIASLocale specifies a language-country-currency triple. It is used to specify the desired localization of ClientLibException messages and Product descriptions.

Method Detail

getCountry

The country is an ISO 3166, 2-character country code.

getCurrency

The currency is an ISO 4217, 3-character currency code.

getLanguage

The language is an ISO 639, 2-character language code.

Product

Description

Products are returned by the `KIASSystem.Session.GetAvailableProducts` (`com.kodak.kias.clientlib.KIASLocale`) method. Each product contains useful information about a print product that the server supports. The only mandatory fields within product are `productID` and `description`. Servers are not required to fill the other fields.

Method Summary

getDescription

A localized, human readable, description of the product.

getImageHeight

The height of the area to be printed with the image. This area may be smaller than the entire page.

getImageWidth

The width of the area to be printed with the image. This area may be smaller than the entire page.

getLengthUnit

The unit of measurement used by the other fields in `Product`. Values are inch and mm.

getLocale

Locale is specified if the product is intended to be locale specific.

getOptimalHres

The server may specify `optimalHres` to indicate the optimal height, in pixels, of a submitted image for this product. If an image of optimal size is submitted, the server should process it without resizing. This may save time and enhance image quality.

getOptimalWres

The server may specify `optimalWres` to indicate the optimal width, in pixels, of a submitted image for this product. If an image of optimal size is submitted, the server should process it without resizing. This may save time and enhance image quality.

getPageHeight

`PageHeight` is the total height of the print product.

`getPageWidth`

`PageWidth` is the total width of the print product.

getProductID

The `productID` is a unique (within the product world) identifier. The `productID` should be used when constructing an order file to specify which products one wants.

Output

Description

Output's are returned by the `KIASSystem.Session.GetAvailableOutputs` (`com.kodak.kias.clientlib.KIASLocale`) method. Each Output contains the `outputID` and description of an output that the server supports.

Method Summary

getDescription

A localized, human readable, description of the Output.

getOutputID

The `outputID` is a unique output identifier. The `outputID` should be used when constructing an order file to specify outputs.

User

Description

User represents end user information. It is much the same user information that is contained in an order file.

Method Summary

getAccountName

A unique account identifier.

getCustomerAddress

The customer's street address.

getCustomerEmail

The customer's email address.

getCustomerName

The customer's name.

getCustomerPhone

The customer's phone number.

getPassword

The password, if any.

SessionInfo

Description

SessionInfo contains information returned by a successful login. It is possible to examine the returned KIAS version, KIAS locale information, and the sessionID. The sessionID is unimportant to a JAVA user but must be remembered by a C language client using the JNI layer.

Method Detail

getLocale

The default and initial locale for this session.

getSessionID

A unique, per KODAK Image Access server, identifier for this session.

getVersion

The current version of the KODAK Image Access server. You should have already called GetServerVersion() if you care about the version.

VersionInfo

Description

VersionInfo is the version of the Server as returned by KIASSystem.GetServerVersion(). It is a sequence of four numbers, i.e., 1.0.1.3. The four numbers have the conventional names Major.Minor.Bug.Build. A VersionInfo is returned by connect() as well as by GetServerVersion().

Method Summary

getBug
getBuild
getMajor
getMinor

ClientLibException (KODAK Image Access Client SDK Error Codes)

Description

This is the base exception class thrown by KIASClient methods. All exceptions thrown by KODAK Image Access Client are derived from this exception. The base class `java.lang.Exception` provides a message string that the `ClientLibException` constructor fills in with a localized error message. In addition, an error code is added to the base class. The error code indicates the specific error, while the error message describes it in the local language. The KODAK Image Access Client error codes are each defined in this class.

Method

`getErrorCode`

Return the error code.

Valid Throws

Error Code	Error Name	Description
C0001022	ARRAYSIZE	Two arrays that should be the same size aren't.
C0001024	BADCOUNTRY	The Country code must be exactly two characters.
C0001025	BADCURRENCY	The Currency code must be exactly three characters.
C000100F	BADFILE	Unable to open file.
C0001023	BADLANGUAGE	The Language code must be exactly two characters.
C000102A	BADRESPONSE	Submitted order, an error HTTP response received from server.
C0001006	BADURL	Bad Connect URL.
C000101E	BIGFILE	The partial file is larger than the requested URL.
C000101F	CANTGET	Problem getting image file.
C000100A	CANTSEND	Unable to send to KODAK Image Access.
C0001028	CLASSLOAD	Unable to instantiate the requested KIASServer stub.
C000101A	IMAGENUM	The KODAK Image Access Server returned a different number of image URLs than was requested.
C000101C	NOPARTIAL	Expected but did not receive a 206 'partial content' from the Web server.
C0001029	NORESPONSE	Submitted order, but got no response from server.
C0001011	NOSESSION	No such session; perhaps Disconnect() was called.
C0001014	NOSLASH	A required forward slash was not found in the URL.
C000101D	NOTRES	The resolution was less than -1.

Error Code	Error Name	Description
C0001012	NULLPARAM	Client application passed a null parameter.
C000100B	SOAPERR	Internal SOAP error.
C000102B	SOAPFAULT	The server returned a fault. The fault string is appended to the exception string and should give some indication as to what went wrong. There are 2 likely causes. The first is that the server is not set up properly. If you know that the server generally is working, the likely cause is that the server does not implement the method that was called. This could be the case if the version of the server is older than that of the client. You must update the KIAS server software with a newer version if you wish to use the method that is failing.
C000102C	KCE	Communication error. Contact support.
C000102D	KIAS1_0	Your KIAS 2.0 client software is not compatible with a KIAS 1.0 server. Please upgrade the server to KIAS 2.0.
0	SUCCESS	The method completed successfully.

ClientJNIException (KIASClient.dll Error Codes)

Description

These error codes are thrown exclusively by the Client JNI layer. They cannot occur when using the JAVA KODAK Image Access Client library directly.

Methods inherited from JAVA language throwable

fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Field Detail

BADPOINTER

public static final int **BADPOINTER**

The pointer being KIASFree'd was not KODAK Image Access Client allocated. This error will occur if you call KIASFree on a pointer that is not an array allocated by KODAK Image Access Client. This error also occurs if you call KIASFree on the same pointer twice.

JNIERR

public static final int **JNIERR**

The JNI portion of the JRE threw an exception. Check that JRE version is 1.3.0 or greater. Otherwise, contact Kodak support.

JNIINIT

public static final int **JNIINIT**

Not all KODAK Image Access Client JAVA classes initialized. Check that your KIAS_JARS_HOME environment variable points to the location of your KIASClientLib.jar file. Make sure that it has a trailing slash. If that fails, you may have a version mismatch between KIASClientLib.jar and KIASClient.dll. You should make sure that the files came from the same distribution of the KODAK Image Access Client.

JVM

public static final int **JVM**

Unable to start JVM. Make sure that your KIASJVM environment variable points to a working JRE. The default value of this variable is:

C:\Program Files\javaSoft\JRE\1.3\bin\hotspot\jvm.dll

NOJNISESSION

public static final int **NOJNISESSION**

The sessionID passed does not exist. You may only use sessionIDs given to you by the Connect call. Other sessionIDs will not correspond to JAVA session objects.

NOORDER

public static final int **NOORDER**

The orderID passed does not exist. The orderID must be an orderID that was returned by GetOrders. Other orderIDs may not correspond to JAVA Order objects.

UNKNOWN

public static final int **UNKNOWN**

An unknown error, catch (...), occurred.

ClientLibSubmitException

Description

This exception is thrown only by `Session.SubmitCosOrder()`. It contains the `sequenceID` that would have been returned by `SubmitCosOrder()` had it returned normally. It is important that this ID be retained so that the file that was being transmitted when the exception occurred may be resent without starting at the beginning. For the details, see `SubmitCosOrder()`.

See Also

`KIASSystem.Session.SubmitCosOrder(String, String)`

Method Summary

`getSequenceId()`

Get the `sequenceID` that `SubmitCosOrder()` would have returned if it had returned normally.

GetImagesException

Description

This exception is only thrown by the `GetImages` call. It is thrown if all of the images could not be retrieved. If this exception occurs: all `imageCount` images were successfully retrieved by `GetImages()`, the `imageCount` image may have been partially retrieved, and any subsequent images were not retrieved at all. `GetImages` quits as soon as there is a problem.

Method Summary

`getImageCount()`

Get the number of images that `GetImages` successfully retrieved.

7. KODAK Image Access Client Date Encoding

The KODAK Image Access Client SDK uses dates in two related places:

- GetOrders() method used a date range as a parameter to the GetOrders() method.
- The OrderInfo structure that is part of the returned Order contains a date for that order. The returned date and time are dependent upon the server implementation. One server may choose to use 'time archived' while another could use 'time submitted.' Regardless of the time defined by the server, two orders submitted a day apart have times a day apart. No expectation of precision should be presumed as some servers may only give the time to the nearest hour while others are accurate to the second.

Time Format

The format of the time is as a C language time_t. A time_t is a 4 byte signed integer containing the number of seconds from January 1, 1970. JAVA times are similar to time_t's, a JAVA time is an 8 byte signed integer (a long in JAVA) containing the number of milliseconds from January 1, 1970. Conversion to and from time_t <-> JAVA times is done by multiplying or dividing by 1000. Note - if your time zone is not GMT, a time_t of 0 will be displayed as one of the last hours of 1969. A time_t is limited to times from the year 1901 through the year 2038.

Here is an example of how to convert to and from `time_t` <-> `long`.

```
public static long time_t2long(int time_t) {
    long jtime = (long)time_t;
    jtime *= 1000; // Seconds to mSec
    return jtime;
}

public static int long2time_t(long jtime) {
    jtime /= 1000; // mSec to seconds
    if (jtime > (long)Integer.MAX_VALUE) {

        // this isn't really satisfactory, but if the JAVA time
        // is greater than
        // 2038 we're going to have to truncate it to 2038
        jtime = (long)Integer.MAX_VALUE;
    }
    if (jtime < (long)Integer.MIN_VALUE) {

        // likewise for small times, bump up to 1901
        jtime = (long)Integer.MIN_VALUE;
    }
    return (int)jtime;
}
```

Once you have a long JAVA time, use this as a parameter to set a JAVA Calendar object. A Calendar object provides useful time/date formatting methods for displaying the time to a user.

```
import JAVA.util.Calendar;
import JAVA.util.Date;
```

```
Calendar time = Calendar.getInstance();
time.setTime(new Date(time_t2long(time_t)));
```

If you are using the Native library, cast back and forth from `KIASINT` to `time_t`.

```
time_t time = (time_t)myKIASINT;
```

8. ***Native version of KODAK Image Access Client API***

Native library version

Memory Allocation

The user of the native API must be concerned with the allocation of memory. The JAVA user has no such concerns. There are several rules to follow.

- All strings returned by the KODAK Image Access Client library are of fixed length and must be allocated by the caller. The following #defines define the lengths of all strings used in the API.

```
#define IDSIZE 64 // length of sessionID, orderID, and productID
#define DESCsize 128 // length of image and order descriptions
#define ERRsize 128 // length of the returned error message
#define PATHsize 1024 // length of image pathnames
```

For example, the error string return should be pre-allocated to ERRSIZE. You may use any allocation/deallocation mechanism. Here, the string is just placed on the stack.

```
...
wchar_t errorMessage[ERRsize]; // static allocation of string upon the stack
KIASINT retVal = KIASDisconnect(errorMessage, sessionID);
if (0 != retVal) wprintf(L"Error Disconnecting - %s", errorMessage);
...
```

- Any structure parameters to API methods must be allocated by the caller. Any allocation method may be used. Here dynamic allocation with new/delete is performed.

```
...
KIASLocale *pOutLocale = new KIASLocale();
KIASINT retVal = KIASGetLocale(errorMessage, sessionID, pOutLocale);
if (0 != retVal) wprintf(L"Error Getting Locale - %s", errorMessage);
...
delete pOutLocale;
...
```

- Several methods return variable sized arrays. For each of these, the library shall allocate the memory required by the array. The array must be freed by the client to prevent a memory leak. This must be done by calling the KIASFree function provided for this purpose. This is the only method in the native KIAS API that does not have an analogue in the JAVA version of the API.

```
KIASProduct *pProducts = 0;
KIASINT NumProducts;
KIASINT retVal = KIASGetAvailableProducts(errMessage,
    sessionId, &NumProducts, &pProducts);

if (0 != retVal) wprintf(L"Error Getting Products - %s",
    errMessage);

...
KIASFree(errMessage, pProducts);
pProducts = 0;
...
```

Error Handling

The JAVA KIAS Library throws a variety of exceptions. The native library translates these exceptions into (error code / error message) pairs. Each native API method, without exception, has with the following signature:

```
KIASINT methodName(wchar_t *pError, other parameters);
```

The error code is returned in the return value and the error message is returned in the first parameter. An error code of 0 denotes success. See the main JAVA documentation for discussion of which exceptions each method may throw. In addition to the exceptions that may be thrown by the JAVA library, there are errors that may happen at the native library level. A set of error codes has been set aside for those errors. The possible native library specific errors are:

```
// A null pointer was passed to the library as a parameter
#define NULLPARAM 0xc0001012

// The JNI library returned an error
#define JNIERR 0xc0001015

// the *pSessionID passed in is not valid.
#define NOSESSION 0xc0001016

// Not all JAVA classes initialized, check KIAS_JARS_HOME
// environment variable
#define JNIINIT 0xc0001017

// Unable to start JVM, check KIASJVM environment variable
// , verify JRE is properly installed
#define JVM 0xc0001018
```

```
// some unknown exception
#define UNKNOWN 0xc0001019
```

```
// the pointer passed to KIASFree is invalid
#define BADPOINTER 0xc0001020
```

```
// an invalid orderID was passed
#define NOORDER 0xc0001021
```

Methods of Global Scope

KIASFree is the only 'Global' method - not associated with a specific KODAK Image Access Client system, session, or order. It has no counterpart in the JAVA API.

```
KIASINT KIASFree( wchar_t *pError, void* array );
```

Methods of KIAS System Scope

These methods are called upon a specific KODAK Image Access Client system, specified as a parameter. They correspond to the methods of the JAVA class KIASSystem.

```
KIASINT KIASConnect( wchar_t *pError, KIASLoginInfo LI
, SessionInfo *pSI );
```

```
KIASINT KIASGetServerVersion( wchar_t *pError, wchar_t *pUrl
, wchar_t *dummy, KIASVersionInfo *pVI );
```

```
KIASINT KIASAreICCProfilesSupported( wchar_t *pError
, wchar_t *pUrl, wchar_t *dummy
, KIASINT *pbAreSupported );
```

Methods of Session Scope

These methods are called upon a specific session. Each these methods takes a second parameter of *pSessionID, which is essentially the 'this' pointer of a session object. These methods correspond to the methods of the JAVA class KIASSystem.Session. The native library maintains a table of valid pSessionIDs and will throw NOSESSION if the passed in ID is not found in the table.

```
KIASINT KIASDisconnect( wchar_t *pError, wchar_t *pSessionID );
```

```
KIASINT KIASGetOrderStatus( wchar_t *pError
, wchar_t *pSessionID, wchar_t *pOrderID
, OrderStatus *pOS );
```

```
KIASINT KIASSubmitCosOrder( wchar_t *pError
, wchar_t *pSessionID, wchar_t *pCosFilePath
```

```
, wchar_t *pOrderIDIn, wchar_t *pOrderIDOut );

KIASINT KIASReconnect( wchar_t *pError, wchar_t *pSessionID
, SessionInfo *pSI );

KIASINT KIASGetNewOrderID( wchar_t *pError
, wchar_t *pSessionID, wchar_t *pOrderIDOut );

KIASINT KIASSetLocale( wchar_t *pError, wchar_t *pSessionID
, KIASLocale Locale, KIASLocale *pLocaleOut );

KIASINT KIASGetLocale( wchar_t *pError, wchar_t *pSessionID
, KIASLocale *pLocaleOut );

KIASINT KIASGetAvailableLocales( wchar_t *pError
, wchar_t *pSessionID, KIASINT *NumReturned
, KIASLocale **pLocalesOut );

KIASINT KIASGetAvailableProducts( wchar_t *pError
, wchar_t *pSessionID, KIASLocale Locale
, KIASINT* NumReturned, KIASProduct **pProductsOut );

KIASINT KIASGetAvailableOutputs( wchar_t *pError
, wchar_t *pSessionID, KIASLocale Locale
, KIASINT* NumReturned, KIASOutput **pOutputsOut );

KIASINT KIASValidateUser ( wchar_t *pError
, wchar_t *pSessionID, KIASUser User
, KIASINT *pbUserOK);

KIASINT KIASGetOrders( wchar_t *pError, wchar_t *pSessionID
, wchar_t *pCustomerID, wchar_t *pCustomerPassword
, KIASINT StartDate, KIASINT EndDate
, KIASINT OrderStatus, wchar_t *pOrderID
, KIASINT* pNumReturned, OrderInfo **pOrdersOut );

KIASINT KIASGetOrdersByCustomer( wchar_t *pError
, wchar_t *pSessionID, wchar_t *pCustomerID
, wchar_t *pCustomerPassword, KIASINT* pNumReturned
, OrderInfo **pOrdersOut );
```

```
CIASINT KIASGetOrdersByOrderID( wchar_t *pError
    , wchar_t *pSessionID, wchar_t *pCustomerID
    , wchar_t *pCustomerPassword, wchar_t *pOrderID
    , CIASINT* pNumReturned, OrderInfo **pOrdersOut );
```

```
CIASINT KIASGetOrdersByDateRange( wchar_t *pError
    , wchar_t *pSessionID, wchar_t *pCustomerID
    , wchar_t *pCustomerPassword, CIASINT StartDate
    , CIASINT EndDate, CIASINT* pNumReturned
    , OrderInfo **pOrdersOut );
```

```
CIASINT KIASGetOrdersByStatus( wchar_t *pError
    , wchar_t *pSessionID, wchar_t *pCustomerID
    , wchar_t *pCustomerPassword, CIASINT OrderStatus
    , CIASINT* pNumReturned, OrderInfo **pOrdersOut );
```

Methods of KIAS Order scope

These methods operate on a specific KODAK Image Access Client order, as returned by GetOrders(). They correspond to the methods of JAVA class `KIASSystem.Session.Order`. Each method requires a third parameter of `*pOrderID`, which corresponds to the 'this' pointer of an order object.

```
CIASINT KIASGetImageRefs( wchar_t *pError, wchar_t *pSessionID
    , wchar_t *pOrderID, CIASINT *pNumReturned
    , KIASImageInfo **pImages );
```

```
CIASINT KIASGetImages( wchar_t *pError, wchar_t *pSessionID
    , wchar_t *pOrderID, CIASINT resolution
    , CIASINT NumRequested, wchar_t **destFiles
    , CIASINT *imageIDs );
```

```
CIASINT KIASGetImage( wchar_t *pError, wchar_t *pSessionID
    , wchar_t *pOrderID, CIASINT resolution
    , wchar_t *destFile, CIASINT imageID );
```

```
CIASINT KIASGetLowResImage( wchar_t *pError
    , wchar_t *pSessionID, wchar_t *pOrderID
    , CIASINT ImageID, wchar_t *destFile );
```

```
CIASINT KIASGetHighResImage( wchar_t *pError
    , wchar_t *pSessionID, wchar_t *pOrderID
    , CIASINT ImageID, wchar_t *destFile );
```

```
KIASINT KIASGetLowResImages( wchar_t *pError  
    , wchar_t *pSessionID, wchar_t *pOrderID  
    , KIASINT ImageID, wchar_t *destFile );
```

```
KIASINT KIASGetHighResImages( wchar_t *pError  
    , wchar_t *pSessionID, wchar_t *pOrderID  
    , KIASINT ImageID, wchar_t *destFile );
```

Native library data types

KIASINT is typedefed to be a long. It is intended to be the same as a jint, as defined in the JAVA SDK's jni.h. Since jni.h is not required to use the KODAK Image Access Client API, we have defined KIASINT. It is a standard signed 2's complement 4 byte integer.

KIASFLOAT is typedefed to float. Like KIASINT, is a redefinition of jfloat from jni.h. It is a standard IEEE single precision floating point number.

wchar_t is the standard C library wide character type. It is declared in the standard C library <wchar.h>. The KIAS API uses UNICODE, UTF16 - no BOM. Throughout the API, there are no char's, only wchar_t's. As with traditional C language char strings, wchar_t strings are NULL terminated.

The below structures correspond to classes in the JAVA API.

```
typedef struct tagKIASLoginInfo {
    wchar_t *pLoginUserName;
    wchar_t *pPassword;
    wchar_t *pUrl;
    wchar_t *pProxyUrl;
    wchar_t *pReserved1;
    wchar_t *pReserved2;
    KIASINT  Reserved3;
    KIASINT  Reserved4;
} KIASLoginInfo;
```

```
typedef struct tagKIASVersionInfo {
    KIASINT nMajor;
    KIASINT nMinor;
    KIASINT nBug;
    KIASINT nBuild;
} KIASVersionInfo;
```

```
typedef struct tagKIASLocale {
    wchar_t Language[3];
    wchar_t Country[3];
    wchar_t Currency[4];
} KIASLocale;
```

```
typedef struct tagSessionInfo {
    wchar_t SessionId[IDSIZE];
    KIASVersionInfo Version;
    KIASLocale Locale;
```

```
    } SessionInfo;

typedef struct tagOrderStatus {
    KIASINT Status;
} OrderStatus;

typedef struct tagOrderInfo {
    wchar_t OrderID[IDSIZE];
    wchar_t Description[DESCSIZE];
    KIASINT DateTime; // time_t
    KIASINT ImageCount;
} OrderInfo;

typedef struct tagKIASImageInfo {
    KIASINT ImageID;
    wchar_t Description[DESCSIZE];
    KIASINT NativeResolution;
} KIASImageInfo;

typedef struct tagKIASImageRequest {
    KIASINT ImageID;
    KIASINT Resolution;
    wchar_t DestinationFileName[PATHSIZE];
} KIASImageRequest;

typedef struct tagKIASProduct {
    wchar_t ProductID[IDSIZE];
    wchar_t Description[DESCSIZE];
    KIASFLOAT PageWidth;
    KIASFLOAT PageHeight;
    KIASFLOAT ImageWidth;
    KIASFLOAT ImageHeight;
    KIASINT LengthUnit;
    KIASLocale Locale;
    KIASINT OptimalWres;
    KIASINT OptimalHres;
} KIASProduct

typedef struct tagKIASOutput {
    wchar_t OutputID[IDSIZE];
    wchar_t Description[DESCSIZE];
```

```
} KIASOutput;

typedef struct tagKIASUser {
    wchar_t  AccountName[IDSIZE];
    wchar_t  Password[IDSIZE];
    wchar_t  CustomerName[DESCSIZE];
    wchar_t  CustomerPhone[DESCSIZE];
    wchar_t  CustomerAddress[DESCSIZE];
    wchar_t  CustomerEmail[IDSIZE];
} KIASUser;
```


9. Sample Code

JAVA Sample Code

```
package com.kodak.kias.Example;

import com.kodak.kias.clientlib.*;
import com.kodak.kias.clientlib.KIASSystem.*;

import java.io.File;

/**
 * Title:          Example Kodak Image Access client application
 * Description:    A simple, command line, example use of the client API
 * Copyright:     Copyright (c) 2002
 * Company:       Eastman Kodak
 * @version 1.0
 */

public class Example {

    public Example() { }

    // print out the order structure as received by each of the GetOrdersXXX
    // calls
    private static void PrintOrder(OrderInfo OI) {
        System.out.println("\t"
            + OI.getOrderID() + " "
            + OI.getDescription() + " "
            + OI.getDateTime() + " "
            + OI.getImageCount() + " "
            + OI.getCustomerID()
        );
    }

    // test the Client API
    // calls each API method
    public static void main(String[] args) {
        String Url = "http://localhost:8080/soap/servlet/rpcrouter";
        String Name      = "";
        String Pass      = "";
        String submitFile = "";
        String custID    = "";
        String reqOrderID = "";
        String pathGetImages = "c:\\\\GetImages.jpg";
        String pathGetImage = "c:\\\\GetImage.jpg";
        String pathGetLowResImage = "c:\\\\GetLowResImage.jpg";
        String pathGetHighResImage = "c:\\\\GetHighResImage.jpg";
    }
}
```

```
String pathGetLowResImages = "c:\\GetLowResImages.jpg";
String pathGetHighResImages = "c:\\GetHighResImages.jpg";

System.out.println("Kodak Image Access example version 2.3.0.0");
if (args.length != 6 && args.length != 5) {
    System.out.println(
        "\nUsage:\nexample cosfile Url name pass custID orderID\n" +
        "cosfile - the path of the COS file to submit\n" +
        "Url      - the url of the server\n" +
        "name     - the name passed to the Connect call\n" +
        "pass    - the password passed to the Connect call\n" +
        "custID  - the customer ID specified to GetOrders\n" +
        "orderID - optional - the orderID specified to GetImageRefs\n" +
        "if orderID is not specified, first one retrieved by " +
        "GetOrders is used"
    );
}

return;
}
if (args.length > 0) submitFile = args[0];
if (args.length > 1) Url       = args[1];
if (args.length > 2) Name      = args[2];
if (args.length > 3) Pass      = args[3];
if (args.length > 4) custID    = args[4];
if (args.length > 5) reqOrderID = args[5];

System.out.println("\nSending " + submitFile + " to " + Url);

try {

    KIASSystem clientlib = new KIASSystem(Url, "");

    // ***** call the 2 methods that do not require a prior Connect

    // getversion
    VersionInfo vi = clientlib.GetServerVersion();
    System.out.println("GetServerVersion() returns:");
    System.out.println("\t" + vi);

    // getICC
    boolean icc = clientlib.AreICCProfilesSupported();
    System.out.println("AreICCProfilesSupported() returns:");
    System.out.println("\t" + icc);

    // ***** move on to Connect

    // connect
    Session session = clientlib.new Session(Name, Pass);

    SessionInfo si = session.getSessionInfo();
    System.out.println("Session constructor succeeded");
    System.out.println("getSessionInfo() returns:");
    System.out.println("\t" + si);

    // GetNewOrderID
    String orderID = session.GetNewOrderID();
    System.out.println("GetNewOrderID() returns:");
    System.out.println("\t" + orderID);

    // disconnect
    session.Disconnect();
    System.out.println("Disconnect() succeeded");
```

```
// Reconnect
session.Reconnect();
System.out.println("Reconnect() succeeded");

// submit
String orderID2 = session.SubmitCosOrder(submitFile, orderID);
System.out.println("SubmitCosOrder() returns:");
System.out.println("\t" + orderID2);

// getstatus
OrderStatus st = session.GetOrderStatus(orderID2);
System.out.println("GetOrderStatus() returns:");
System.out.println("\t" + st);

// GetTimeToComplete
int ttc = session.GetTimeToComplete(orderID2);
System.out.println("GetTimeToComplete() returns:");
System.out.println("\t" + ttc);

// GetLocale
KIASLocale locale = session.GetLocale();
System.out.println("GetLocale() returns:");
System.out.println("\t" + locale);

// GetAvailableLocales
KIASLocale[] locales = session.GetAvailableLocales();
System.out.println("GetAvailableLocales() returns:");
for (int localei = 0; localei < locales.length; localei++) {
    System.out.println("\t" + locales[localei]);
}

// SetLocale
locale = session.SetLocale(new KIASLocale("JA", "JP", "JAY"));
System.out.println("SetLocale() returns:");
System.out.println("\t" + locale);

// ValidateFulfillmentSystemUser
boolean fsu = session.ValidateFulfillmentSystemUser(
    new User("sdjhfd39", "billspass", "bill smith"
        , "301-234-5678", "123 3rd st. bowie md 20708"
        , "bsmith@hotmail.com"));
System.out.println("ValidateFulfillmentSystemUser() returns:");
System.out.println("\t" + fsu);

// GetAvailableProducts
Product[] products = session.GetAvailableProducts(new KIASLocale());
System.out.println("GetAvailableProducts() returns:");
for (int producti = 0; producti < products.length; producti++) {
    System.out.println("\t" +
        products[producti].getProductID() + " " +
        products[producti].getDescription() + " " +
        products[producti].getImageWidth() + " " +
        products[producti].getImageHeight() + " " +
        products[producti].getLocale() + " " +
        products[producti].getPageWidth() + " " +
        products[producti].getPageHeight() + " " +
        products[producti].getLengthUnit() + " " +
        products[producti].getOptimalWres() + " " +
        products[producti].getOptimalHres()
    );
}

// GetAvailableOutputs
```

```
Output[] outputs = session.GetAvailableOutputs(new KIASLocale());
System.out.println("GetAvailableOutputs() returns:");
for (int outputi = 0; outputi < outputs.length; outputi++) {
    System.out.println("\t" +
        outputs[outputi].getOutputID() + " " +
        outputs[outputi].getDescription()
    );
}

// GetOrders
Session.Order[] orders = session.GetOrders(custID, "", 0, 0, 0, "");
System.out.println("GetOrders() returns:");
for (int orderi = 0; orderi < orders.length; orderi++) {
    PrintOrder(orders[orderi].getOrderInfo());
}

// GetImageRefs
ImageInfo[] imageInfos = orders[0].GetImageRefs();
System.out.println("GetImageRefs() returns:");
for (int imageInfoi = 0; imageInfoi < imageInfos.length; imageInfoi++)
{
    System.out.println("\t" +
        imageInfos[imageInfoi].getImageID() + " " +
        imageInfos[imageInfoi].getDescription() + " " +
        imageInfos[imageInfoi].getNativeResolution());
}

// remove temporary images from previous run
File fff = new File(pathGetImages);
fff.delete();
fff = new File(pathGetImage);
fff.delete();
fff = new File(pathGetLowResImage);
fff.delete();
fff = new File(pathGetHighResImage);
fff.delete();
fff = new File(pathGetLowResImages);
fff.delete();
fff = new File(pathGetHighResImages);
fff.delete();
fff = null;

// GetImages
orders[0].GetImages(imageInfos[0].getNativeResolution(),
    new String[] {pathGetImages},
    new int[] { imageInfos[0].getImageID() });
System.out.println("GetImages() succeeded");

// GetImage
orders[0].GetImage(imageInfos[0].getNativeResolution(),
    pathGetImage, imageInfos[0].getImageID());
System.out.println("GetImage() succeeded");

// GetLowResImage
orders[0].GetLowResImage(imageInfos[0].getImageID()
    , pathGetLowResImage);
System.out.println("GetLowResImage() succeeded");

// GetHighResImage
orders[0].GetHighResImage(imageInfos[0].getImageID()
    , pathGetHighResImage);
System.out.println("GetHighResImage() succeeded");
```

```
// GetLowResImages
orders[0].GetLowResImages(new String[] {pathGetLowResImages}
, new int[] {imageInfos[0].getImageID()} );
System.out.println("GetLowResImages() succeeded");

// GetHighResImages
orders[0].GetHighResImages(new String[] {pathGetHighResImages}
, new int[] {imageInfos[0].getImageID()} );
System.out.println("GetHighResImages() succeeded");

// GetOrdersAdmin
orders = session.GetOrdersAdmin(0, 0, 0, "");
System.out.println("GetOrdersAdmin() returns:");
for (int orderi = 0; orderi < orders.length; orderi++) {
    PrintOrder(orders[orderi].getOrderInfo());
}

// GetOrdersByOrderID
orders = session.GetOrdersByOrderID(custID, "", "");
System.out.println("GetOrdersByOrderID() returns:");
for (int orderi = 0; orderi < orders.length; orderi++) {
    PrintOrder(orders[orderi].getOrderInfo());
}

// GetOrdersByCustomer
orders = session.GetOrdersByCustomer(custID, "");
System.out.println("GetOrdersByCustomer() returns:");
for (int orderi = 0; orderi < orders.length; orderi++) {
    PrintOrder(orders[orderi].getOrderInfo());
}

// GetOrdersByDateRange
orders = session.GetOrdersByDateRange(custID, "", 0, 0);
System.out.println("GetOrdersByDateRange() returns:");
for (int orderi = 0; orderi < orders.length; orderi++) {
    PrintOrder(orders[orderi].getOrderInfo());
}

// GetOrdersByStatus
orders = session.GetOrdersByStatus(custID, "", 0);
System.out.println("GetOrdersByStatus() returns:");
for (int orderi = 0; orderi < orders.length; orderi++) {
    PrintOrder(orders[orderi].getOrderInfo());
}

// disconnect
session.Disconnect();
System.out.println("Disconnect() succeeded");
}
catch (ClientLibException ex) {
    System.out.println("Exception occurred: "
+ Integer.toHexString(ex.getErrorCode()) + " - "
+ ex.getLocalizedMessage());
}
}
}
}
```

Native Sample Code

```
#include <stdio.h>
#include <wchar.h>

#include "KIASClient.h"

// print an error message to the console
void PrintException(wchar_t* method, KIASINT code, wchar_t* message) {
    wprintf(L"%s FAILED\n", method);
    wprintf(L"Exception code: 0x%x\n", code);
    wprintf(L"Exception message: %s\n\n", message);
}

// print out the order structure as received by each of the GetOrdersXXX
calls
void PrintOrder(KIASOrderInfo& koi) {
    wprintf(L"\t%s %s %d %d %s\n"
        , koi.OrderID
        , koi.Description
        , koi.DateTime
        , koi.ImageCount
        , koi.CustomerID
    );
}

// test the Kodak Image Access Client API
// calls each API method
int wmain(int argc, wchar_t* argv[]) {
    KIASINT retcode;
    wchar_t message[ERRSIZE];

    KIASLoginInfo LI = {
        L"kiassu",
        L"kimyaxyz",
        L"http://localhost:8080/soap/servlet/rpcrouter",
        L"", L"", L"", 0, 0
    };

    KIASVersionInfo VI;
    KIASINT bIccSupported;
    KIASINT bUserOK;
    KIASSessionInfo SI;
    wchar_t OrderID[IDSIZE];
    KIASOrderStatus OS;
    KIASLocale Locale;
    wchar_t *pCosFile = 0;
    wchar_t *custID = 0;
    wchar_t *orderID = 0;
    KIASINT NumReturned;
    KIASLocale *pLocales = 0;
    KIASLocale setLocale = {L"EN", L"US", L"USD"};
    KIASUser testUser = {L"sdjhfd39", L"billspass", L"bill smith",
        L"301-234-5678", L"123 3rd st. bowie md 20708",
        L"bsmith@hotmail.com"};

    int index;
    KIASProduct *pProducts = 0;
    KIASOutput *pOutputs = 0;
    KIASOrderInfo *pOrders = 0;
    KIASImageInfo *pImages = 0;
}
```

```

wchar_t*      pathGetImages      = L"c:\\GetImages.jpg";
wchar_t*      pathGetImage       = L"c:\\GetImage.jpg";
wchar_t*      pathGetLowResImage  = L"c:\\GetLowResImage.jpg";
wchar_t*      pathGetHighResImage = L"c:\\GetHighResImage.jpg";
wchar_t*      pathGetLowResImages = L"c:\\GetLowResImages.jpg";
wchar_t*      pathGetHighResImages = L"c:\\GetHighResImages.jpg";

wprintf(L"Kodak Image Access example version 2.3.0.0\n");
if (argc != 6 && argc != 5) {
    wprintf(
        L"\nUseage:\nexample cosfile Url name pass custID
orderID\n"
        L"cosfile - the path of the COS file to submit\n"
        L"Url      - the url of the server\n"
        L"name     - the name passed to the Connect call\n"
        L"pass     - the password passed to the Connect call\n"
        L"custID   - the customer ID specified to GetOrders\n"
        L"orderID  - optional - the orderID specified to
GetImageRefs\n"
        L"if orderID is not specified, first one retrieved by "
        L"GetOrders is used\n"
    );

    return 0;
}
if (argc > 1) pCosFile      = argv[1];
if (argc > 2) LI.pUrl       = argv[2];
if (argc > 3) LI.pLoginUserName = argv[3];
if (argc > 4) LI.pPassword  = argv[4];
if (argc > 5) custID       = argv[5];
if (argc > 6) orderID      = argv[6];

wprintf(L"\nSending %s to %s\n", pCosFile, LI.pUrl);

#define SYS message
#define SES SYS, SI.SessionId
#define ORD SES, orderID

// ***** call the 2 methods that do not require a prior Connect

// KIASGetServerVersion
retcode = KIASGetServerVersion(SYS, LI.pUrl, L"", &VI);
if (0 != retcode) {
    PrintException(L"KIASGetServerVersion", retcode, message);
    goto EXCEPTION;
}
wprintf(L"Version = %d.%d.%d.%d\n", VI.nMajor, VI.nMinor
, VI.nBug, VI.nBuild);

// KIASAreICCProfilesSupported
retcode = KIASAreICCProfilesSupported(SYS, LI.pUrl, L"
, &bIccSupported);
if (0 != retcode) {
    PrintException(L"KIASAreICCProfilesSupported", retcode,
message);
    goto EXCEPTION;
}
wprintf(L"ICCProfilesSupported = %s\n"
, bIccSupported ? L"TRUE" : L"FALSE");
// ***** move on to Connect

// KIASConnect
retcode = KIASConnect(SYS, LI, &SI);

```

```
if (0 != retcode) {
    PrintException(L"KIASConnect", retcode, message);
    goto EXCEPTION;
}
wprintf(L"Connected ->\n");
wprintf(L"\tsessionId = %s\n", SI.SessionId);
wprintf(L"\tVersion = %d.%d.%d.%d\n", SI.Version.nMajor
    , SI.Version.nMinor, SI.Version.nBug, SI.Version.nBuild);
wprintf(L"\tLocale = %s %s %s\n"
    , SI.Locale.Language, SI.Locale.Country, SI.Locale.Currency);

// KIASGetNewOrderID
retcode = KIASGetNewOrderID(SES, OrderID);
if (0 != retcode) {
    PrintException(L"KIASGetNewOrderID", retcode, message);
} else {
    wprintf(L"Got OrderID = %s\n", OrderID);
}

// Disconnect
retcode = KIASDisconnect(SES);
if (0 != retcode) {
    PrintException(L"KIASDisconnect", retcode, message);
} else {
    wprintf(L"Disconnected\n");
}

// Reconnect
retcode = KIASReconnect(SES, &SI);
if (0 != retcode) {
    PrintException(L"KIASReconnect", retcode, message);
    goto EXCEPTION;
} else {
    wprintf(L"Reconnected ->\n");
    wprintf(L"\tsessionId = %s\n", SI.SessionId);
    wprintf(L"\tVersion = %d.%d.%d.%d\n", SI.Version.nMajor
        , SI.Version.nMinor, SI.Version.nBug,
SI.Version.nBuild);
    wprintf(L"\tLocale = %s %s %s\n"
        , SI.Locale.Language, SI.Locale.Country,
SI.Locale.Currency);
}

// KIASSubmitCosOrder
retcode = KIASSubmitCosOrder(SES, pCosFile
    , L"", OrderID);
if (0 != retcode) {
    PrintException(L"KIASSubmitCosOrder", retcode, message);
} else {
    wprintf(L"Submitted, OrderID = %s\n", OrderID);
}

// KIASGetOrderStatus
retcode = KIASGetOrderStatus(SES, OrderID, &OS);
if (0 != retcode) {
    PrintException(L"KIASGetOrderStatus", retcode, message);
} else {
    wprintf(L"OrderStatus = %d\n", OS.Status);
}

// KIASGetTimeToComplete
KIASINT ttc;
retcode = KIASGetTimeToComplete(SES, OrderID, &ttc);
```

```

if (0 != retcode) {
    PrintException(L"KIASGetTimeToComplete", retcode, message);
} else {
    wprintf(L"GotTimeToComplete, TTC = %d\n", ttc);
}

// KIASGetLocale
retcode = KIASGetLocale(SEs, &Locale);
if (0 != retcode) {
    PrintException(L"KIASGetLocale", retcode, message);
} else {
    wprintf(L"GetLocale -> %s %s %s\n"
            , Locale.Language, Locale.Country, Locale.Currency);
}

// KIASGetAvailableLocales
retcode = KIASGetAvailableLocales(SEs, &NumReturned, &pLocales);
if (0 != retcode) {
    PrintException(L"KIASGetAvailableLocales", retcode, message);
} else {
    wprintf(L"GetAvailableLocales ->\n");
    for (index = 0; index < NumReturned; index++) {
        wprintf(L"\t%s %s %s\n"
                , pLocales[index].Language
                , pLocales[index].Country
                , pLocales[index].Currency);
    }
}

// KIASSetLocale
retcode = KIASSetLocale(SEs, setLocale, &Locale);
if (0 != retcode) {
    PrintException(L"KIASSetLocale", retcode, message);
} else {
    wprintf(L"SetLocale -> %s %s %s\n"
            , Locale.Language, Locale.Country, Locale.Currency);
}

// ValidateFulfillmentSystemUser
retcode = KIASValidateFulfillmentSystemUser(SEs, testUser, &bUserOK);
if (0 != retcode) {
    PrintException(L"KIASValidateFulfillmentSystemUser", retcode,
message);
} else {
    wprintf(L"ValidateFulfillmentSystemUser = %s\n"
            , bUserOK ? L"TRUE" : L"FALSE");
}

// KIASGetAvailableProducts
retcode = KIASGetAvailableProducts(SEs, setLocale, &NumReturned
, &pProducts);

if (0 != retcode) {
    PrintException(L"KIASGetAvailableProducts", retcode, message);
} else {
    wprintf(L"GetAvailableProducts ->\n");
    for (index = 0; index < NumReturned; index++) {
        wprintf(L"\t%s %s %f %f %f %d %s %s %s %d %d\n"
                , pProducts[index].ProductID
                , pProducts[index].Description
                , pProducts[index].PageWidth
                , pProducts[index].PageHeight
                , pProducts[index].ImageWidth

```

```

        , pProducts[index].ImageHeight
        , pProducts[index].LengthUnit
        , pProducts[index].Locale.Language
        , pProducts[index].Locale.Country
        , pProducts[index].Locale.Currency
        , pProducts[index].OptimalWres
        , pProducts[index].OptimalHres
    );
    }
}

// KIASGetAvailableOutputs
retcode = KIASGetAvailableOutputs(SES, setLocale, &NumReturned,
&pOutputs);
if (0 != retcode) {
    PrintException(L"KIASGetAvailableOutputs", retcode, message);
} else {
    wprintf(L"GetAvailableOutputs ->\n");
    for (index = 0; index < NumReturned; index++) {
        wprintf(L"\t%s %s\n"
            , pOutputs[index].OutputID
            , pOutputs[index].Description
        );
    }
}

// KIASGetOrders
retcode = KIASGetOrders(SES, custID, L"", 0, 0, 0, L"
    , &NumReturned, &pOrders);
if (0 != retcode) {
    PrintException(L"KIASGetOrders", retcode, message);
    goto EXCEPTION;
} else {
    wprintf(L"GetOrders ->\n");
    for (index = 0; index < NumReturned; index++) {
        PrintOrder(pOrders[index]);
    }
}
if (NumReturned == 0) goto DONE;
if (!orderID) orderID = pOrders[0].OrderID;

// KIASGetImageRefs
retcode = KIASGetImageRefs(ORD, &NumReturned, &pImages);
if (0 != retcode) {
    PrintException(L"KIASGetImageRefs", retcode, message);
} else {
    wprintf(L"GetImageRefs ->\n");
    for (index = 0; index < NumReturned; index++) {
        wprintf(L"\t%d %s %d\n"
            , pImages[index].ImageID
            , pImages[index].Description
            , pImages[index].NativeResolution
        );
    }
}

// remove temporary images from previous run
_wremove(pathGetImages);
_wremove(pathGetImage);
_wremove(pathGetLowResImage);
_wremove(pathGetHighResImage);
_wremove(pathGetLowResImages);

```

```

_wremove(pathGetHighResImages);

// KIASGetImages
retcode = KIASGetImages(ORD, pImages[0].NativeResolution
    , 1, &pathGetImages, &pImages[0].ImageID);

if (0 != retcode) {
    PrintException(L"KIASGetImages", retcode, message);
} else {
    wprintf(L"GetImages\n");
}

// KIASGetImage
retcode = KIASGetImage(ORD, pImages[0].NativeResolution
    , pathGetImage, pImages[0].ImageID);

if (0 != retcode) {
    PrintException(L"KIASGetImage", retcode, message);
} else {
    wprintf(L"GetImage\n");
}

// KIASGetLowResImage
retcode = KIASGetLowResImage(ORD, pImages[0].ImageID,
pathGetLowResImage);
if (0 != retcode) {
    PrintException(L"KIASGetLowResImage", retcode, message);
} else {
    wprintf(L"GetLowResImage\n");
}

// KIASGetHighResImage
retcode = KIASGetHighResImage(ORD, pImages[0].ImageID
    , pathGetHighResImage);

if (0 != retcode) {
    PrintException(L"KIASGetHighResImage", retcode, message);
} else {
    wprintf(L"GetHighResImage\n");
}

// KIASGetLowResImages
retcode = KIASGetLowResImages(ORD, 1, &pathGetLowResImages
    , &pImages[0].ImageID);

if (0 != retcode) {
    PrintException(L"KIASGetLowResImages", retcode, message);
} else {
    wprintf(L"GetLowResImages\n");
}

// KIASGetHighResImages
retcode = KIASGetHighResImages(ORD, 1, &pathGetHighResImages
    , &pImages[0].ImageID);

if (0 != retcode) {
    PrintException(L"KIASGetHighResImages", retcode, message);
} else {
    wprintf(L"GetHighResImages\n");
}

// KIASFree ( we're reusing pOrders, so free the prior contents )
retcode = KIASFree(SYS, pOrders);

```

```
if (0 != retcode) goto EXCEPTION;
wprintf(L"KIASFree'd Order array\n");

// KIASGetOrdersAdmin
retcode = KIASGetOrdersAdmin(SYS, 0, 0, 0, L"
    , &NumReturned, &pOrders);
if (0 != retcode) {
    PrintException(L"KIASGetOrdersAdmin", retcode, message);
} else {
    wprintf(L"GetOrdersAdmin ->\n");
    for (index = 0; index < NumReturned; index++) {
        PrintOrder(pOrders[index]);
    }
}

// KIASFree ( we're reusing pOrders, so free the prior contents )
retcode = KIASFree(SYS, pOrders);
if (0 != retcode) goto EXCEPTION;
wprintf(L"KIASFree'd Order array\n");

// KIASGetOrdersByOrderID
retcode = KIASGetOrdersByOrderID(SYS, custID, L"", L"
    , &NumReturned, &pOrders);
if (0 != retcode) {
    PrintException(L"KIASGetOrdersByOrderID", retcode, message);
} else {
    wprintf(L"GetOrdersByOrderID ->\n");
    for (index = 0; index < NumReturned; index++) {
        PrintOrder(pOrders[index]);
    }
}

// KIASFree ( we're reusing pOrders, so free the prior contents )
retcode = KIASFree(SYS, pOrders);
if (0 != retcode) goto EXCEPTION;
wprintf(L"KIASFree'd Order array\n");

// KIASGetOrdersByCustomer
retcode = KIASGetOrdersByCustomer(SYS, custID, L"
    , &NumReturned, &pOrders);
if (0 != retcode) {
    PrintException(L"KIASGetOrdersByCustomer", retcode, message);
} else {
    wprintf(L"GetOrdersByCustomer ->\n");
    for (index = 0; index < NumReturned; index++) {
        PrintOrder(pOrders[index]);
    }
}

// KIASFree ( we're reusing pOrders, so free the prior contents )
retcode = KIASFree(SYS, pOrders);
if (0 != retcode) goto EXCEPTION;
wprintf(L"KIASFree'd Order array\n");

// KIASGetOrdersByDateRange
retcode = KIASGetOrdersByDateRange(SYS, custID, L"", 0, 0
    , &NumReturned, &pOrders);
if (0 != retcode) {
    PrintException(L"KIASGetOrdersByDateRange", retcode, message);
} else {
    wprintf(L"GetOrdersByDateRange ->\n");
    for (index = 0; index < NumReturned; index++) {
        PrintOrder(pOrders[index]);
    }
}
```

```
    }
}

// KIASFree ( we're reusing pOrders, so free the prior contents )
retcode = KIASFree(SYS, pOrders);
if (0 != retcode) goto EXCEPTION;
wprintf(L"KIASFree'd Order array\n");

// KIASGetOrdersByStatus
retcode = KIASGetOrdersByStatus(SES, custID, L"", 0
, &NumReturned, &pOrders);
if (0 != retcode) {
    PrintException(L"KIASGetOrdersByStatus", retcode, message);
} else {
    wprintf(L"GetOrdersByStatus ->\n");
    for (index = 0; index < NumReturned; index++) {
        PrintOrder(pOrders[index]);
    }
}

// KIASDisconnect
retcode = KIASDisconnect(SES);
if (0 != retcode) {
    PrintException(L"KIASDisconnect", retcode, message);
} else {
    wprintf(L"Disconnected\n");
}

// KIASFree
retcode = KIASFree(SYS, pLocales);
if (0 != retcode) goto EXCEPTION;
wprintf(L"KIASFree'd Locale array\n");

retcode = KIASFree(SYS, pProducts);
if (0 != retcode) goto EXCEPTION;
wprintf(L"KIASFree'd Product array\n");

retcode = KIASFree(SYS, pOutputs);
if (0 != retcode) goto EXCEPTION;
wprintf(L"KIASFree'd Output array\n");

retcode = KIASFree(SYS, pOrders);
if (0 != retcode) goto EXCEPTION;
wprintf(L"KIASFree'd Order array\n");

retcode = KIASFree(SYS, pImages);
if (0 != retcode) goto EXCEPTION;
wprintf(L"KIASFree'd Image array\n");

DONE:
    return 0;

EXCEPTION:
    return 1;
}
```


10. **Obtaining an X509 SSL Certificate for Your Kodak Image Access Client**

To enable Secure Sockets Layer (SSL) encrypted communications for your DLS server you need to obtain a server certificate. For the purposes of this document, we will discuss the required steps from a TOMCAT perspective since TOMCAT is the default Web Server shipped with the Kodak Image Access SDK. If you are using a different Web Server, use their instructions.

A certificate is the public key portion of the private / public key-pair for your server, combined with your identity (DNS name), and signed by a trusted certificate authority. At the handshake step of establishing an SSL socket, the server passes its certificate to the client. If the client finds that the certificate is not signed by a trusted party, is not from the server denoted by the DNS name in the certificate, or is past its expiration date, the client terminates the connection.

Before you enable SSL, decide whether you really need it. Most likely, if you are on a private intranet, it is unnecessary. Conversely, if your communications are over the Internet, it should be used. If you decide to enable SSL, be aware that there is a large performance penalty, especially on the server. The SSL implementation used by Tomcat, JSSE, will probably use 100% of your CPU by the time you get to 10 requests per second.

Install Sun JSSE 1.0.2

The client library sits upon the client computer's JRE. JRE 1.3.0 does not come with the package required for SSL support. To enable SSL you must install the JSSE (Java Secure Sockets Extension) package on the server and all clients. The JSSE is not installed with the Kodak Image Access software, you must download and install version 1.0.2 from:

<http://java.sun.com/products/jsse>

1. Download JSSE 1.0.2.
2. Decompress the file.
3. Follow steps 1 through 4a in the uncompressed JSSE INSTALL.txt file.
4. If you have a custom client application, your application should follow step 6 from the JSSE INSTALL.txt file.

Adding a trusted root certificate to your JRE

The JRE comes with a list of trusted Certificate Authorities (CAs). This list contains root certificates from just two companies, Verisign inc. and Thawte inc. This is not affected by your choice of Web Server. Therefore, unless you wish to require each client to install a custom root certificate, you must obtain your server certificate from either:

<http://www.verisign.com>

or

<http://www.thawte.com>

If you wish to obtain a certificate from a different vendor, install the appropriate root certificate from that vendor on each client computer. To do this:

1. Obtain the root certificate by copying it from the certificate vendor's Web site to your clipboard. Paste it into a text editor and save the file. For example, root.cer.
2. Locate the file 'cacerts' for your install of the JAVA runtime. Relative to the top level of your JRE directory tree, cacerts is located in /lib/security. On a typical WINDOWS install of the JRE, the full path of the file is:
C:\ProgramFiles\JavaSoft\JRE\1.3\lib\security\cacerts
3. Copy your root.cer file to the same location and change to that directory from a command prompt.
4. Execute the keytool import command with the following command line, replacing the ?questions? items with values appropriate for you:

```
keytool -import -file ?root.cer? -keystore cacerts -storepass ?changeit? -alias ?root?
```

Where: ?root cer? specifies the certificate vendor name and ?change it? specifies the certificate password.

The password for cacerts defaults to 'changeit.' If it has been changed, use the appropriate password. Specify a name for the root as the alias parameter.

Generate a private / public key-pair for your server

1. Use the JRE keytool utility with the following command line, replacing the ?questioned? entities with appropriate values for you. The command is quite long and must be placed on a single line despite it being line-wrapped here. You may wish to make a .bat file of it.

```
keytool -genkey -alias ?myserv? -dname "CN=?myserv.mycompany.com?,  
O=?My Company?, OU=?Photofinishing Division?, L=?Chicago?, S=?Illinois?,  
C=?US?" -keyalg RSA -keypass ?mypass? -storepass ?mypass? -keystore  
?myserv.keystore?
```

2. Within the command line, ensure that the following requirements have been met:
CN is your DNS name. Your CA will verify that your company legally owns the domain name.

O is your official company name. Your CA will cross verify this with the name associated with your DUNS number.

L, S, and C match your DUNS information.

OU is your division or other identifying information.

3. Obtain and secure your private key.

The resulting file, myserv.keystore, contains your key-pair. It is important to keep this file secure; it contains your private key. Loss of your private key compromises your security.

Generate a CSR (Certificate Signing Request) for your CA (Certificate Authority)

1. Use the JRE keytool utility to generate a CSR from the keystore. Use the following command line: keytool -certreq -alias ?myserv? -file ?myservCSR.txt? -keypass ?mypass? -storepass ?mypass? -keystore ?myserv.keystore?
2. Open the new CSR in a text editor such as Notepad. Select and copy the entire contents.

Submit your CSR and get back your certificate

1. Fill out the various web forms as required by your CA.
2. When asked for your CSR, paste in the CSR copied in step 2 above.
3. After the CA is satisfied with your application and payment, usually several days, your new certificate is received via email.
4. Select and copy the certificate from the email. Paste it into a new Notepad document. Save the document as mycert.cer. When selecting, get all of the text between
-----BEGIN CERTIFICATE-----
and
-----END CERTIFICATE-----
including those two lines.
5. Examine your certificate with Microsoft Explorer Certificate Viewer by double clicking the .cer file.

Import the new certificate into your Keystore

1. Import any intermediate certificates. Usually, when a CA signs a certificate request, it does so not with its root certificate but with another certificate that is in turn signed by the root. In order to import your certificate, the entire chain from root down is required. The email that contained your certificate should also contain a link to any required intermediate certificates. Follow the specified link, paste the intermediate certificate into Notepad, and save the file as `intermediate.cer`.
2. Use the following command line to import the intermediate certificate into your keystore: `keytool -import -alias intermediate -file intermediate.cer -storepass ?mypass? -keystore ?myserv.keystore? -trustcacerts`
3. Import your certificate with the following command line. Be sure that `?myserv?` is the same alias used when initially creating the keystore. `keytool -import -alias ?myserv? -file ?mycert.cer? -storepass ?mypass? -keystore ?myserv.keystore? -trustcacerts`

Configure your Web Server

After importing the certificate into your keystore, you are now ready to configure TOMCAT. The following steps document the enabling of SSL in TOMCAT 4.0.1.

1. Open the server.xml file in the tomcat\conf directory in Notepad or some text editor.
2. Navigate to a section that defines the Connector elements.
Note: Search for the text "Define an SSL HTTP/1.1 Connector on port 8443."
3. The Connector element in this section appears as follows.

```
<!-- Define an SSL HTTP/1.1 Connector on port 8443 -->
<!--
  <Connector
    className="org.apache.catalina.connector.http.HttpConnector"
    port="8443" minProcessors="5" maxProcessors="75"
    enableLookups="true"
    acceptCount="10" debug="0" scheme="https" secure="true">
    <Factory
      className="org.apache.catalina.net.SSLServerSocketFactory"
      clientAuth="false" protocol="TLS"/>
    </Connector>
-->
```

Change the above element to appear as the following.

```
<!-- Define an SSL HTTP/1.1 Connector on port 8443 -->
<!--      [The comment block is moved here]
-->
<Connector
className="org.apache.catalina.connector.http.HttpConnector"
  port="8443" minProcessors="5" maxProcessors="75"
  enableLookups="true"
  acceptCount="10" debug="0" scheme="https" secure="true">
<Factory
  className="org.apache.catalina.net.SSLServerSocketFactory"
  clientAuth="false" protocol="TLS"
  keystoreFile=" ?path/myserv,keystore?"
  keystorePass=" ?mypass?" />
</Connector>
```

Note: Two attributes have been added to the "Factory" sub-element namely, "keystoreFile" and "keystorePass". The "keystoreFile" attribute refers to the keystore file containing the imported certificate. Set the "keystoreFile" attribute to refer to the directory where the certificate is located. Set the "keystorePass" attribute to the value used while purchasing the certificate.

Parameter Key for Common Parameters

?root.cer?	File name of a new root certificate.
?changeit?	Keystore password for cacerts.
?root?	Alias for a new root certificate within the cacerts root keystore.
?myserv?	Alias for your keypair within your keystore.
?mypass?	Keystore password for your keystore (not cacerts).
?myserv.keystore?	File name of your keystore.
?path/myserv.keystore?	Full path to ?myserv.keystore?
?myservCSR.txt?	Filename of your CSR.
?mycert.cer?	Filename of your actual certificate.

11. Server Status Codes

This chapter captures the various status codes and messages that are returned by a server to its clients. This is aimed to provide a reference for the developer, support personnel and clients.

No.	Message	Description
0	Success	The call completed successfully.
40002000	No data retrieved for GetImageRefs.	[Servlets] The KODAK Image Access Enabling software was unable to find the image references on the fulfillment system. The Image refs count is returned as 0.
40002001	No data retrieved for GetOrders.	[Servlets] No order information was found on the fulfillment system matching your query. The order count is returned as 0.
40002002	No data retrieved for GetImages.	[Servlets] No images were found on the fulfillment system for the order specified. The image count is returned as 0.
40002003	No data retrieved for GetAvailableProducts.	[Servlets] No products were found on the fulfillment system. The product count is returned as 0.
40002004	No Locales found in the locales table.	[Servlets] No locales were found on the KODAK Image Access Enabling software. The locale includes the language, currency and country codes.
40002005	No data retrieved for GetOrdersAdmin.	[Servlets] No order information was found on the fulfillment system matching your query. The order count is returned as 0.
40002006	No data retrieved for GetOutputs	[Servlets] No data was found on the fulfillment system matching your query.
C0002100	Blank Login Name Not Allowed.	[Servlets] KODAK Image Access Enabling software's Java layer logs this error if the client sends a blank login name for the Connect() method.

No.	Message	Description
C0002101	Invalid Password Try Again.	[Servlets] KODAK Image Access Enabling Software's JAVA layer logs this error if the client sends an invalid password name for the Connect() method.
C0002102	Invalid SessionID Try Again.	[Servlets] KODAK Image Access Enabling Software's JAVA layer logs this error if the client sends an invalid session id for a request that requires session id validation.
C0002103	Invalid language.	[Servlets] KODAK Image Access Enabling Software's JAVA layer logs this error if the client sends an invalid language id for a request that requires language id validation.
C0002104	Zero length or NULL Order ID is invalid.	[Servlets] The client sends an invalid order id to the server.
C0002105	Invalid User.	[Servlets] KODAK Image Access Enabling Software's JAVA layer logs this error if the client tries to log into the system with a user account that does not exist in the server.
C0002115	Invalid IP address.	[Servlets] KODAK Image Access Enabling Software's JAVA layer logs this error if the client sends an invalid (blank) IP address in its connect request.
C0002116	Invalid file size.	[Servlets] KODAK Image Access Enabling Software's JAVA layer logs this error if the client sends an invalid file size.
C000211C	KIAS Server Error	[Servlets] A generic error message text that is sent to clients in place of server centric messages.
C000211E	Insufficient bytes sent from client for the COS order submission.	[Servlets] KODAK Image Access Enabling Software's JAVA layer logs this error if client does not send the exact number of bytes that was reserved for a COS order.
C000211F	Conflicting file size requested for a partial order submission.	[Servlets] KODAK Image Access Enabling Software's JAVA layer logs this error if client submits a reserve space request during a partial upload attempt with a file size that is different than what it had submitted originally.
C0002FFF	KIAS function not implemented.	[KODAK Image Access Enabling software] KODAK Image Access Enabling Software's JAVA layer logs this error if the DLS Server has not implemented this function.

No.	Message	Description
C000212E	Access denied for GetOrdersAdmin.	[Servlets] KODAK Image Access Enabling Software's JAVA layer logs this error if a client without admin privileges tries to retrieve "admin-privileged" orders.
C000212F	KIAS Device communication failed	[Enabling Software for DLS] KODAK Image Access Enabling Software's JAVA layer logs this error if the client receives cannot communicate to the DLS Server.
C0002133	Invalid input for ValidateFulfillmentSystemUser	[Enabling Software for DLS] KODAK Image Access Enabling Software's JAVA layer logs this error if the client submits a null or empty string of all input parameters.
40003C02	Order rejected.	[Enabling Software for DLS] KODAK Image Access Enabling Software's JAVA layer logs this error, if the client submits an invalid COS file.
40003C03	Order reserved.	[Enabling Software for DLS] KODAK Image Access Enabling Software's JAVA layer logs this error, if the client requests reserved space prior to the COS file being submitted.
40003C04	Order received.	[Enabling Software for DLS] KODAK Image Access Enabling Software's JAVA layer logs this error, if the client requests order status prior to the COS file process being started.
40003C05	Order in process.	[Enabling Software for DLS] KODAK Image Access Enabling Software's JAVA layer logs this error, if the client requests order status prior to the COS file process being completed.
40003C06	Order completed.	[Enabling Software for DLS] KODAK Image Access Enabling Software's JAVA layer logs this error, if the client requests order status when the COS file processing is complete. The COS file is in the system.

12. Glossary

Archived Order

Orders that are archived on the server and are available for Image Access. Not all orders are archived. Some orders are just printed. Whether orders are archived depends upon the server configuration.

COS (Customer Order Specification) file

A container file that may hold a set of desired products, a set of image files, and customer information. The Image Access Client SDK requires that orders be submitted in the COS format. Kodak provides a COS SDK for creating COS files. It is recommended that you use only version 2.6 of this SDK. Version 3.0 is still too new to be supported on all Servers.

DLS (Digital Lab System) Server

A server used by photo finishers to fulfill customer photographic orders. The KODAK Image Access Client SDK requires that your DLS be at version 2.0 and have had the server patch applied. In the future, other Kodak photo finishing products and services will also provide server capability.

Image Access

The process of retrieving images archived upon a server. The general steps are: get a list of orders, get a list of images within an order, and get an image.

ICC (International Color Consortium) profile

A precise specification of a color space. A server indicates True / False in response to the `AreICCProfilesSupported()` method as to whether it will honor ICC profile specifications embedded within submitted order files. Currently, all Servers return False - there is no support.

JAVA

A popular computer language developed by SUN MICROSYSTEMS. It is syntactically similar to C++ but has the special feature of being compiled into a portable byte code that may be run on the JAVA Virtual Machine from any JRE. The KODAK Image Access Client software is written in JAVA and runs on any JAVA 2.0 JRE. The KODAK Image Access Client software also provides a JNI wrapper over the library to allow the use of other languages, like C++.

JDK (JAVA Development Kit)

A software package used by a developer to create JAVA software. It contains tools such as a JAVA compiler, as well as a JRE.

JNI (JAVA Native Interface)

A library provided by SUN so that Native Code may call JAVA code, and vice versa. KODAK Image Access Client software has used this library to create a native layer over the JAVA KODAK Image Access Client SDK library.

JRE (JAVA Runtime Environment)

The software required to run a JAVA program. Versions of JRE are available for many computer platforms, including WINDOWS, SOLARIS, and LINUX.

KODAK Image Access Client

A software application that uses the KODAK Image Access Client library to communicate with a server. A KODAK Image Access Client may reside either on the same computer as the server or on a different computer. These two cases would be referred to as 'local client' and 'remote client,' respectively.

Server

A networked computer that accepts order submissions and image access requests. Servers consists of a Web server, a DLS, and server software attached to the Web server.

Native Code

A generic term for software that has been compiled to run on a specific computer and OS. We use the term to mean 'anything that isn't JAVA.' We provide a C language header file and expect that any native KIAS application that you write will be written in C++.

Submitted Order

A request for a photo finishing order sent to the DLS server. The KODAK Image Access Client SDK allows a client application to submit orders to a Server. A submitted order, unless it is improperly formed, will have the products specified within it produced. Submitted orders are not necessarily archived.



EASTMAN KODAK COMPANY
Kodak Consumer Imaging Division
Rochester NY 14650
www.kodak.com/go/kiasdevelopers
© Eastman Kodak Company, 2002