ION Java User's Guide

July, 2003 Edition

Copyright © Research Systems, Inc.
All Rights Reserved

RSI
Research Systems Inc.

0703ION20JVA

Restricted Rights Notice

The IDL®, ION Script™, and ION Java™ software programs and the accompanying procedures, functions, and documentation described herein are sold under license agreement. Their use, duplication, and disclosure are subject to the restrictions stated in the license agreement. Research Systems, Inc., reserves the right to make changes to this document at any time and without notice.

Limitation of Warranty

Research Systems, Inc. makes no warranties, either express or implied, as to any matter not expressly set forth in the license agreement, including without limitation the condition of the software, merchantability, or fitness for any particular purpose.

Research Systems, Inc. shall not be liable for any direct, consequential, or other damages suffered by the Licensee or any others resulting from use of the IDL or ION software packages or their documentation.

Permission to Reproduce this Manual

If you are a licensed user of this product, Research Systems, Inc. grants you a limited, nontransferable license to reproduce this particular document provided such copies are for your use only and are not sold or distributed to third parties. All such copies must contain the title page and this notice page in their entirety.

Acknowledgments

IDL® is a registered trademark and ION™, ION Script™, ION Java™, are trademarks of Research Systems Inc., registered in the United States Patent and Trademark Office, for the computer program described herein.

Numerical Recipes™ is a trademark of Numerical Recipes Software. Numerical Recipes routines are used by permission.

GRG2™ is a trademark of Windward Technologies, Inc. The GRG2 software for nonlinear optimization is used by permission.

NCSA Hierarchical Data Format (HDF) Software Library and Utilities
Copyright 1988-2001 The Board of Trustees of the University of Illinois
All rights reserved.

NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities
Copyright 1998, 1999, 2000, 2001, 2002 by the Board of Trustees of the University of Illinois. All rights reserved.

CDF Library
Copyright © 1999
National Space Science Data Center
NASA/Goddard Space Flight Center

NetCDF Library
Copyright © 1993-1996 University Corporation for Atmospheric Research/Unidata

HDF EOS Library
Copyright © 1996 Hughes and Applied Research Corporation

This software is based in part on the work of the Independent JPEG Group.

Portions of this software are copyrighted by INTERSOLV, Inc., 1991-1998.

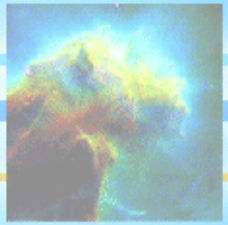
Use of this software for providing LZW capability for any purpose is not authorized unless user first enters into a license agreement with Unisys under U.S. Patent No. 4,558,302 and foreign counterparts. For information concerning licensing, please contact: Unisys Corporation, Welch Licensing Department - C1SW19, Township Line & Union Meeting Roads, P.O. Box 500, Blue Bell, PA 19424.

Portions of this computer program are copyright © 1995-1999 LizardTech, Inc. All rights reserved. MrSID is protected by U.S. Patent No. 5,710,835. Foreign Patents Pending.

This product includes software developed by the Apache Software Foundation (http://www.apache.org/)

IDL Wavelet Toolkit Copyright © 2002 Christopher Torrence.

Other trademarks and registered trademarks are the property of the respective trademark holders.



Contents

Chapter 1:
Configuring ION Java 9

Starting and Configuring the ION Daemon 10
Configuring ION Java for Windows 11
The ION Java Properties Dialog 11
Checking Status with the ION Java Status Utility 19
Windows Command Line Installation of the ION Daemon 20
Using Windows Services Manager to Start the ION Daemon 21
Configuring ION Java for UNIX 22
Starting the ION Daemon on UNIX 22
Starting the ION Daemon at Boot Time 26
Checking the Status of the ION Daemon 27
Shutting Down the ION Daemon 27

Manually Configuring Your Web Server 29
Configuring The ION HTTP Tunnel Broker 31
Using the Tunnel Broker 31
Starting the ION Tunnel Broker Daemon 32
Command Security 34
Security Command Files 34
Client Verification 35
Connection Limit 35

Chapter 2: Overview 37

What is ION Java? 38
Recommended Skills 38
ION Java Architecture 40
ION Server 40
ION Daemon 41
ION HTTP Tunnel Broker 41
Pre-Built ION Client Applets 43
ION Component Classes 43
ION Low-Level Classes 43
ION Java Limitations 44
Server Limitations 44
IDL Limitations 44
ION Java Performance Considerations 45
Tips for Increasing Execution Speed in ION Java 45
Bandwidth Issues 46
Running the ION Java Examples 48
Where to Place HTML and Class Files 51
Testing ION Applications Locally 51
Publishing ION Applications on Your Web Server 51
Where to Locate the ION Class Files 52
What Are the Required Class Files? 53

Chapter 3:
Overview of the ION Java Classes 55

The ION Java Class Hierarchy 56
ION Low-Level Classes 57
ION Component Classes 58
ION Pre-Built Applets 61
Using the Component Classes 63
Setting Values 63
Getting and Setting Properties 63
Drawing 63
AWT vs. Swing 64

Chapter 4:
Using ION's Pre-Built Applets 67

The <APPLET> Tag 68
Attributes 68
Supporting Java-Incapable Browsers 71
Parameters Specified via <PARAM> Tags 72
Connecting to the ION Server 72
Behavior Parameters 74
IONGraphicApplet 76
IONContourApplet 78
IONMapApplet 81
IONPlotApplet 84
IONSurfaceApplet 86

Chapter 5:
Building ION Applets and Applications 89

Direct Graphics in ION 90
The ION Device 90
Keywords Accepted by the ION Device 90
Object Graphics in ION 94
Using Object References 95

Compiling .java Files 96
Error Handling and ION Exceptions 98
Debug Mode 99
Debugging Your Application 99
Converting Between IDL and Java Bytes 100
Considerations Specific to ION Applets 102
Including Applets in HTML Pages 102
Supporting Java Archive Files 103
Browser Support of ION Class Library Versions 103
Supporting Multiple Browser Types 104
Simple Applet Example 105
Further Examples 109
ION Applets and Scripting Languages 109
Tips and Tricks 115

Chapter 6:
ION Java Class and Method Reference 117

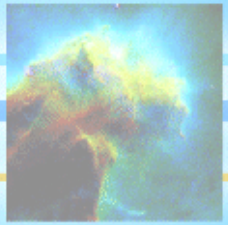
How to Use this Chapter 119
Syntax 119
Arguments 120
Exceptions 121
Example 121
IONCallableClient Class 122
IONCanvas / IONJCanvas Class 134
IONCommandDoneListener Interface 139
IONComplex Class 141
IONContour / IONJContour Class 146
IONDComplex Class 153
IONDisconnectListener Interface 158
IONDrawable Interface 160
IONGraphicsClient Class 163
IONGraphicConnection Interface 172

IONGrConnection / IONJGrConnection Class 173
IONGrContour Class 180
IONGrDrawable / IONJGrDrawable Class 188
IONGrGraphic Class 197
IONGrMap Class 203
IONGrMapContinents Class 209
IONGrMapGrid Class 212
IONGrMapImage Class 215
IONGrPlot Class 220
IONGrSurface Class 226
IONMap / IONJMap Class 232
IONMouseListener Interface 238
IONOffScreen Class 242
IONOutputListener Interface 245
IONPlot / IONJPlot Class 246
IONSurface / IONJSurface Class 252
IONVariable Class 259
IONWindowingClient Class 282

Chapter 7: Troubleshooting 287

Avoiding Conflicting ION Installations 288
Checking Web Server Communication 289
Troubleshooting ION Service Problems 289
Troubleshooting Applets that Fail to Display 290
Troubleshooting “Not Found” Errors 292
Troubleshooting Licensing Errors 293
Setting the IDL Path 293
Troubleshooting Security Errors 294
Encountering Browser Timeouts with Java Errors 295
ION Server Timeout 295
JDK 1.2 Required for Clients 295

Index 297



Chapter 1:
Configuring ION Java

This chapter discusses the process of setting up and starting the ION Daemon after it has been installed, and discusses strategies for locating your HTML and Java class files. The following topics are covered:

- Configuring ION Java for Windows
- Configuring ION Java for UNIX
- Configuring The ION HTTP Tunnel Broker
- Command Security

Starting and Configuring the ION Daemon

The ION Daemon is a process that listens to a specified socket port, waiting for a communication request. Once a connection is received and verified, the daemon starts up an ION Server process, connects the client to the server process and waits for further connection requests.

Note

If you are unfamiliar with ION Java, it may be helpful to refer to Chapter 2, "Overview" for information regarding general ION Java architecture including an overview of the ION server/client relationship, the ION Daemon and ION Tunnel Broker services before beginning the configuration process.

If you have client applets located behind a firewall, which attempt and fail to communicate with an ION Server on the other side of a firewall, you will need to start and configure the ION Tunnel Broker in addition to the ION Daemon. See "Configuring The ION HTTP Tunnel Broker" on page 31 for more information.

ION provides a set of utility programs that allow you start, configure and manage the ION Daemon and the HTTP Tunnel Broker. See the following section for your platform:

- On UNIX platforms, run the utility programs from the shell prompt. See "Configuring ION Java for UNIX" on page 22 for more information.
- On Windows platforms use the GUI utilities described in the section. "Configuring ION Java for Windows" on page 11.

All utilities and command-line programs discussed in this chapter are located in the following platform directory:

- Windows: *RSI-DIR*\IDL60\products\ION20\ion_java\bin
- UNIX: *RSI-DIR*/ion_1.6/ion_java/bin

where *RSI-DIR* is the directory in which you installed ION.

Configuring ION Java for Windows

ION Java for Windows includes two dialog-based utilities that duplicate the functionality of the command-line utilities discussed in "Starting the ION Daemon on UNIX" on page 22.

- The **ION Java Properties** utility presents a tabbed dialog that allows you to start, configure and control ION Daemon settings. For more information, see the following section, "The ION Java Properties Dialog".
- The **ION Java Status** utility allows you to check the status of the ION Daemon or the Tunnel Broker. For more information, see "Checking Status with the ION Java Status Utility" on page 19.

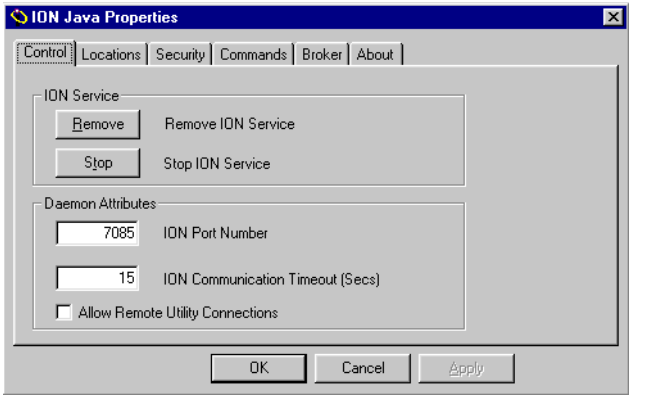
The ION Java Properties Dialog

To configure ION Java for Windows perform the following steps:

1. Access the ION Java Properties utility from the **Start** menu by selecting **Programs** → **Research Systems ION 1.6** → **ION Java Properties**. This program, `wionprop.exe`, is located in the `bin` directory of your ION Java installation.
2. Modify the desired settings on each of the tabs described in the following sections:
   - See "The Control Tab" on page 12 to install, remove, start or stop the ION Daemon service.
   - See "The Locations Tab" on page 14 to set ION and IDL directory paths, the ION log file and IDL search path.
   - See "The Security Tab" on page 16 and "The Commands Tab" on page 17 to define which IDL commands should or should not be executed.
   - See "The Broker Tab" on page 18 to configure the ION Tunnel Broker.
3. After making changes, click "OK" to accept the change and close the dialog, "Apply" to accept the change but leave the dialog open, or "Cancel" to close the dialog without making any changes.

The Control Tab

The Control Tab is used to start, stop or remove the ION Service and to configure the ION daemon attributes.



Use the Control tab to make the following changes:

Attribute	Description
Remove / Install	Removes or installs the ION Service in the Windows service registry. **Note -** The service was automatically installed during your installation which called `ion_srvinst.exe` with the `-install` option so you will probably not need to remove or install the service until a new version of ION is released. See "Windows Command Line Installation of the ION Daemon" on page 20 for more information regarding the `ion_srvinst` utility.

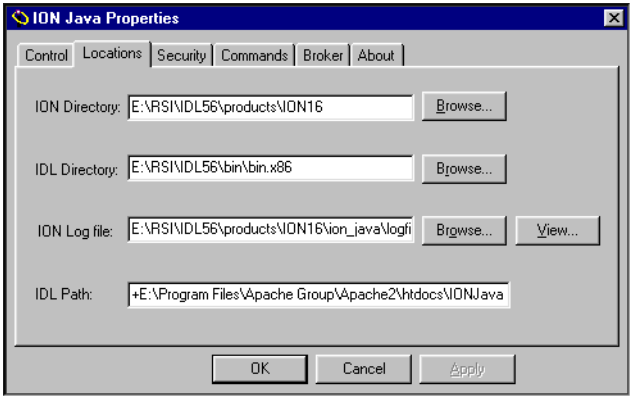
Table 1-1: ION Java Properties — Control Tab

Attribute	Description
Start / Stop	Starts or stops the ION service. This button performs the same actions as the Services dialog described in the section "Using Windows Services Manager to Start the ION Daemon" on page 21. **Note -** Occasionally, the service may be unable to respond to a request. For more information, see "Troubleshooting ION Service Problems" on page 289.
ION Port Number	Defines the port number the ION Service listens to for connection requests.
ION Communication Timeout	Specifies the number of seconds ION will wait before closing a connection.
Allow Remote Utility Connections	Select this checkbox to allow the `iondown` and `ionstat` utility programs to be run from computers other than the one on which the ION service is installed.

Table 1-1: ION Java Properties — Control Tab (Continued)

The Locations Tab

The Locations Tab of the ION Java Properties dialog configures the paths used by ION Java.



Use the Location Tab, to define the following settings:

Attribute	Description
ION Directory	Specifies the ION installation directory on your server machine.
IDL Directory	Specifies the IDL directory on your server machine.
ION Log File	Specifies the location of a text file that will contain the ION Server logs. Click “View” to view the contents of the log file.

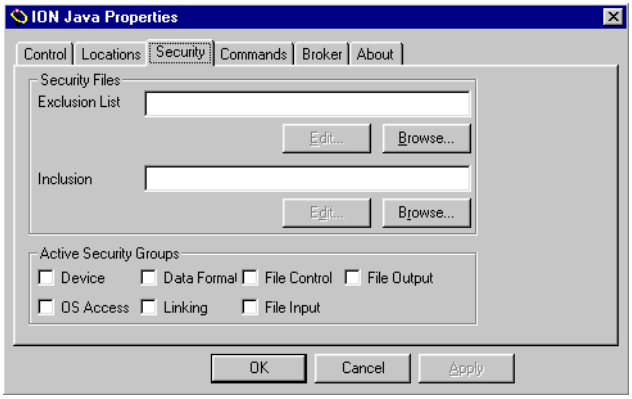
Table 1-2: ION Java Properties — Locations Tab

Attribute	Description
IDL Path	Specifies the search path to the directory or directories containing the IDL library (.pro and .sav) files. In this field, specify the Web server's ION Java directory where IDL files were copied during ION installation. For example, using the Apache web server, this path may be: `+C:\Program Files\Apache Group\Apache2\htdocs\IONJava` You can enter either a ";" separated list of multiple directories, or use a "+" in front of a directory indicating that all subdirectories of the specified directory should be searched. See "Setting the IDL Path" on page 293 if you need more information.

Table 1-2: ION Java Properties — Locations Tab (Continued)

The Security Tab

The Security Tab, pictured in the following figure, allows you to define the commands that ION should or should not execute.



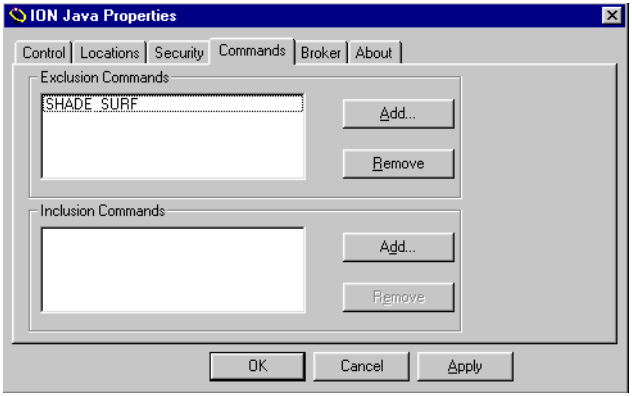
Use the Location Tab, to define the path to text files containing the following items:

Attribute	Description
Exclusion List	Contains a list of the commands ION should not execute. Click the **Edit** button on either field to edit the text file. See "Security Command Files" on page 34 for details on how the ION Daemon handles inclusion and exclusion lists and how they can be created.
Inclusion List	Contains a list of the commands ION is allowed to execute. Click the **Edit** button on either field to edit the text file. See "Security Command Files" on page 34 for details on how the ION Daemon handles inclusion and exclusion lists and how they can be created.
Active Security Groups	Select options to disable entire classes of IDL functionality. See the table, "Active Security Group Tokens" on page 25 for a description of the security tokens associated with this field.

Table 1-3: ION Java Properties — Security Tab

The Commands Tab

The Commands tab dialog offers another way of specifying IDL commands which should or should not be executed.



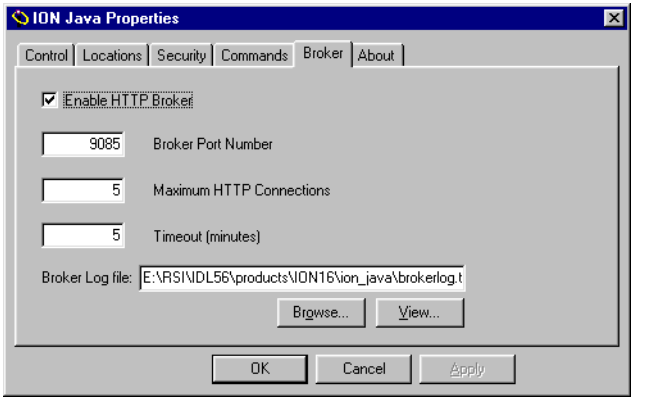
To add a command, click **Add** and enter a of an IDL command to the list of individual commands to be excluded or allowed by the ION security mechanism. To remove a command, select a command from either list and click **Remove**.

Note

Security Command files can also be used to designate which IDL commands are or are not executed. See “Security Command Files” on page 34 for details.

The Broker Tab

The Broker tab allows the configuration of the ION Tunnel Broker. See "Configuring The ION HTTP Tunnel Broker" on page 31 for information about the Tunnel Broker.



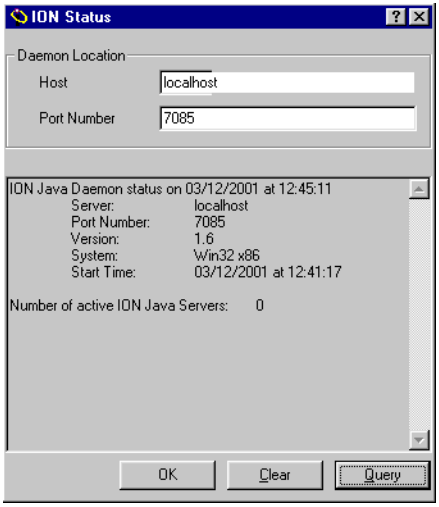
Within the Broker tab, you can make the following configurations:

Attribute	Description
Enable HTTP Broker	Select this option to enable or disable the ION Tunnel Broker.
Broker Port Number	Enter the port number to which the ION Tunnel Broker will listen. The default value is 9085.
Maximum HTTP Connections	Enter the maximum number of HTTP connections allowed at any one time.
Timeout (minutes)	Enter the number of minutes an ION peer should wait before closing a connection.
Broker Log File	Specifies the location of a text file that will contain the ION Tunnel Broker logs. Click **View** to view the contents of the log file.

Table 1-4: ION Java Properties — Broker Tab

Checking Status with the ION Java Status Utility

The ION Java Status utility allows you to obtain information about the current state of the ION Daemon and ION Tunnel Broker. Access the ION Java Status utility from the **Start** menu by selecting **Programs → Research Systems ION 1.6 → ION Java Status**. This program, `wionstat.exe`, is located in the `bin` directory of your ION Java installation.



To check the status of a service set the following fields:

Field	Description
Host	Set this field to the name of the computer on which either the ION Daemon or ION Tunnel Broker is running.

Table 1-5: ION Status

Field	Description
Port Number	Set this field equal to the port being watched by either the ION Daemon or the ION Tunnel Broker. The default is port 7085 which is the default port for the ION Daemon. To check the status of the ION HTTP Tunnel Broker, you must specify the port on which the Tunnel Broker is listening (the default is 9085). Click **Query** to retrieve information on the Daemon or Tunnel Broker running on the specified host and port. Click **Clear** to clear the display, or **OK** to dismiss the dialog.

Table 1-5: ION Status (Continued)

Windows Command Line Installation of the ION Daemon

Use the `ion_srvinst.exe` program to install, control, and check the status of the ION Daemon Windows service. The ION installation process automatically calls `ion_srvinst` with the `-install` flag so you do not need to install it again. You can use this program to remove the service or configure how the daemon is started.

Note

To start, stop and remove the ION Daemon service, you can also use the ION Java Properties dialog, described in "Configuring ION Java for Windows" on page 11. To configure manual or automatic startup, you can also use the Windows Services dialog described in "Using Windows Services Manager to Start the ION Daemon" on page 21.

The `ion_srvinst` command uses the following syntax:

```
ion_srvinst [-install | -remove] [-start = auto | manual]
[-iondir=iondir]
```

Note

If no switches are specified, `ion_srvinst` prints the status of the service.

The switches to the `ion_srvinst` command are described below:

-install

Set this switch to install the ION Daemon service into the system.

-remove

Set this switch to remove the ION Daemon service from the system.

-start

Set this switch to specify the *start type* of the service. If set to `auto`, the ION Daemon service will be started by the Windows system at startup. If set to `manual` (the default), the ION Daemon service must be started through the ION Java Properties dialog or the Control Panel Services dialog. Note that this option is ignored if the `-install` switch is not also specified.

-iondir

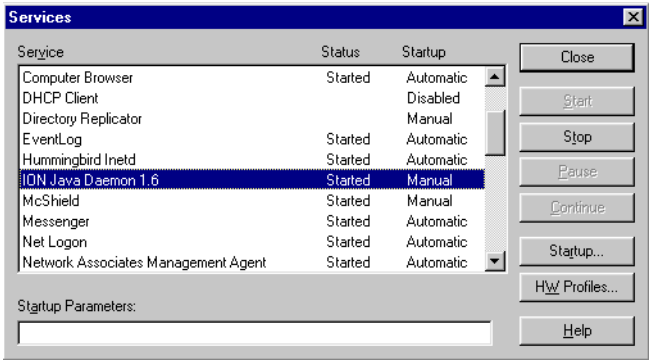
Use this switch to specify the ION installation directory, for example, *`RSI-DIR`*`\IDL60\products\ION20`, *not* *`RSI-DIR`*`\IDL60\products\ION20\ion_java`. Setting this switch will override any Windows registry entries and environment variable settings.

Using Windows Services Manager to Start the ION Daemon

Use the Services dialog to start, stop or configure automatic or manual startup modes of the ION Java Daemon. To open the Service dialog, do one of the following:

- On Windows NT, select **Start → Settings → Control Panel → Services**
- On Windows 2000, select **Start → Settings → Control Panel → Administrative Tools → Services**

In the Services dialog, select ION Java Daemon 1.6 and use the interface to modify the settings. The following figure shows the Windows NT Services dialog with the ION Daemon selected.



Configuring ION Java for UNIX

This section covers the following topics:

- "Starting the ION Daemon on UNIX" on page 22
- "Starting the ION Daemon at Boot Time" on page 26
- "Checking the Status of the ION Daemon" on page 27
- "Shutting Down the ION Daemon" on page 27

Starting the ION Daemon on UNIX

Start the ION Daemon process by executing the `iond` command at the shell prompt. The `iond` command uses the following syntax:

```
iond [-exfile=filename] [-infile=filename] [-excomm="routine0,
routine1, ...routinen"] [-incomm="routine0, routine1,
...routinen"] [-http] [-httplog=filename] [-httpport=port]
[-httptimeout=minutes] [-logfile=filename ]
[-maxconn=connections] [-port=port] [-rutil]
[-security="device, df, filein, fileout, fileio, linking,
none, os"] [-timeout=seconds]
```

Note

You must execute the `ion_setup` script before starting the ION Daemon. For more information, see "Define ION Environment Variables and Aliases" in the *Installing and Licensing IDL 6.0* manual.

The following command line parameters are accepted by the ION Daemon:

-exfile

Set this switch to the name of a file that contains a list of IDL commands (procedure or function names) that the server should not accept. Any command that attempts to execute one of the listed routines will be rejected. The file should contain one routine name on each line. Blank lines and lines that begin with the "#" character are ignored.

Specifying an exclude file will not alter the list of routines rejected as a result of the setting of the `-security` switch.

-infile

Set this switch to the name of a file that contains a list of IDL commands (procedure or function names) that the server should accept. Any command that attempts to

execute a routine that is not in the list will be rejected. The file should contain one routine name on each line. Blank lines and lines that begin with the "#" character are ignored.

Specifying an include file will not alter the list of routines rejected as a result of the setting of the `security` switch.

Note

If a routine is *excluded* (either via an exclude file, a list of excluded routines, or via the `-security` switch), it will be rejected even if that routine is also included in an include file or list.

-excomm

Set this switch to a comma-separated list of IDL commands (procedure or function names) to add to the exclusion list. This switch works in the same way as the `-exfile` switch; it is provided as a convenience.

Specifying a list of routines to exclude will not alter the list of routines rejected as a result of the setting of the `-security` switch.

-incomm

Set this switch to a comma-separated list of IDL commands (procedure or function names) to add to the inclusion list. This switch works in the same way as the `-infile` switch; it is provided as a convenience.

Specifying a list of routines to include will not alter the list of routines rejected as a result of the setting of the `-security` switch.

Note

If a routine is *excluded* (either via an exclude file, a list of excluded routines, or via the `-security` switch), it will be rejected even if that routine is also included in an include file or list.

-http

Set this switch to start the ION HTTP Tunnel Broker when starting the ION Daemon. See “Configuring The ION HTTP Tunnel Broker” on page 31 for details on the ION Tunnel Broker.

-httplog

Set this switch to the name of the file in which you wish to save informational messages from the ION Tunnel Broker. If no logfile is specified, messages will be written to the standard out.

-httpport

Set this switch to the port number that the ION HTTP Tunnel Broker should watch for connection requests. If you do not specify a value for the `-httpport` switch, the ION Tunnel Broker watches port 9085.

-httptimeout

Set this switch to the number of minutes the ION Tunnel Broker HTTP peer should stay alive for without hearing from the client. A timeout is necessary to close Tunnel Broker peer processes that may be left running if a browser crashes or experiences some other error that disconnects the browser without shutting down the peer process. If set to 0, the peer will never time out.

-logfile

Set this switch to the name of the file in which you wish to save informational messages from the ION Daemon. If no logfile is specified, messages will be written to the standard output. (Under UNIX, you can create a log file by redirecting the output from `iond` to a log file of your choosing using the normal system output redirection mechanism.)

-maxconn

Set this switch to the maximum number of connections that can be active at once. If you do not specify a value for the `-maxconn` switch, the maximum number of connections will be equal to the number of IDL licenses you have available.

-port

Set this switch to the port number that the ION Daemon should watch for connection requests. If you do not specify a value for the `-port` switch, the ION Daemon watches port 7085.

-rutil

Set this switch to allow the utility routines `iondown` and `ionstat` to be run from any host. By default, connections from these routines are allowed only if the routines are run on the same host as the ION Daemon.

-security

Set this switch to a comma-separated list of tokens that define a list of IDL routines. IDL routines specified via a token in the security list will not be passed through to the IDL session by the ION Server.

If you do not include the `security` switch when starting the ION Daemon, the following default tokens are set:

```
fileio, os, linking, device, df
```

If you include the `-security` switch when starting the ION Daemon, only the tokens you specify are set. See the discussion of the `-infile`, `-exfile`, `-incomm`, and `-excomm` switches for further information on specifying which IDL commands will be accepted by daemon.

The `-security` switch accepts the following tokens. (In the lists below, the asterisk is used to represent all IDL routines of a given type.)

Token	Description
df	Disables all Scientific Data Format routines (CDF_*, EOS_*, HDF_*, NCDF_*).
device	Disables changing devices using the `SET_PLOT` routine.
filein	Disables file input operations by disallowing use of the following routines: GET_KBRD, OPENR, READ, READF, READU, READ_*, TAPRD
fileout	Disables file output operations by disallowing use of the following routines: OPENW, PRINTF, TAPWRT, WEOF, WRITEU, WRITE_*
fileio	Disables file input and output operations by disallowing use of the following routines: ASSOC, CLOSE, EOF, FILEPATH, FLUSH, FSTAT, GET_LUN, IOCTL, OPENU, POINT_LUN, REWIND, SKIPF
linking	Disables calls from IDL to external code by disallowing use of the following routines: CALL_EXTERNAL, LINKIMAGE
none	No security checking is provided.

Table 1-6: Active Security Group Tokens

Token	Description
os	Disables operating system access by disallowing use of the following routines: CD, CALL_FUNCTION, CALL_METHOD, CALL_PROCEDURE, DEFINE_KEY, DELETE_SYMBOL, DELLOG, EXECUTE, FILEPATH, FINDFILE, GETENV, GET_SYMBOL, POPD, PRINTD, PUSHD, SETENV, SETLOG, SET_SYMBOL, SPAWN, TRNLOG

Table 1-6: Active Security Group Tokens (Continued)

-timeout

Set this switch to the number of seconds ION will wait to receive a response. If no response is received within the timeout interval, ION will make a second attempt (it will "ping" the remote machine). If no response is received within the second timeout interval, ION will close the connection.

The default timeout value is 60 seconds. You may wish to increase the timeout value with extremely slow network connections.

Starting the ION Daemon at Boot Time

You can automatically start the ION daemon by adding the command *RSI-DIR*`/ion_1.6/ion_java/bin/iond` to your system startup script, or by installing and configuring the `sys5_iond` boot time startup script, as described below:

Note

The following instructions may differ for your platform. For additional information, refer to your host operating system documentation or the man pages for `init`, `rc0`, `rc2`, and `rc3`.

- **Linux** — Using any text editor, add the ION daemon startup command, *RSI-DIR*`/ion_1.6/ion_java/bin/iond`, to the end of the `/etc/rc.d/rc.local` file.
- **Sun Solaris, SGI IRIX** — You must place a controlling script in a directory (usually `/etc/init.d` or `/sbin/init.d`) and create links to that script which runs at system startup and shutdown. A template for the controlling script can be found in the file *RSI-DIR*`/ion_1.6/ion_java/bin/sys5_iond`. This file contains instructions on how to customize this script for your system, copy the file to

the appropriate directory, and create the links that will automatically run the script at boot time.

Checking the Status of the ION Daemon

Use the `ionstat` utility to determine the current status of the ION Daemon or Tunnel Broker. The status report includes the start time of the daemon and information about clients currently connected to the ION Server.

The `ionstat` command uses the following syntax:

```
ionstat [-host=hostname] [-port=port]
```

The switches to the ionstat command are described below:

-host

Set this switch to the name of the host on which the ION Daemon or HTTP Tunnel Broker is running. Unless the `-rutil` switch was set when the ION Daemon was started, `ionstat` requests are only accepted from the host on which the daemon is running.

-port

Set this switch to the port number of the port that the ION Daemon or HTTP Tunnel Broker is listening. The default is port 7085 which is the default port for the ION Daemon. To check the status of the ION HTTP Tunnel Broker, you must specify the port on which the Tunnel Broker is listening (the default is 9085).

Shutting Down the ION Daemon

Use the `iondown` utility to shut down the ION Daemon or Tunnel Broker. The `iondown` command uses the following syntax:

```
iondown [-force] [-host=hostname] [-port=port]
```

Note

Under Windows, you will generally use the ION service rather than starting and stopping the ION Daemon manually. However, if you used the `iond` command to start the ION Daemon on your machine, you can use the `iondown` command to stop it.

The switches to the `iondown` command are described below:

-force

Set this switch to force the ION Daemon or HTTP Tunnel Broker to shut down without prompting. If `-force` is not specified, `iondown` will prompt you before shutting down the daemon.

-host

Set this switch to the name of the host on which the ION Daemon or HTTP Tunnel Broker is running. Unless the `-rutil` switch was set when the ION Daemon was started, `iondown` requests are only accepted from the host on which the daemon is running.

-port

Set this switch to the port number of the port that the ION Daemon or HTTP Tunnel Broker is watching. The default is port 7085 which is the default port for the ION Daemon. To shut down the ION HTTP Tunnel Broker, you must specify the port on which the Tunnel Broker is listening (the default is 9085).

Manually Configuring Your Web Server

If you skipped the “Web Server Configuration” step during installation, you will need create directories and copy files from the ION installation to your Web server directory after installing and configuring a Web server. Follow the steps for your platform.

On UNIX — Run the configuration script, `java_config`, located in the default installation directory, *`RSI-DIR`*`/ion_1.6/ion_java/bin`. This script will create directories and copy the required files to your Web server’s HTML files directory as well as configure the IDL_PATH.

On Windows — manually copy files from the ION distribution into the Web server’s HTML files directory as follows:

1. Create a directory named `IONJava` in the *`Web Server`*`\htdocs` directory (or the *`Web Server`*`\wwwroot` directory for IIS).
2. Copy `index.html` from the main ION installation directory, *`RSI-DIR`*`\IDL60\products\ION20\ion_java` directory to the *`Web Server`*`\htdocs\IONJava` directory.
3. Create a subdirectory named `classes` in the *`Web Server`*`\htdocs\IONJava` directory. Copy the following files from *`RSI-DIR`*`\IDL60\products\ION20\ion_java\classes` to the new directory, *`Web Server`*`\htdocs\IONJava\classes`:

   `ion_16.jar`

   `ion_16.zip`

   all `.class` files

   Note

   You do not need to copy the `com` directory, contained in the `classes` directory, or any of its subdirectories. All of these files are packaged into the `ion.jar` and `ion.zip` files.

4. Copy the entire `examples` directory, including all files and subdirectories, from *`RSI-DIR`*`\IDL60\products\ION20\ion_java` to the *`Web Server`*`\htdocs\IONJava` directory.

   When you finish, the main level `htdocs\IONJava` directory of your Web server will include two subdirectories, `classes` and `examples` and an `index.html` file.

5. Update your IDL search path to include the `Web Server/htdocs` directory. For more information, see "The ION Java Properties Dialog" on page 11.

Configuring The ION HTTP Tunnel Broker

The ION HTTP Tunnel Broker is a program that allows ION client applets (running in World Wide Web browsers) located behind a network firewall to communicate with an ION Server on the other side of the firewall. For more information about network firewalls in regards to the ION client/server model, see “ION HTTP Tunnel Broker” on page 41.

Note

Due to errors in virtual machine implementations, Java Applets may use SOCKS to open an HTTP connection. As such, the broker may fail with firewalls that do not support the SOCKS protocol.

Using the Tunnel Broker

Using the ION Tunnel Broker is very simple. On the server side, you must ensure that the ION Tunnel Broker is running; see “Starting the ION Tunnel Broker Daemon” on page 32 for details. On the client side, you have the option of specifying one of three connection types via the CONNECTION_TYPE parameter in an ION applet:

- HTTP_CON — Make only HTTP connections, using the The ION HTTP Tunnel Broker. These connections are typically slower than SOCK_CON.
- SOCK_CON — Make only socket connections, using only the ION Daemon.
- BEST_CON — Attempt to make a socket connection. If a socket connection is not possible, attempt to make an HTTP connection. This is the default setting.

Since “BEST_CON” is the default, you do not need to add the CONNECTION_TYPE parameter at all if you want your ION Server to accept either socket or HTTP connections. See “Parameters Specified via <PARAM> Tags” on page 72 for details on other parameters related to the ION Tunnel Broker.

Starting the ION Tunnel Broker Daemon

The ION Tunnel Broker Daemon must be running for ION to be able to use HTTP connections. There are three ways to start the ION Tunnel Broker:

1. On Windows systems, by using the ION Java Properties dialog. See the section "The Broker Tab" on page 18 for more information.
2. By specifying the `-http` switch to the `iond` command. With this method, both the ION Daemon and the ION Tunnel Broker are started at the same time. You can also specify the `-httpport` and `-httplog` switches to specify ION Tunnel Broker options. See "Configuring ION Java for UNIX" on page 22 for details.
3. Using the `ion_httpd` command at the command line.

The ion_httpd command uses the following syntax:

```
ion_httpd [-ionhost=hostname] [-ionport=port] [-port=port]
[-logfile=filename ] [-maxpeer=number_of_peers] [-timeout=minutes]
```

Command-line switches for the `ion_httpd` command are listed below:

-ionhost

Set this switch equal to the name of the host on which the ION Daemon is running. Note that the ION Tunnel Broker may be running on a different host than the ION Daemon and ION Server. Note that Java applet security requires that the Tunnel Broker be run on the same machine from which the ION Java classes were loaded.

-ionport

Set this switch equal to the port number on which the ION Daemon is listening. If not specified the ION Daemon listens on port 7085.

Note

The ION Tunnel Broker must be listening to a different port than the ION Daemon.

-port

Set this switch to the port number that the ION HTTP Tunnel Broker should watch for connection requests. If you do not specify a value for the `-httpport` switch, the default port is 9085.

-logfile

Set this switch to the name of the file in which you wish to save informational messages from the ION Tunnel Broker. If no logfile is specified, messages will be written to the standard output. (Under UNIX, you can create a log file by redirecting the output from `ion_httpd` to a log file of your choosing using the normal system output redirection mechanism.)

-maxpeer

Set this switch to the maximum number of ION Tunnel Broker peers that can be active at once. If set to 0 (the default setting), the maximum number of peers will be equal to the number of IDL licenses you have available.

-timeout

Set this switch to the number of minutes the ION Tunnel Broker HTTP peer should stay alive without hearing from the client. A timeout is necessary to close Tunnel Broker peer processes that may be left running if a browser crashes or experiences some other error that disconnects the browser without shutting down the peer process. If set to 0, the peer will never time out.

Command Security

The ION Server implements a security system based on IDL command filtering. The security system has two internal command lists: one list consists of commands that *are not* allowed to be run on the IDL server process; the other list specifies commands that are allowed. (If an IDL command is included in both lists, it will *not* be allowed to run.)

When an ION client sends an IDL command to the ION Server for execution, the command line is scanned for function and procedure names. These names are first checked against the command inclusion list (commands that can be run on the server), and if the command is not in the list it is rejected. If the command inclusion check passes, the routine is then checked against the command exclusion list (routines that should not be run on the server). If the command is in the command exclusion list, it is rejected. If the command passes the exclusion list check, it is sent to the ION Server process for execution.

Note

ION's command security configurations are designed to prevent IDL commands from being used in an unauthorized or hostile manner during connections to your ION Server. Remember that you must also properly configure your Web server to prevent unauthorized access to your site via other mechanisms.

Security Command Files

Using a text file, you can specify IDL commands to be included or excluded from the ION Server. Inclusion and exclusion text files consist of a single command on each line. Lines that are blank or start with the "#" character are ignored. For example, you could create an ION exclude file containing the following lines:

```
# Commands to prevent execution of
CALL_FUNCTION
XBM_EDIT
```

To use an include or exclude file see the following directions for your platform:

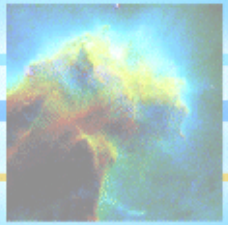
- On UNIX, start the ION Daemon using the `-infile` and `-exfile` command-line switches. See "Starting the ION Daemon on UNIX" on page 22.
- On Windows, see "The Security Tab" on page 16.

Client Verification

When the ION Daemon detects an incoming server connection, the daemon verifies that the client is a valid ION client. ION clients are valid if they have been created using the ION Java classes described in this document. If the client is not valid, the daemon rejects the connection and no ION Server process is started.

Connection Limit

There are two limits set on the number of connections the ION Server will accept. If you have specified a maximum number of connections via the `-maxconn` switch to the ION Daemon process, the ION Daemon will reject new clients after reaching that limit. If no maximum number of connections is specified to the daemon, the maximum number of connections allowed is defined by the ION Server license. If the limit is reached, the ION Daemon will notify new ION clients that the limit has been reached and will close the connection.



Chapter 2:
Overview

This chapter introduces ION Java and discusses the ION Java architecture, including the ION Service and Tunnel Broker Daemons. This chapter includes the following topics:

- What is ION Java?
- ION Java Architecture
- ION Java Limitations
- ION Java Performance Considerations
- Running the ION Java Examples
- Where to Place HTML and Class Files

What is ION Java?

ION Java is a sophisticated system that brings the power of IDL to the Internet. ION Java uses Java and Internet technology to deliver efficient data analysis and visualization capabilities to World Wide Web client applications. ION Java is ideal for organizations that have shared data that needs to be accessed and visualized by a wide variety of users. ION Java can be configured as part of a public Web server, a proprietary intranet server, or as both at the same time.

ION Java combines both IDL, the Interactive Data Language, and Java into a single, powerful tool for building Web-based applications. Both IDL and Java are cross-platform, interpreted languages. In contrast to Java, IDL is specifically designed for the visualization and analysis of large, multi-dimensional technical datasets. IDL is the language of choice for technical professionals, offering simple syntax, array-oriented architecture, and rich library of analysis and visualization routines. ION Java, ideal for client-server applications or Web-applets, gives Java developers the power to deploy their applications for data sharing and data analysis more rapidly.

ION Java allows access to IDL from virtually any computer in the world. Updating and maintaining ION is simple, since the product resides only on the server. Applets are sent to clients over the Web, as needed.

Recommended Skills

ION is designed to make it easy for you to create interactive Web pages or Internet/Intranet applications that use IDL. The following competencies are recommended for efficient ION Java application development:

Familiarity with Web Server Administration

Even if you do not maintain the World Wide Web server at your site, you should be aware of the configuration details. You will need to know where files should be located for server access, what file permissions are necessary, and any other site-specific details that apply to publishing HTML pages on the World Wide Web.

JAVA Programming Knowledge

If you wish to build your own applications or applets, you will need to be familiar with Java programming concepts. You will also need to know how applets are embedded in HTML pages.

Understanding of IDL

ION is designed to interact with IDL. To use ION, you will need to be familiar with IDL's basic command syntax and features.

ION Java Architecture

The components that make up ION Java are illustrated in Figure 2-1.

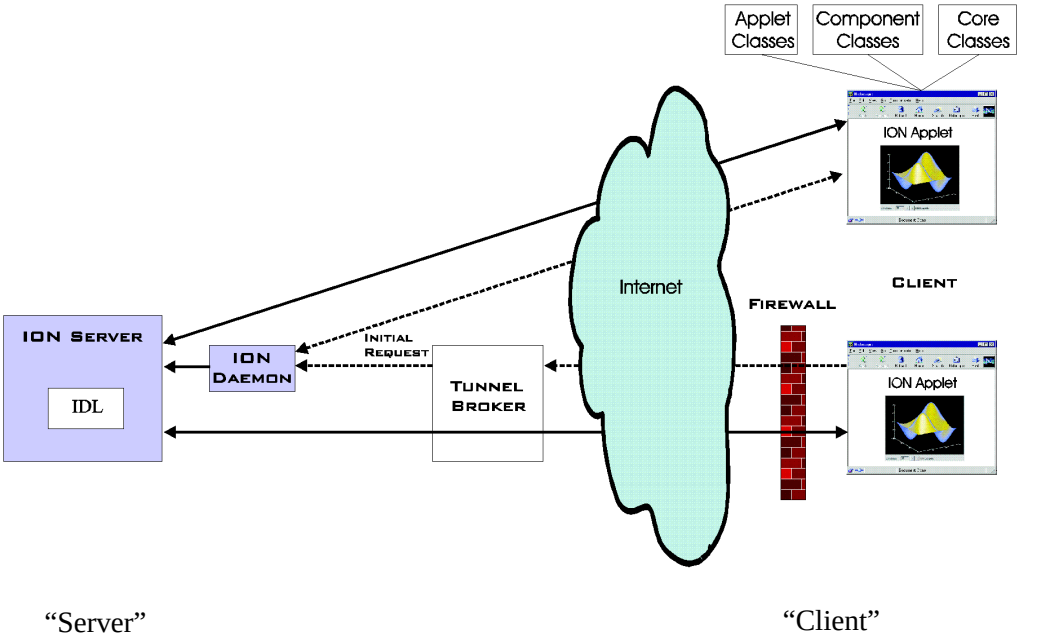


Figure 2-1: ION Java Architecture

ION Server

The ION Server is a program that manages communication between an ION client application (either a Java Applet running in a Web browser or a stand-alone Java application) and IDL. The ION Server translates requests from ION clients into commands that can be processed by IDL, and then passes output from IDL back to the client for display. The ION Server is discussed in detail in Chapter 1, "Configuring ION Java".

Once the incoming client has been verified by the ION Daemon, the ION Daemon starts an ION Server process and connects the client with the ION Server process. The ION Server process checks out an ION license and then begins command processing. The ION Server process is responsible for the following:

- Reading requests from the ION client,
- Performing security checks on the client request,
- Executing valid ION/IDL commands,
- Sending graphic information and data to the ION client.

Security Checks

Once a command is received from the client, the request is passed through the ION security system. Any security failure causes the command to be logged and an error condition to be sent to the client. If the command passes the security system, it is passed to IDL for execution.

Command Execution

When a command is executed, all graphic and command log information is sent to the client. Once the command is completed the error status is sent to the client and the ION Server process waits for the next request.

ION Daemon

The ION Daemon is a program that makes the initial connection between an ION client and the ION Server. The ION Daemon "watches" a specific port on the ION Server's host computer. When the daemon receives a request for connection, it performs basic security screening before connecting the ION client to the ION Server. The ION Daemon is discussed in detail in Chapter 1, "Configuring ION Java".

The ION Daemon is responsible for the following:

- Parsing command line parameters
- Establishing the security level and initializing security levels
- Maintaining server logs
- Managing the number of current connections
- Receiving connections and starting ION Server processes
- Verifying incoming requests as valid ION clients

ION HTTP Tunnel Broker

The ION HTTP Tunnel Broker is a program that allows ION client applets (running in World Wide Web browsers) located behind network firewalls to communicate with

an ION Server on the other side of the firewall. The Tunnel Broker is discussed in detail in “Configuring The ION HTTP Tunnel Broker” on page 31.

Network *firewalls* work by isolating a network from the Internet as a whole and allowing only pre-specified network operations to take place. In many cases, this means that traffic between an internal network and the Internet must go through a single computer, which allows connections of specified types and denies other connections. Firewalls allow computers and data on the “inside” to be relatively safe from intrusion by outsiders, while allowing the inside computers to make connections with computers on the Internet via a set of relatively limited protocols, such as HTTP and FTP.

The ION client/server model is based on a persistent two-way socket connection between the client and server. Firewalls, in most cases, do not allow arbitrary processes to open socket connections to remote servers. Because ION communication is not based on any standard protocol, it may not be able to penetrate a firewall that allows connection through only the standard and well-known protocols (HTTP, FTP).

Freestanding ION Java applications running behind a firewall have little chance of obtaining a connection through the firewall. However, ION applets running in a web browser can take advantage of the HTTP connections provided by the browser and use them to “tunnel” through the firewall and communicate with an external ION Server. The ION HTTP Tunnel Broker provides all the functionality necessary for ION Applets to successfully tunnel across most firewalls.

The Tunnel Model

The ION Tunnel model includes an HTTP communications layer on the ION Client that maintains a connection to an ION HTTP Tunnel Broker. The Broker manages a set of *peers* that communicate with and control ION Servers. HTTP requests sent from the client to the server are dispatched to the appropriate peer and peer responses are sent back to the client through the Broker in the form of an HTTP response.

Normally, when an ION client makes a connection, the ION Daemon sets up a direct socket connection between the client and an ION Server process.

In the Tunnel model, ION clients connect to the ION Tunnel Broker rather than to the ION Daemon. The Tunnel Broker requests that the ION Daemon start an ION Server process, and attaches a peer to the server. The peer then acts as the client for communication with the ION Server. The peer buffers server responses, packs them into HTTP messages, and sends them through the Tunnel Broker back to the ION client.

Pre-Built ION Client Applets

The ION package includes a set of pre-built Java applets. The pre-built applets allow you to begin using ION immediately, without the need to write Java code. See Chapter 4, “Using ION’s Pre-Built Applets” for details.

ION Component Classes

The ION Component classes provide a simple, straightforward interface that allows you to create ION client applets and applications quickly and easily. While using the ION Component classes does require that you write Java code, the classes handle most of the details of writing applications to interact with IDL seamlessly. See Chapter 3, “Overview of the ION Java Classes” for details.

ION Low-Level Classes

The ION low-level classes are the backbone of the ION Java system; they provide the tools a professional Java programmer needs to create robust applications to interact with IDL. The ION Component classes and the ION pre-built applets are both built directly from the ION low-level classes. See “ION Low-Level Classes” on page 57 for details.

ION Java Limitations

Server Limitations

If the server is behind a firewall, you can use the ION Tunnel Broker which uses HTTP Protocol to pass ION information through the firewall. (See “Configuring The ION HTTP Tunnel Broker” on page 31 for details on the Tunnel Broker.)

IDL Limitations

The following IDL features are unavailable with ION Java:

- IDL Widgets
- The IDL line continuation character, $

All of IDL’s analytical routines and all of the IDL Direct Graphics and Object Graphics routines are available, subject to the constraints imposed by the ION security mechanism. (See“Command Security” on page 34 for more on ION’s security mechanism.)

ION Java Performance Considerations

There are several issues which impact ION Java performance. While steps can be taken to improve performance (see "Tips for Increasing Execution Speed in ION Java" on page 45), many users note that execution of ION Java applications are slower than equivalent applications executed in IDL. Also, IDL commands called from an applet execute more slowly than IDL command line execution. Performance can also differ between client platforms.

Extra communication layers are necessary when executing an IDL command in ION Java and then displaying the results. When an IDL command is called from an ION Java applet or Java application, the following required steps impact execution time:

1. The Java Virtual Machine interprets the Java code.
2. The initial connection to the ION Server initiates an IDL session.
3. The client browser running the Java applet sends requests to the ION Server. Network traffic and bandwidth affects transmission rates.
4. The ION Server translates each request into commands that can be processed by IDL. This may involve security command screening.
5. IDL interprets and executes the commands.
6. The ION Server returns output from IDL to the client for display.
7. The Java application draws graphic primitives received from the ION Server in the browser. Java drawing routines are slower than raw UNIX motif or Windows GDI devices. The drawing time can even vary between Java Virtual Machines.

Note

Using the same machine as both the client and the server can further degrade ION Java performance. Although network traffic is not an issue, communication must still be routed through sockets and HTTP. Depending on the server machine specifications, the extra resources required for context switching between the server and the browser may hinder performance.

Tips for Increasing Execution Speed in ION Java

The following items can increase the execution speed of ION Java applets and applications:

Package Multiple IDL Commands into a Single .pro File

It is always more efficient to package multiple IDL commands into a single .pro file than to call individual commands. With individual commands, the communication layer must be transversed for each command. With a a single package of commands, the communication layer is transversed only once. An example is included in the "Advanced" section of the ION Examples. See "Running the ION Java Examples" on page 48.

Convert TrueColor Images

TrueColor (24-bit) images are three times as large as indexed (8-bit) images. While ION Java is capable of displaying 24-bit TrueColor images, you can speed up ION Java execution by converting 24-bit images to 8-bit images. To do so, use the IDL COLOR_QUAN function before displaying the image. By decreasing the image size, this significantly reduces the transfer time necessary to display a graphics primitive sent from the server to the client. Related considerations include clients who may not have displays configured to display 24-bit images and browsers which automatically dither images. An example is included in the "Basic" section of the ION examples. See "Running the ION Java Examples" on page 48.

Send Complex Plots as a Single Image

ION Java sends graphics primitives to the client to be drawn by Java. More complicated plots transfer more graphics primitives and take a longer time to be drawn. You can decrease the amount of information sent to the client and time required to draw complicated plots by doing one of the following:

- Render the plot to an off-screen pixmap in IDL and then use DEVICE, with the COPY keyword to capture the image.
- Use the z-buffer in conjunction with an off-screen pixmap and then capture the image using TVRD in conjunction with TV.

These methods send a single image to the client to be drawn. For very complicated plots, this can be more efficient. For simple plots, however, this could increase the amount of data that is sent to the client so using the default graphics primitives may be more efficient. An example is included in "Advanced" section of the ION Examples. See "Running the ION Java Examples" on page 48.

Bandwidth Issues

Because ION Java applications can be image-intensive, their performance depends strongly on network bandwidth. Bandwidth may not be an issue if you are serving your ION Java applications only to the users of your high-speed company intranet,

but if your users are likely to be accessing your application over the Internet, through an analog telephone line and low-speed modem, then close consideration must be given to the size of data transferred to and from the ION Server. For example, if your application allows the user to zoom in on a region of interest, then you could provide the smallest, lowest-quality image necessary to give the user the required information at each stage in the drill-down process.

Avoid Using Device Fonts

When a TrueType font is rendered in IDL, it is sent to the device as a set of polygons. Depending on the symbol being rendered, the number of polygons generated can be quite large which can increase download times to client machines. If you use hardware fonts, the amount of data being sent to the client can be decreased in certain situations since only the attributes and parameters of the fonts are being sent. Another workaround is to render the graphic before sending it to the client. See “Send Complex Plots as a Single Image” on page 46 for more information.

Running the ION Java Examples

Once you have the ION Server properly configured and started, you are ready to run the example applications. Several example applications are placed in your Web server directory during the ION Java installation process.

Note

If you skipped the step which provided the location of your Web server's HTML files directory during the installation process, you will need to copy the appropriate files to your Web server before running the examples. See "Manually Configuring Your Web Server" on page 29 for instructions.

The examples illustrate ION features and many of the examples allow you to view the Java source within your browser. These examples consist of at least two types of files: HTML files that contain the Java applets, and the Java applets themselves, which are contained in .class files. The raw Java source files for the example ION Java classes are included in the `src` subdirectory of the `examples` directory. Also included in the `examples` directory are a number of IDL `.pro` files that are called by the ION demonstration applets.

To run the ION Java examples, complete the following steps:

1. Add the ION Java `examples` directory to IDL's Search path (note that this step may have been completed during the installation of ION Java if you completed the "ION Java Web Server Configuration" dialog):

   On Windows, add *`Web_Server`*`\IONJava\examples` to the IDL Search path. For example, using the Apache web server, this might be `C:\Program Files\Apache Group\Apache2\htdocs`.

   On UNIX, add *`Web_Server`*`\IONJava\examples` to the IDL Search path. For example, using the Apache web server, this might be `usr\local\apache2\htdocs`.

   `Web_Server` is the path specifying the location of the Web server's HTML files directory.

   For more information about modifying the IDL Search Path, see "Setting the IDL Path" on page 293.

2. Open your browser and enter the following URL.

   ```
   http://hostname/IONJava/index.html
   ```

   where *hostname* is your qualified domain name or machine name. This loads a page containing ION Java basic and advanced applet links as well as a link to Research System's ION web site.

3. For a simple example, select the Basic ION Java Applets link and choose "Simple Plot". Click "View Source Code" to see the code required for this applet.

The ION Java examples provide many samples of Web-based Java applications. The examples have been divided into three levels:

- **Basic ION Java Applets** — illustrate simple ION Java concepts that are necessary to understand before building your own applets. You can examine the source code of each basic example to better understand the implementation of simple concepts. Three categories of basic ION Java examples include pre-built applets, component classes and low-level classes.
- **Advanced ION Java Applets** — illustrate advanced programming concepts in ION Java. You can examine the source code and even use the applets as building blocks for your own applets. Three categories of basic ION Java examples include pre-built applets, component classes and low-level classes.
- **ION Online Demos** — display interactive ION Java web applications. A link is provided to Research System's ION site where you can view applications that exemplify the power of ION Java.

When you are ready to develop your own applications, see the following section, "Where to Place HTML and Class Files" on page 51 for strategies on where to store the files required for your applications.

Note

If you are using Internet Explorer, you must access any HTML page that calls an applet by specifying a URL. Attempting to open such a page using the browser's **File** → **Open** command or by double-clicking on an .html file fails to display the applet and results in security errors. Use a URL that contains http:// rather than file://.

Note

The source for a freestanding Java Console application has also been provided. See the `getversion.java` file in the *`RSI-DIR`*`\idl60\products\ion20\ion_java\examples\src` directory (Windows) or *`RSI-DIR`*`/ion_2.0/ion_java/examples/src` directory (UNIX). You can compile and run this example.

Where to Place HTML and Class Files

When you begin developing your own applications, you'll need to decide where you will put the HTML and Java class files that make up your applications. This section discusses some strategies for locating your files.

Web-based ION Java applications consist of at least two types of files:

- HTML files — are the containers for your Java applets
- .class files — are containers for the Java applets themselves

HTML files must reside on your Web server, which may or may not be the same machine on which the ION Server is located. Your class files, however, must reside on the same host machine as the ION Server. This is due to Java applet security mechanisms.

Testing ION Applications Locally

When learning how to write ION applications, and running the example applications included with ION Java, you may find it easier to load the applications directly from the ION Server machine rather than placing the files on your Web server and loading them over a network. This allows you to run the example applications right from one of the examples directories of your ION Java installation, and makes the process of developing and testing your applications easier. This also takes the Web server out of the loop, thereby eliminating the Web server as a potential source of application errors. If you run the example applet applications directly from the *RSI-DIR*`\IDL60\products\ION20\ion_java\examples` directory (Windows) or *RSI-DIR*`/ion_1.6/ion_java/examples` directory (UNIX), you do not need to change the CODEBASE attribute for any of the <APPLET> tags.

Note

Certain browsers may generate a Java security exception when attempting to start an applet contained in an HTML file opened by selecting "Open" from the File menu. This exception prevents the applet from running. To work around this exception, browse to the `basic.html` file using a URL that looks like http:// rather than file://.

Publishing ION Applications on Your Web Server

Once you have developed your ION applications, you will need to place the HTML files on your Web server. The recommended method is to create a subdirectory for

ION applications under the default documents directory on your Web server. For example, suppose you are using the Apache Web server. You could create a subdirectory under the `htdocs` directory called `\IONJava\myhtml` in which you place all your HTML files. The URL of such a page might be:

```
http://myhost.mydomain.com/myhtml/index.html
```

You can then develop and test your applications locally. When everything is working correctly, you can publish your application by copying the `myhtml` directory containing your HTML files, and your .class files to your web server.

If you do not place HTML files in a directory that is in or under the default documents directory on your Web server (such as the `htdocs` directory on the Apache Web server), you need to configure your Web server to allow access to files in your directory. For example, if you place your HTML pages in a directory called `C:\rsi\IDL60\products\ion_java\html`, you need to configure your Web server to allow access to this directory. Using Apache, modify `DocumentRoot` in the `httpd.conf` file to include additional directories or modify `Alias` in the `httpd.conf` file to add aliases to directories where you can place HTML files.

Where to Locate the ION Class Files

ION applets and applications must have access to the ION class files in order to run. There are two ways to provide access to the ION class files:

- Use the <APPLET> tag's CODEBASE attribute to point to the directory that contains the required classes.
- Place the required class files in the same directory that contains the HTML page that loads the applet.

Placing class files and HTML files in the same directory saves you from having to use the CODEBASE attribute, but we recommend that you point to the .jar or .zip files, described in the following section, and create separate directories for class and HTML files.

For example, suppose you are using the Apache Web server. You could create a subdirectory under the `htdocs` directory called `\IONJava\classes` in which you place all the required class files. Assuming that your HTML files are in the `\IONJava\html` directory, you would specify the CODEBASE attribute as follows:

```
CODEBASE="../classes"
```

See "CODEBASE" on page 69 for further details.

What Are the Required Class Files?

During installation, ION class files are installed in the following location on the ION Server machine:

Windows:

```
RSI_DIR\IDL60\products\ION20\ion_java\classes\
```

UNIX:

```
RSI_DIR/ion_1.6/ion_java/classes/
```

where *RSI_DIR* is the location of the RSI directory on your system.

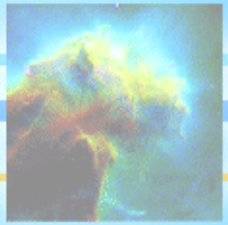
The ION installation program also copies these files to your Web server's java files directory if you completed the "ION Java Web Server Configuration" dialog. For example, using the Apache Web server on Windows, the `Web_Server` directory might be similar to `C:\Program Files\Apache Group\Apache\htdocs`.

ION's Java class files are provided in three formats:

1. The raw Java class files are located in the `com/rsi/ion` subdirectory of the `classes` directory.
2. A ZIP file named `ion_16.zip`. This contains compressed versions of all the classes. This file is installed in the `classes` directory and is also copied to your `Web_Server/IONJava/classes` directory.
3. A Java archive (JAR) file named `ion_16.jar`. This file contains uncompressed version of the ION class files. This file is installed in the `classes` directory an is also copied to your `Web_Server/IONJava/classes` directory.

If you decide to create your own directory for the ION Java classes, you will need to copy the `classes` directory files specified in the section "Manually Configuring Your Web Server" on page 29 to the new directory. Copying the `com` directory and subdirectories to your new directory is optional.

For more on the ION Java class files, see "Supporting Java Archive Files" on page 103.



Chapter 3:
Overview of the ION Java Classes

This chapter provides a high-level overview of the Java classes that make up ION Java. The following topics are covered in this chapter:

- The ION Java Class Hierarchy
- AWT vs. Swing
- Using the Component Classes

The ION Java Class Hierarchy

ION Java consists of three levels of Java classes:

- ION Low-level classes
- ION Component classes
- ION Pre-built Applet classes

The relationship between the ION Java classes is illustrated in Figure 3-1 in the Unified Modeling Language (UML).

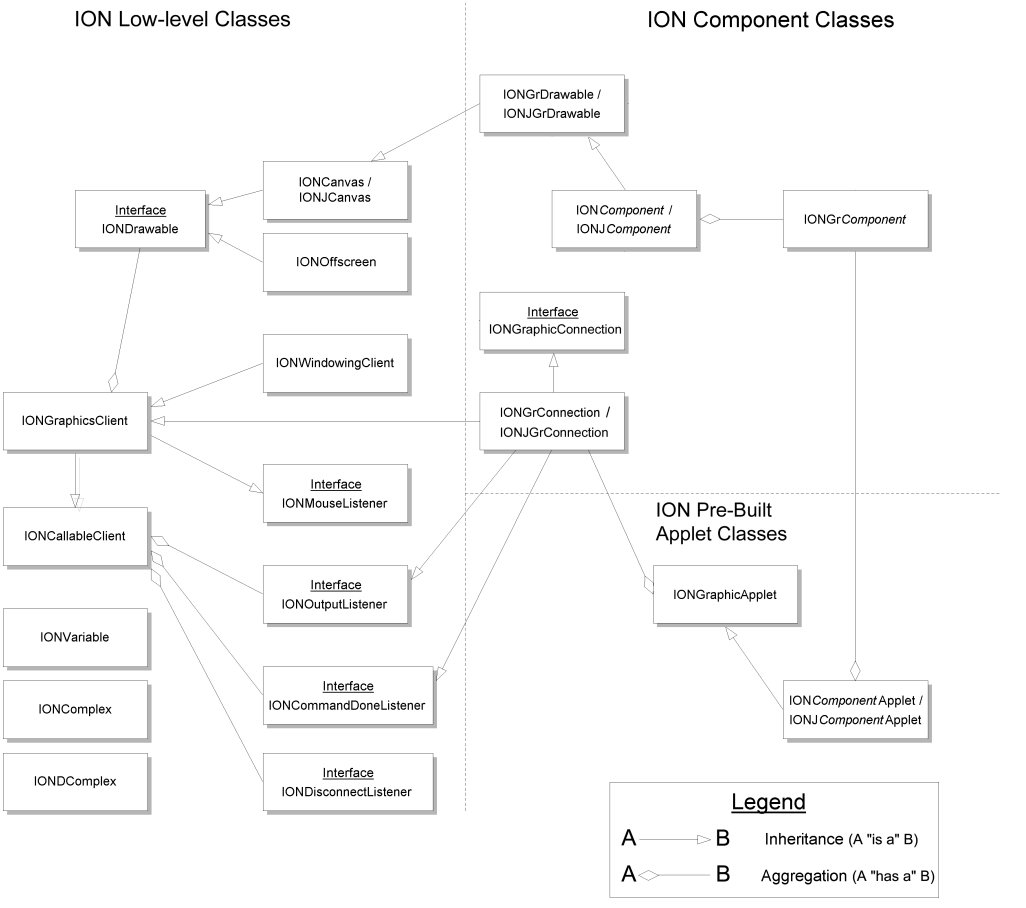


Figure 3-1: The ION Java Classes

Note

In Figure 3-1, classes that contain *Component* represent Contour, Map, Plot, and Surface. For example, ION*Component* represents the IONContour, IONMap, IONPlot, and IONSurface classes.

ION Low-Level Classes

The ION low-level classes are the most basic building blocks of ION applications. The ION low-level classes are the only classes required to build ION applications that contain IDL data and graphics output. All other ION Java classes are built on top of these low-level classes. The ION low-level classes provide a degree of control not available with the ION Graphics Component classes, but require more sophisticated Java and IDL programming skills. Each of the ION low-level classes is described below:

IONCallableClient

IONCallableClient provides mechanisms to handle communication with the server, execution of IDL commands, retrieving IDL command log output and the getting and setting of IDL variables on the ION Server. IONCallableClient is the only class required to write a non-graphical ION application.

IONGraphicsClient

This class provides mechanisms to handle the processing of graphic primitive data sent from the ION Server. Information sent by the server is read by mechanisms provided by the parent class IONCallableClient.

IONWindowingClient

This class provides mechanisms to handle the processing of the windowing commands that are part of an IDL Direct Graphics driver. This includes the creation, deletion, showing, hiding, and iconization of windows on the client.

IONDrawable

This interface defines the methods that an object must implement to act as an ION drawable object. An ION drawable is an object that can be drawn to by an IONGraphicsClient. This interface is implemented as either an IONCanvas or IONOffScreen object.

IONGR2Drawable

This interface defines the methods required for a class to be a drawing area for Object Graphics.

IONCanvas / IONJCanvas

These classes represents a visible drawing area upon which graphics can be displayed. They implement the ION Drawable interface.

IONOffScreen

This class represents an undisplayed drawing area on which graphic output can be placed. This implements the ION Drawable interface

IONCommandDoneListener

This interface defines the methods a class must implement to register and to receive notification that an IDL command has completed.

IONMouseListener (deprecated)

This interface defines the callback methods that a class must define to be notified of mouse events occurring on an object that implements the IONDrawable interface. This interface is deprecated in ION 1.4. It is recommended that you use the more robust Java MouseListener and/or the Java MouseMotionListener.

IONOutputListener

This interface defines the methods that a class must implement to receive ION Server output text.

IONVariable

This class is a client side representation of an IDL variable. IONVariable objects are used to read and write data between the IDL server and clients.

IONComplex

This class is the client side representation of IDL's single-precision complex number.

IONDComplex

This class is the client side representation of IDL's double-precision complex number.

ION Component Classes

The ION Component classes are a set of high level Java classes that provide a rapid and powerful way to include IDL graphics in a Java application or Java applet. Built

on top of the low-level classes, the Component classes encapsulate specific IDL functionality and provide a simpler interface, which allows you to connect to the ION Server and display graphics generated by IDL. The component classes are easier to use than the low-level classes, while providing less flexibility.

IONGrConnection

An IONGrConnection object provides a connection between the ION Server and the client. In addition to establishing and ending the connection, IONGrConnection allows you to get and set the values of IDL variables on the ION Server, add and remove drawable objects to the connection, and execute IDL commands directly. It also logs server messages automatically, and displays them via IONGrDrawable/IONJGrDrawable.

IONGrDrawable / IONJGrDrawable

An IONGrDrawable object creates a drawing area that presents graphics produced by the ION Server. IONGrDrawable allows you to configure the drawing area to draw one or more objects, add and remove graphic objects from a drawable, and execute IDL commands directly. An IONGrDrawable also contains a debug window. Objects of this type can be inserted into the AWT tree. The IONGrDrawable class is the AWT implementation, and IONJGrDrawable is the Swing implementation.

IONContour / IONJContour

An IONContour object represents a contour graphic and a drawing area. IONContour allows you to get and set properties of the contour (via keywords to the IDL CONTOUR routine) and to draw the contour object. IONContour extends IONGrDrawable and includes an IONGrContour object. IONContour is the AWT implementation, and IONJContour is the Swing implementation.

IONMap / IONJMap

An IONMap object represents a map graphic and drawing area. IONMap allows you to get and set map properties (via keywords to the IDL MAP_SET procedure) and to draw the map. Several other classes can be used with IONMap, including IONGrMapContinents, IONGrMapGrid, and IONGrMapImage. IONMap extends IONGrDrawable and includes an IONGrMap object. IONMap is the AWT implementation, and IONJMap is the Swing implementation.

IONPlot / IONJPlot

An IONPlot object represents a plot and a drawing area. IONPlot allows you to get and set properties of the plot (via keywords to the IDL PLOT routine) and to draw the

plot object. IONPlot extends IONGrDrawable and includes an IONGrPlot object. IONPlot is the AWT implementation, and IONJPlot is the Swing implementation.

IONSurface / IONJSurface

An IONSurface object represents a surface graphic and a drawing area. IONSurface allows you to get and set properties of the surface (via keywords to the IDL SURFACE routine) and to draw the surface object. IONSurface extends IONGrDrawable and includes an IONGrSurface object. IONSurface is the AWT implementation, and IONJSurface is the Swing implementation.

IONGrGraphic

An IONGrGraphic object provides methods used to manage graphic properties. The other IONGr objects extend this object. IONGrGraphic allows you to get and set graphic properties, and to manage property lists for the graphic object.

IONGrContour

An IONGrContour object is a property manager for a contour graphic. IONGrContour allows you to get and set properties of the contour plot via keywords to the IDL CONTOUR procedure, but does not contain a drawing area. (Use IONContour if you want a contour an object *and* a drawing area managed by a single object.) IONGrContour extends IONGrGraphic. The IONGr* components are useful for overlaying graphics on top of one another.

IONGrMap

An IONGrMap object is a property manager for a map graphic. IONGrMap allows you to get and set the properties of the map via keywords to the IDL MAP_SET procedure, but does not contain a drawing area. (Use IONMap if you want a map object *and* a drawing area managed by a single object.) IONGrMap extends IONGrGraphic. The IONGr* components are useful for overlaying graphics on top of one another.

IONGrMapContinents

An IONGrMapContinents object allows you to get and set properties of map outlines such as continental and political boundaries, coastlines, and rivers.

IONGrMapGrid

An IONGrMapGrid object allows you to get and set properties of map grids to be drawn on a map projection.

IONGrMapImage

An IONGrMapImage object allows you to get and set properties of images to be projected onto a map projection.

IONGrPlot

An IONGrPlot object is a property manager for a plot graphic. IONGrPlot allows you to get and set properties of the plot via keywords to the IDL PLOT procedure, but does not contain a drawing area. (Use IONPlot if you want a plot object *and* a drawing area managed by a single object.) IONGrPlot extends IONGrGraphic. The IONGr* components are useful for overlaying graphics on top of one another.

IONGrSurface

An IONGrSurface object is a property manager for a surface graphic. IONGrSurface allows you to get and set properties of the surface via keywords to the IDL SURFACE procedure, but does not contain a drawing area. (Use IONSurface if you want a surface object *and* a drawing area managed by a single object.) IONGrSurface extends IONGrGraphic. The IONGr* components are useful for overlaying graphics on top of one another.

ION Pre-Built Applets

The ION Pre-Built Applets allow you to interact with the ION Server with a minimum of Java knowledge or experience. Because the applets are pre-built, you can include them in Web pages using only HTML code.

IONGraphicApplet

The IONGraphicApplet is a general purpose applet that is used to execute a series of IDL commands and display the results.

IONContourApplet

The IONContourApplet displays an IDL contour plot. The X, Y and Z values of the plot and most IDL Contour properties supported by ION can be set through parameters to the applet.

IONMapApplet

The IONMapApplet is an applet that displays 2D data on a map projection. The data can be displayed as an image or a contour plot and can contain latitude/longitude grid lines, and landmass and political boundaries. The applet is capable of projecting multiple contour plots, one image, latitude/longitude grid lines, and boundaries onto the drawing area.

IONPlotApplet

The IONPlotApplet displays an IDL plot. The X and Y values of the plot and most IDL plot properties supported by ION can be set through parameters to the applet.

IONSurfaceApplet

The IONSurfaceApplet displays an IDL Surface plot. The X, Y and Z values of the plot and most IDL Surface properties supported by ION can be set through parameters to the applet.

Using the Component Classes

The ION Component classes have a number of common features. The contour, map, plot, and surface objects all allow you to set the data values, retrieve and set properties, and draw the object. See Chapter 6, "ION Java Class and Method Reference" for a complete list of methods for each class.

Setting Values

The ION Graphics objects that include data all allow you to set the initial data values when you create the object. You can also reset the data values using the `setXValue` / `setYValue` / `setZValue` methods. The `set` methods enable you to change the value of the displayed data on the fly without re-creating the object in question.

Getting and Setting Properties

The contour, map, plot, and surface objects can all be modified by changing the value of a set of properties associated with the objects. The list of properties available for modification is a subset of the list of properties controlled by keywords to the corresponding IDL Direct Graphics routine (CONTOUR, MAP_SET, PLOT, or SURFACE). Consult the IDL Reference Guide for details about the settings for individual properties.

Drawing

With the exception of the IONGrConnection object, all of the ION component objects have a draw() method. Calling the draw() method on a given object causes it to be displayed in the associated drawing area.

AWT vs. Swing

Each ION Java component is shipped in two forms—one built on AWT classes, the other on Swing classes. This section discusses the difference between AWT and Swing, the advantages and disadvantages of each, and how to distinguish between the ION AWT classes and the ION Swing classes.

AWT and Swing are both part of a group of Java class libraries called the Java Foundation Classes (JFC). The Abstract Windowing Toolkit (AWT) is the original GUI toolkit shipped with the Java Development Kit (JDK). The AWT provides a basic set of graphical interface components similar to those available with HTML forms. Swing is the latest GUI toolkit, and provides a richer set of interface components than the AWT. In addition, Swing components offer the following advantages over AWT components:

- The behavior and appearance of Swing components is consistent across platforms, whereas AWT components will differ from platform to platform.
- Swing components can be given their own “look and feel”.
- Swing uses a more efficient event model than AWT, therefore, Swing components can run more quickly than their AWT counterparts.

On the other hand, Swing components can take longer to load than AWT components.

ION Applications should use either all AWT-based components, or all Swing-based components. Mixing AWT and Swing components in the same application can cause problems with the stacking order of your components.

The ION Swing components can be identified by a “J”. For example, the Swing version of the IONPlot class is called IONJPlot.

Note

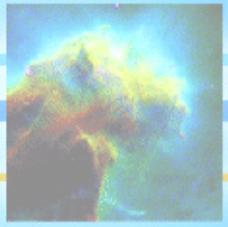
If you use Swing components, you need to define the certain attributes in your HTML file. This is due to certain browsers not supporting Swing components. For example:

```
CODE=com.rsi.ion.IONJPlotApplet.class
CODEBASE="../classes"
ARCHIVE="ion_16.jar, swingall.jar"
```

If you are not running your Java application through a browser, you need to set

CLASSPATH to include the `swingall.jar` file.

The swingall.jar file is available from `http://java.sun.com`.



Chapter 4: Using ION's Pre-Built Applets

The simplest way to create an ION Java application is to plug existing ION Java applets into a Web page. The pre-built applets included with ION Java allow you to interact with the ION Server with a minimum of Java knowledge or experience. Because the applets are pre-built, you can include them in Web pages using only HTML code. This chapter discusses the <APPLET> and <PARAM> tags, describes how to set up and customize each of ION's pre-built applets, and provides example code.

Note

You can use the examples in the chapter directly in your own Web pages by specifying the appropriate host and port settings for your server, and by specifying the CODEBASE attribute to reflect the location of the ION class files.

The <APPLET> Tag

The HTML <APPLET> tag is used to include Java applets in your HTML code. For more information on embedding applets into a web page, consult an HTML manual The syntax of the <APPLET> tag is as follows:

```
<APPLET
        [ALIGN={"left" | "right" |"top" | "middle" | "bottom"}]
        [ALT="alternate text"]
        [ARCHIVE="zip or jar file"]
        CODE="class file"
        [CODEBASE="path or URL"]
        HEIGHT="height"
        [HSPACE="pixels"]
        [NAME="name"]
        [VSPACE="pixels"]
        WIDTH="width" >
</APPLET>
```

Attributes

The <APPLET> tag takes the following attributes:

ALIGN

ALIGN specifies either the position of the applet in relation to the left and right borders of the browser, or the alignment of text in relation to the applet:

- LEFT - The applet is aligned with the left border of the browser.
- RIGHT - The applet is aligned with the right border of the browser.
- TOP - Text to the left and right of the applet is aligned with the top edge of the applet.
- MIDDLE - Text to the left and right of the applet is aligned with the vertical midpoint of the applet.
- BOTTOM - Text to the left and right of the applet is aligned with the bottom edge of the applet.

ALT

The ALT attribute specifies a text string to be displayed if for some reason the applet cannot be loaded. The ALT attribute is not required, but consider adding something

like the following to your applet description to enhance the user-friendliness of your HTML page:

```
ALT="ION Applet failed to load. Is Java enabled in your browser?"
```

Note

If you include HTML-formatted text within your <APPLET> tag, it will be displayed only if the Java Virtual Machine fails to start. This is slightly different from the ALT attribute, which contains text to be displayed only if the Java applet fails to load.

ARCHIVE

The ARCHIVE attribute is not required. However, it is recommended that you download all of the ION classes as a single package. See "Supporting Java Archive Files" on page 103 for a discussion of Java archive files.

CODE

A string specifying the name of the applet class. The CODE attribute should specify the *fully-qualified* class name relative to the directory in which the HTML file is located. If the CODEBASE attribute is included, the class name specified in the CODE attribute should be relative to the directory specified by CODEBASE.

For example, if you were to place an HTML file that used the IONPlotApplet in an `html` subdirectory of the ION directory, the CODE, CODEBASE and ARCHIVE attributes would be:

```
CODE=com.rsi.ion.IONPlotApplet.class
CODEBASE="../classes"
ARCHIVE="ion_16.jar"
```

because the `IONPlotApplet.class` file is located in the `com/rsi/ion` subdirectory within the ION `.jar` file. Similarly, if you were to place all of the Java class files necessary for your applet in the directory containing your HTML files, you could omit the CODEBASE attribute and use something like the following:

```
CODE=MyApplet.class
```

The CODE attribute is required for all ION applets.

CODEBASE

The CODEBASE attribute is not strictly required, but is often useful. The Java class loader searches for the contents of the `classes` directory in *current directory* — that is, the directory from which the HTML page containing the <APPLET> tag was loaded. If you locate the HTML page somewhere other than the `IONJava/classes`

directory, you will need to set the CODEBASE attribute to the *relative* path from the page location to the `classes` directory, or to a URL that specifies the location.

For example, if your HTML page is located in a directory called `/rsi/IDL60/products/ION20/ion_java/html`, you would set the CODEBASE attribute as follows:

```
CODEBASE="../classes"
```

Note

If the CODEBASE attribute is set equal to a URL, then the host specified by the URL can be used for ION network connections, but the host that is serving the HTML page cannot. This allows you to set up the ION Server and all of the ION class files on a machine separate from your web server, provided you include the SERVER_NAME parameter with the same hostname as in the CODEBASE URL. If you use this method, both the CODEBASE and SERVER_NAME attributes must refer to the same machine or Java security errors will result. In addition, the ION Server machine will still need to run a web server, but it will only be used to get the `.class` (or archive) files for the applets.

HEIGHT

The height of the applet in pixels. ION uses the HEIGHT attribute when creating the drawing area. This attribute is required for all ION applets.

HSPACE

The amount of white space to the left and right of the applet, in pixels.

NAME

A string containing a unique name for the applet. The string should be enclosed in double quotes marks. This attribute is required for all ION applets.

WIDTH

The width of the applet in pixels. ION uses the WIDTH attribute when creating the drawing area. This attribute is required for all ION applets.

VSPACE

The amount of white space on the top and bottom of the applet, in pixels.

Example

The following <APPLET> tag creates an applet of the IONGraphicApplet class, with a drawing area 100 pixels by 100 pixels, with the name "MyApplet." The HTML page

containing the applet code is assumed to be located in the directory `/rsi/IDL60/products/ION20/classes`, so no CODEBASE attribute is included.

```
<APPLET NAME="MyApplet" WIDTH=100 HEIGHT=100
    CODE=com.rsi.ion.IONGraphicApplet.class
<!- Other applet code >
</APPLET>
```

Supporting Java-Incapable Browsers

You can include HTML text within an applet tag, but the text will only be displayed if the Java virtual machine fails to start. You may find it useful to include something like the following:

```
<APPLET attributes>
   <!-- Applet code -->
   <B>Java virtual machine failed to start.
   Is Java enabled in your browser? </B>
</APPLET>
```

People with browsers that do not support Java would see the text:

Java virtual machine failed to start. Is Java enabled on your browser?

while those with browsers that do support Java would see only the applet.

Parameters Specified via <PARAM> Tags

The HTML <PARAM> tag is more like an attribute of the <APPLET> tag than a separate HTML tag. Although it is a tag, it is valid only inside an <APPLET> tag (or an <OBJECT> tag). It functions to pass parameters to the applet. This section discusses parameters common to all ION Applets. Parameters specific to individual applets included in the ION package are discussed in the applet-specific sections below.

The syntax of the <PARAM> tag is as follows:

<PARAM NAME="*name*" VALUE="*value*">

The NAME attribute can be set to one of the following parameters:

Connecting to the ION Server

Before IDL commands can be executed and graphics created, the ION applet must connect to the ION Server. Establish a connection by including the following connection parameters in the HTML code that creates the applet.

SERVER_NAME

Set this value of this parameter equal to the name of the computer on which the ION Server is running. The server name can be either a simple host name (i.e. `myhost`) or a fully-qualified domain name (i.e. `myhost.mycompany.com`). Java security mechanisms require that the applet be located on the same machine as the ION Server. If the server name is not provided, the host name of the machine from which the applet was loaded is used.

PORT_NUMBER

The port number of the port on the server where the ION Daemon is listening. By default, the ION Server listens to port 7085.

SERVER_DISCONNECT

Set the value of this parameter equal to "YES" if you want the applet to disconnect from the server when all commands have been processed. (Note that if more than one applet is using the connection, the connection will not be closed until all commands from all of the connected applets have been completed.) The default value is "NO".

CONNECTION_TYPE

Set the value of this parameter to specify what type of connection ION should use. The three possible values are:

- HTTP_CON — Make only HTTP connections, using the ION HTTP Tunnel Broker.
- SOCK_CON — Make only socket connections, using only the ION Daemon.
- BEST_CON — Attempt to make a socket connection. If a socket connection is not possible, attempt to make an HTTP connection. This is the default setting.

See "Configuring The ION HTTP Tunnel Broker" on page 31 for additional details about HTTP connections.

CONNECTION_TIMEOUT

Set the value of this parameter to an integer number of seconds to wait before assuming that a socket connection has failed. If the CONNECTION_TYPE parameter is set to BEST_CON, ION will attempt to make an HTTP connection if the timeout time expires before a socket connection is made.

HTTP_HOSTNAME

Set the value of this parameter equal to the hostname of the computer on which the ION HTTP Tunnel Broker is running.

HTTP_PORT

Set the value of this parameter equal to the port number the ION HTTP Tunnel Broker is listening to.

Example

The following connects the "MyApplet" applet to a server named "Server1", using the default port number, the default connection type, and specifies that the applet should not disconnect from the server when all commands have been processed:

```
<APPLET NAME="MyApplet" WIDTH=100 HEIGHT=100
    CODE=com.rsi.ion.IONGraphicApplet.class>
    ARCHIVE="ion_16.zip"
    CODEBASE=../classes>
    <PARAM NAME="SERVER_NAME" VALUE="Server1">
    <PARAM NAME="SERVER_DISCONNECT" VALUE="NO">
<!-- Other applet code -->
</APPLET>
```

Using the Same Connection for Multiple Applets

Multiple ION applets can share a single connection to the ION Server. Since each open connection consumes network bandwidth, it is often efficient to let several applets share the same connection.

To specify an existing connection for a new applet, use the ION_CONNECTION_NAME parameter rather than the SERVER_NAME, PORT_NUMBER, and SERVER_DISCONNECT parameters.

Note

All applets using the same connection must be loaded into the browser at the same time. In general, this means that applets that share a connection should be included in the same HTML page.

ION_CONNECTION_NAME

Set the value of this parameter equal to the name of the applet whose connection you wish to share. The applet's name is specified by the NAME attribute in the APPLET tag.

Example

The following creates a second applet named "AnotherApplet" and specifies that it share the server connection created for "MyApplet":

```
<APPLET NAME="AnotherApplet" WIDTH=100 HEIGHT=100
    CODE=com.rsi.ion.IONGraphicApplet.class
    ARCHIVE="ion_16.zip"
    CODEBASE=../classes>
    <PARAM NAME="ION_CONNECTION_NAME" VALUE="MyApplet">
<!-- Other applet code -->
</APPLET>
```

Behavior Parameters

Two behavior parameters determine how an applet responds to certain user actions. The two behaviors currently supported by all ION applets allow the applets to display debug information and link to other HTML pages. Use the following parameters to alter the behavior of pre-built applets:

DEBUG_MODE

If the value of this parameter is set to "YES", holding down the shift key and clicking the mouse in the applet drawing area displays a window containing the IDL commands and server responses associated with the applet's connection. If more than

one applet is connected to the connection, the information for all applets is displayed. If the main connection has DEBUG_MODE set to "NO" (or not specified), but an applet connected to it has DEBUG_MODE turned on, debug will be turned on for the entire connection. The default value is "NO."

LINK_URL

Set the value of this parameter to a URL that will be loaded if the user clicks in the applet area. The switch to the linked URL happens before any mouse events are passed to the server. This option should not be used with ION applets running IDL routines that accept mouse input.

Example

The following specifies that the "MyApplet" applet will display debug information and will link to the RSI web page if the user clicks in the applet drawing area:

```
<APPLET NAME="MyApplet" WIDTH=100 HEIGHT=100
    CODE=com.rsi.ion.IONGraphicApplet.class
    ARCHIVE="ion_16.zip"
    CODEBASE=../classes>
    <PARAM NAME="SERVER_NAME" VALUE="Server1">
    <PARAM NAME="SERVER_DISCONNECT" VALUE="NO">
    <PARAM NAME="DEBUG_MODE" VALUE="YES">
    <PARAM NAME="LINK_URL" VALUE="http://www.researchsystems.com">
<!-- Other applet code -->
</APPLET>
```

IONGraphicApplet

The IONGraphicApplet is used to execute a series of IDL commands and display the results. Any valid IDL commands that are not explicitly excluded by the ION security mechanism (see "Command Security" on page 34) can be passed to the IONGraphicApplet for execution. Using the ION Applet parameters, the Applet can also display debug information and be used as a hyperlink to another HTML page.

The IDL commands can be sent synchronously or asynchronously. By default, each command is sent and the client *blocks* (stops accepting commands) until the command is complete. However, in some circumstances the client needs to regain control of the application immediately to be able to process user input. An example of this situation would be when a command starts an IDL routine that requires a large amount of processing. If the command is blocking, the client will not be free to receive user input or possibly even redraw itself.

Parameters

In addition to the parameters described in "Parameters Specified via <PARAM> Tags" on page 72, the IONGraphicsApplet accepts the following parameters:

IDL_COMMAND_0, ..., IDL_COMMAND_*n*

The IDL_COMMAND_* parameters specify the IDL commands to send to the ION Server. The value of each IDL_COMMAND is a valid, single line IDL command (the "$" line continuation is not supported by ION). Note that commands that are explicitly excluded via the ION security mechanism are not processed.

Note

Command numbers must be continuous, beginning with zero and ending with *n*.

AYSNC_COMMANDS

Set the value of this parameter to "YES" if the client should send commands asynchronously. All commands are sent in order, and control is returned to the applet as soon as the commands are sent. The default value is "NO".

DECOMPOSED_COLOR

If set to "YES", the applet will treat pixel values as RGB triplets when on a true-color (24-bit or 32-bit) device. (This is the default.) If set to "NO", the applet will treat the first eight bits (the red portion) of the pixel value as an index into the current color table when displaying on a true color device. For more information on decomposed

color mode, see the documentation for the DECOMPOSED keyword to the DEVICE procedure in the *IDL Reference Guide*.

Example

The following example creates an IONGraphicsApplet that connects to a server, generates some data, sets the color table, and displays the data using IDL's SHOW3 procedure. In the example, debugging mode is enabled, and the applet drawing area is a link to the RSI Web page.

```
<APPLET NAME="CONNECTION" CODE=IONGraphicApplet.class
        WIDTH=200 HEIGHT=200
        ARCHIVE="ion_16.zip"
        CODEBASE=../classes>
    <!-- This applet connects to host KIROC, port 8084 -->
    <PARAM NAME="SERVER_NAME" VALUE="KIROC">
    <PARAM NAME="PORT_NUMBER" VALUE="8084">
    <PARAM NAME="LINK_URL" VALUE="http://www.researchsystems.com">
    <PARAM NAME="DEBUG_MODE" VALUE="YES">
    <PARAM NAME="SERVER_DISCONNECT" VALUE="YES">
    <PARAM NAME="IDL_COMMAND_0"
        VALUE="a = exp(-(shift(dist(30), 15, 15)/7)^2)">
    <PARAM NAME="IDL_COMMAND_1" VALUE="loadct, 1">
    <PARAM NAME="IDL_COMMAND_2" VALUE="show3, a">
</APPLET>
```

IONContourApplet

The IONContourApplet displays an IDL contour plot. The X, Y and Z values of the plot and any IDL Contour properties supported by ION can be set through parameters to the applet.

Note

You can also create contour plots using the IONGraphicApplet, specifying the contour properties in IDL command strings. The IONContourApplet is merely a simplified way to display contour plots.

Parameters

In addition to the parameters described in "Parameters Specified via <PARAM> Tags" on page 72, the IONContourApplet accepts the following parameters:

X_VALUES

Set the value of this parameter equal to a valid IDL expression that evaluates to a vector or two-dimensional array specifying the X coordinates for the contour surface. If X_VALUES specifies a vector, each element specifies the X coordinate for a column in the Z_VALUES array(e.g., X[0] specifies the X coordinate for Z[0,*]). If X_VALUES specifies a two-dimensional array, each element specifies the X coordinate of the corresponding point in the Z_VALUES array.

Y_VALUES

Set the value of this parameter equal to a valid IDL expression that evaluates to a vector or two-dimensional array specifying the Y coordinates for the contour surface. If Y_VALUES specifies a vector, each element specifies the Y coordinate for a column in the Z_VALUES array(e.g., Y[0] specifies the Y coordinate for Z[0,*]). If Y_VALUES specifies a two-dimensional array, each element specifies the Y coordinate of the corresponding point in the Z_VALUES array.

Z_VALUES

Set the value of this parameter equal to a valid IDL expression that evaluates to a one- or two-dimensional array containing the values that make up the contour surface. If the X_VALUES and Y_VALUES parameters are provided, the contour is plotted as a function of the (X, Y) locations specified by their contents. Otherwise, the contour is generated as a function of the two-dimensional array index of each element of Z_VALUES.

contour_property_*1*, ..., contour_property_*n*

Here, `contour_property_*` is the name of a contour property supported by the IONGrContour class. Properties for the IONContourApplet reflect the capabilities implemented in keywords to the IDL CONTOUR procedure.

Note

Unlike the IONGraphicApplet IDL_COMMAND_* parameter, the `contour_property` parameters are not numbered.

The following IDL Contour properties are supported by IONContourApplet. Refer to the IDL documentation on keywords available for use with the CONTOUR procedure for an explanation of each property:

C_ANNOTATION, C_CHARSIZE, C_COLORS, C_LABELS, C_LINESTYLE, C_ORIENTATION, C_SPACING, CLOSED, DOWNHILL, FILL, CELL_FILL, FOLLOW, IRREGULAR, LEVELS, NLEVELS, OVERPLOT, BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, SUBTITLE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, XCHARSIZE, YCHARSIZE, ZCHARSIZE, XGRIDSTYLE, YGRIDSTYLE, ZGRIDSTYLE, XMARGIN, YMARGIN, ZMARGIN, XMINOR, YMINOR, ZMINOR, XRANGE, YRANGE, ZRANGE, XSTYLE, YSTYLE, ZSTYLE, XTICKFORMAT, YTICKFORMAT, ZTICKFORMAT, XTICKLEN, YTICKLEN, ZTICKLEN, XTICKNAME, YTICKNAME, ZTICKNAME, XTICKS, YTICKS, ZTICKS, XTICKV, YTICKV, ZTICKV, XTITLE, YTITLE, ZTITLE, ZVALUE, ZAXIS

Example

The following example creates an IONContourApplet that connects to the same server used by the "Connection" applet defined in the IONGraphicApplet example. The applet generates some data for the Z value of the contour, and sets the "Title" property of the contour plot.

```
<APPLET NAME="CONTOUR" CODE=IONContourApplet.class
    WIDTH=200 HEIGHT=200
    ARCHIVE="ion_16.zip"
    CODEBASE=../classes>
    <!-- This applet uses the applet 'CONNECTION' to connect
        to the server -->
    <PARAM NAME="ION_CONNECTION_NAME" VALUE="CONNECTION">
    <PARAM NAME="Z_VALUES" VALUE="exp(-(shift(dist(30), 15,
        15)/7)^2)">
    <PARAM NAME="TITLE" VALUE="Contour">
</APPLET>
```

Note that the example uses an IDL expression to generate the Z values for the contour. The Z values could also have been specified as an IDL array, with a statement like:

```
<PARAM NAME="Z_VALUES"
    VALUE="[[1,2,3,4][2,3,4,5][3,4,5,6][4,5,6,7]]">
```

IONMapApplet

The IONMapApplet is an applet that displays 2D data on a map projection. The data can be displayed as an image or a contour plot and can contain latitude/longitude grid lines, and landmass and political boundaries. The applet is capable of projecting multiple contour plots, one image, latitude/longitude grid lines, and boundaries onto the drawing area. In the case of multiple datasets, the drawing order is as follows:

- Images are always drawn first
- Any ordering of the following:
  - Grid Lines (drawn once)
  - Boundaries (continents, drawn once)
  - Contours (in numerical order)

The IONMapApplet is based on the ION[Gr]Map* and IONGrContour objects.

Note

You can also create plots using the IONGraphicApplet, specifying the map properties in IDL command strings. The IONMapApplet is merely a simplified way to display maps.

Parameters

In addition to the standard IONApplet parameters, the IONMapApplet accepts the following parameters:

IDL_COMMAND_*n*

A set of IDL commands starting with *n*=0 that are executed before the map commands

MAP_GRID

Display latitude/longitude lines on the map

MAP_CONT

Display continents on the map

MAP_[LAT,LON]

Center of the map

MAP_ROTATION

Rotation of the map

MAP_*

Keywords accepted by IONGrMap

MAP_GRID_*

Keywords accepted by IONGrMapGrids (valid if MAP_GRID is set)

MAP_CONT_*

Keywords accepted by IONGrContinents (valid if MAP_CONT is set)

MAP_IMAGE_DATA

An IDL statement that evaluates to a 2D dataset that is used as the image data

MAP_IMAGE_*

Keywords accepted by IONGrMapImage (valid if MAP_IMAGE_DATA is set)

MAP_CONTOUR*n*_*

Keywords accepted by the IONContourApplet. *n* identifies the contour to which the keyword is applied. The applet starts processing at MAP_CONTOUR1 and continues sequentially until no more contours are encountered.

MAP_DISP_ORDER

Specifies the order that the data sets are displayed on the map. Valid orders are as follows (CTR = contour, CT = continents, GR = grid lines):

- CTR_CT_GR, GTR_GR_CT
- CT_CTR_GR, CT_GR_CTR
- GR_CTR_CT, GR_CT_CTR

The following IDL MAP_SET properties are supported by IONMapApplet. Refer to the IDL documentation on keywords available for use with the MAP_SET procedure for an explanation of each property:

Projection Types: AITOFF, ALBERS, AZIMUTHAL, CONIC, CYLINDRICAL, GNOMIC, GOODESHOMOLOSINE, HAMMER, LAMBERT, MERCATOR, MILLER, MOLLEWIDE, ORTHOGRAPHIC, ROBINSON, SATELLITE, SINUSOIDAL, STEREOGRAPHIC, TRANSVERSE_MERCATOR

Map Characteristics: ADVANCE, CHARSIZE, CLIP, COLOR, CONTINENTS, CON_COLOR, HIRES, E_CONTINENTS, E_GRID, E_HORIZON, GLINESTYLE, GLINETHICK, GRID, HORIZON, LABEL, LATALIGN, LATDEL, LATLAB, LONDEL, LONLAB, MLINESTYLE, MLINETHICK, NOBORDER, NOERASE, TITLE, USA, XMARGIN, YMARGIN

Projection Parameters: CENTRAL_AZIMUTH, ELLIPSOID, ISOTROPIC, LIMIT, SAT_P, SCALE, STANDARD_PARALLELS

Graphics: POSITION, T3D, ZVALUE

IONPlotApplet

The IONPlotApplet displays an IDL plot. The X and Y values of the plot and any IDL plot properties supported by ION can be set through parameters to the applet.

Note

You can also create plots using the IONGraphicApplet, specifying the plot properties in IDL command strings. The IONPlotApplet is merely a simplified way to display plots.

Parameters

In addition to the parameters described in "Parameters Specified via <PARAM> Tags" on page 72, the IONPlotApplet accepts the following parameters:

X_VALUES

Set the value of this parameter equal to a valid IDL expression that evaluates to a vector of X data. If `X_VALUES` is not specified, the data in `Y_VALUES` is plotted as a function of point number (starting at zero). If both arguments are provided, `Y_VALUES` is plotted as a function of `X_VALUES`.

Y_VALUES

Set the value of this parameter equal to a valid IDL expression that evaluates to a vector of Y data.

plot_property_*1*, ..., plot_property_*n*

Here, `plot_property_*` is the name of a plot property supported by the IONGrPlot class. Properties for the IONPlotApplet reflect the capabilities implemented in keywords to the IDL PLOT procedure.

Note

Unlike the IONGraphicApplet IDL_COMMAND_* parameter, the plot_property parameters are not numbered.

The following IDL Plot properties are supported by IONPlotApplet. Refer to the IDL documentation on keywords available for use with the PLOT procedure for an explanation of each property:

BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, PSYM,

SUBTITLE, SYMSIZE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, ZLOG

Example

The following example creates an IONPlotApplet that connects to the same server used by the "Connection" applet defined in the IONGraphicApplet example. The applet generates some data for the X value of the plot, and sets the "Title" and "Linestyle" properties of the plot.

```
<APPLET NAME="PLOT" CODE=IONPlotApplet.class
        WIDTH=200 HEIGHT=200
        ARCHIVE="ion_16.zip"
        CODEBASE=../classes>
    <!-- This applet uses the applet 'CONNECTION' to connect
        to the server -->
    <PARAM NAME="ION_CONNECTION_NAME" VALUE="CONNECTION">
    <PARAM NAME="LINK_URL" VALUE="plotappletsrc.html">
    <PARAM NAME="X_VALUES" VALUE="exp(-(shift(dist(30), 15,
        15)/7)^2)">
    <PARAM NAME="TITLE" VALUE="Plot">
    <PARAM NAME="LINESTYLE" VALUE="2">
</APPLET>
```

IONSurfaceApplet

The IONSurfaceApplet displays an IDL Surface plot. The X, Y and Z values of the plot and any IDL Surface properties supported by ION can be set through parameters to the applet.

Note

You can also create surface plots using the IONGraphicApplet, specifying the plot properties in IDL command strings. The IONSurfaceApplet is merely a simplified way to display surface plots.

Parameters

In addition to the parameters described in "Parameters Specified via <PARAM> Tags" on page 72, the IONSurfaceApplet accepts the following parameters:

X_VALUES

Set the value of this parameter equal to a valid IDL expression that evaluates to a vector or two-dimensional array specifying the X coordinates for the surface. If X_VALUES specifies a vector, each element specifies the X coordinate for a column in the Z_VALUES array(e.g., X[0] specifies the X coordinate for Z[0,*]). If X_VALUES specifies a two-dimensional array, each element specifies the X coordinate of the corresponding point in the Z_VALUES array.

Y_VALUES

Set the value of this parameter equal to a valid IDL expression that evaluates to a vector or two-dimensional array specifying the Y coordinates for the surface. If Y_VALUES specifies a vector, each element specifies the Y coordinate for a column in the Z_VALUES array(e.g., Y[0] specifies the Y coordinate for Z[0,*]). If Y_VALUES specifies a two-dimensional array, each element specifies the Y coordinate of the corresponding point in the Z_VALUES array.

Z_VALUES

Set the value of this parameter equal to a valid IDL expression that evaluates to a one- or two-dimensional array containing the values that make up the surface. If the X_VALUES and Y_VALUES parameters are provided, the contour is plotted as a function of the (X, Y) locations specified by their contents. Otherwise, the surface is generated as a function of the two-dimensional array index of each element of Z_VALUES.

surface_property_*1*, ..., surface_property_*n*

Here, `surface_property_*` is the name of a surface property supported by the IONGrSurface class. Properties for the IONSurfaceApplet reflect the capabilities implemented in keywords to the IDL SURFACE procedure.

Note

Unlike the IONGraphicApplet IDL_COMMAND_* parameter, the surface_property parameters are not numbered.

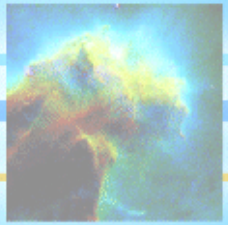
The following IDL Surface properties are supported by the IONSurfaceApplet. Refer to the IDL documentation on keywords available for use with the SURFACE procedure for an explanation of each property:

AX, AZ, BOTTOM, HORIZONTAL, LEGO, LOWER_ONLY, SAVE, SHADES, UPPER_ONLY, ZAXIS, BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, LINESTYLE, NOCLIP, NODATA, NOERASE, NORMAL, POSITION, SUBTITLE, T3D, TICKLEN, TITLE, MAX_VALUE, MIN_VALUE, NSUM, POLAR, XLOG, YNOZERO, YLOG, XCHARSIZE, YCHARSIZE, ZCHARSIZE, XGRIDSTYLE, YGRIDSTYLE, ZGRIDSTYLE, XMARGIN, YMARGIN, ZMARGIN, XMINOR, YMINOR, ZMINOR, XRANGE, YRANGE, ZRANGE, XSTYLE, YSTYLE, ZSTYLE, XTICKFORMAT, YTICKFORMAT, ZTICKFORMAT, XTICKLEN, YTICKLEN, ZTICKLEN, XTICKNAME, YTICKNAME, ZTICKNAME, XTICKS, YTICKS, ZTICKS, XTICKV, YTICKV, ZTICKV, XTITLE, YTITLE, ZTITLE, ZVALUE, ZLOG

Example

The following example creates an IONSurfaceApplet that connects to the same server used by the "Connection" applet defined in the IONGraphicApplet example. The applet generates some data for the Z value of the plot, and sets the "Title" and "Lego" properties of the plot.

```
<APPLET NAME="SURFACE" CODE=IONSurfaceApplet.class
        WIDTH=200 HEIGHT=200
        ARCHIVE="ion_16.zip"
        CODEBASE=../classes>
    <!-- This applet uses the applet 'CONNECTION' to connect
        to the server -->
    <PARAM NAME="ION_CONNECTION_NAME" VALUE="CONNECTION">
    <PARAM NAME="LINK_URL" VALUE="surfaceappletsrc.html">
    <PARAM NAME="Z_VALUES" VALUE="exp(-(shift(dist(30), 15,
    15)/7)^2)">
    <PARAM NAME="TITLE" VALUE="Surface">
    <PARAM NAME="LEGO" VALUE="1">
</APPLET>
```



Chapter 5: Building ION Applets and Applications

This chapter discusses the process of building your own ION Java applets and standalone Java applications. Details on the ION Java classes used to build ION applets and applications can be found in Chapter 3, “Overview of the ION Java Classes” and Chapter 6, “ION Java Class and Method Reference”. The following topics are discussed:

- Direct Graphics in ION
- Object Graphics in ION
- Compiling .java Files
- Error Handling and ION Exceptions
- Debug Mode
- Converting Between IDL and Java Bytes
- Considerations Specific to ION Applets

Direct Graphics in ION

The ION Device

IDL uses the concept of a *current graphics device* when creating and displaying IDL Direct Graphics. When the ION Server requests graphics from IDL, it automatically sets the current graphics device to `'ion'`; graphics output from IDL is sent directly to the ION Server. You do not need to explicitly set the graphics device to `'ion'` unless you have explicitly used the IDL SET_PLOT procedure to change the current device to some other device.

For example, suppose you wish to include a “Print” button in a Java application. Your applet might include something like the following:

```
ion.executeIDLCommand("SET_PLOT, 'printer'")
execute more IDL commands to draw an image on the printer
ion.executeIDLCommand("SET_PLOT, 'ion'")
```

Note

The IDL TVRD function is not supported by the ION Device.

Keywords Accepted by the ION Device

The following keywords to the IDL DEVICE routine are available when the current graphics device is set to `'ion'`. Except where indicated, keywords to the ION device work just as they do for other IDL graphics devices.

COPY

Use this keyword to copy a rectangular area of pixels from one region of a window to another. COPY should be set to a six or seven element array: [X_s, Y_s, N_x, N_y, X_d, Y_d, W], where: (X_s, Y_s) is the lower left corner of the source rectangle, (N_x, N_y) are the number of columns and rows in the rectangle, and (X_d, Y_d) is the coordinate of the destination rectangle. Optionally, W is the index of the window *from which the pixels should be copied* to the *current* window. If it is not supplied, the current window is used as both the source and destination.

DECOMPOSED

This keyword is used to control the way in which graphics color index values are interpreted when using displays with decomposed color (TrueColor visuals). This keyword has no effect with other types of visuals.

Set this keyword to 1 to cause color indices to be interpreted as 3, 8-bit color indices where the least-significant 8 bits contain the red value, the next 8 bits contain the green value, and the most-significant 8 bits contain the blue value. This is the way IDL has always interpreted pixels when using visual classes with decomposed color.

Set this keyword to 0 to cause the least-significant 8 bits of the color index value to be interpreted as a PseudoColor index. This setting allows users with TrueColor displays to use IDL programs written for standard, PseudoColor displays without modification.

In older versions of IDL, color index values higher than !D.N_COLORS-1 were clipped to !D.N_COLORS-1 in the higher level graphics routines. In some cases, this clipping caused the exclusive-OR graphics mode to malfunction with raster displays. This clipping has been removed. Programs that incorrectly specified color indices higher than !D.N_COLORS-1 will now probably exhibit different behavior.

FONT

Set this keyword to a scalar string specifying the name of the font used when the hardware font is selected.

Note

The hardware fonts available are supplied by Java itself, not the platform on which IDL is running. Java's font system supplies several standard fonts. These font names will map to different actual fonts on different platforms, but will always be handled gracefully by Java. If you specify a different font, Java will substitute one of the standard fonts automatically.

Tip

Avoid using device fonts for performance reasons. See "Tips for Increasing Execution Speed in ION Java" on page 45.

Note that hardware fonts cannot be rotated, scaled, or projected, and that the "!" commands accepted for vector fonts for subscripts and superscripts may not work. When generating three-dimensional plots, it is best to use the vector-drawn characters because IDL can draw them in perspective with the rest of the plot.

The GET_FONTNAMES keyword, described below, can be used to retrieve a list of available fonts.

The `FONT` keyword should be set to a string with the following form:

```
DEVICE, FONT="font*modifier1*modifier2*...modifiern"
```

where the asterisk (*) acts as a delimiter between the font's name (*font*) and any modifiers. The string is *not* case sensitive. Modifiers are simply "keywords" that change aspects of the selected font. Valid modifiers are:

- For font size: Any number is interpreted as the point size of the font to use.
- For font weight: PLAIN, BOLD
- For font angle: ITALIC

For example, the following commands tell ION to use TrueType fonts, change the font, and then make a simple plot:

```
ion.executeIDLCommand("!P.FONT = 1")
ion.executeIDLCommand("DEVICE, FONT = 'Helvetica Bold Italic,
/TT_FONT'")
ion.executeIDLCommand("PLOT, FINDGEN(10), TITLE = 'IDL Plot'")
```

GET_CURRENT_FONT

Set this keyword to a named variable in which the name of the current font is returned as a scalar string.

GET_FONTNAMES

Set this keyword to a named variable in which a string array containing the names of available fonts is returned. If no fonts are found, a null scalar string is returned. This keyword must be used in conjunction with the FONT keyword. Set the FONT keyword to a scalar string containing the name of the desired font or a wildcard.

GET_GRAPHICS_FUNCTION

Set this keyword to a named variable that returns the value of the current graphics function (which is set with the SET_GRAPHICS_FUNCTION keyword). This can be used to remember the current graphics function, change it temporarily, and then restore it. See the SET_GRAPHICS_FUNCTION keyword for an example.

GET_SCREEN_SIZE

Set this keyword to a named variable in which to return a two-word array that contains the width and height of the server's screen, in pixels.

SET_CHARACTER_SIZE

The standard size and vertical spacing of vector-drawn fonts can be changed by specifying this keyword with a two-element vector. The first element specifies the new character width and thus the height of the characters (because vector-drawn fonts have a fixed aspect ratio). The second element specifies the vertical distance between

lines. The default produces a character that is approximately 8 pixels wide, with 12 pixels between lines.

SET_GRAPHICS_FUNCTION

Most window systems allow applications to specify the graphics function. This is a logical function which specifies how the source pixel values generated by a graphics operation are combined with the pixel values already present on the screen. ION supports only the following two of the fifteen graphics functions supported by IDL Direct Graphics:

Logical Function	Code	Definition
GXcopy	3	source
GXxor	6	source XOR destination

The default graphics function is GXcopy, which causes new pixels to completely overwrite any previous pixels. Not all functions are available on all window systems.

See “IDL Graphics Devices” in the *IDL Reference Guide* for more information about how IDL handles graphics devices.

Object Graphics in ION

To render IDL Object Graphics in ION Java, you use the following general technique:

1. Create the IDL objects
2. Create an off-screen buffer (an IDLgrBuffer object)
3. Draw the object to the buffer, then read the contents of the buffer as an image
4. TV the image to the ION device

The following example demonstrates this technique.

First, create your IDL graphic objects, contained in the proper object hierarchy (model and view):

```
oView=obj_new('IDLgrView', COLOR=[255,255,255])
oModel=obj_new('IDLgrModel')
oText=obj_new('IDLgrText', 'Hello World', COLOR = [255,0,0])
oModel->Add, oText
oView->Add, oModel
```

Then, create an off-screen buffer object to which to draw in IDL. Match the dimensions of the ION drawable. For example, suppose your .java file contains the following method:

```
public void buildGUI()
    {
       c_ionDrw = new IONGrDrawable(400,400);

       setLayout(new FlowLayout());
       add(c_ionDrw);
    }
```

In the above code, the IONGrDrawable is defined with dimensions of (400, 400). Therefore, you would create the IDLgrBuffer object in IDL as follows:

```
oBuffer=obj_new('IDLgrBuffer',DIMENSIONS=[400,400])
```

Next, draw the object to your buffer object:

```
oBuffer->Draw, oView
```

Then get the image object from the buffer:

```
oImage=oBuffer->Read()
```

Now extract the data:

```
oImage->Getproperty, DATA=image
```

Make sure to destroy the image object since it is no longer needed:

```
OBJ_DESTROY, oImage
```

Next, convert the TrueColor image to 8-bit to reduce the bandwidth required to send it to the client:

```
result=COLOR_QUAN(image,1,r,g,b)
```

Load the color table:

```
TVLCT,r,g,b
```

Lastly, TV the image:

```
TV, result
```

The image is then displayed in the current ION Java drawable.

Note

For a similar but slightly more complex version of this example, click the Object Graphics link on the page `advanced.html`. The Java source code resides in `objgraphics.java` in the `IDL60\products\ION20\ion_java\examples\src` directory. The IDL `create_surface.pro` in the `examples` directory contains a "draw_buffer" procedure that illustrates the coding routine shown above.

Using Object References

When we initially create an object, we get back a reference to that object. Since the IDL session is persistent in ION Java, we can use object references later in event callbacks for the applet. It is not necessary to create Java variables for persistence because the object continues to exist in the persistent IDL session. For instance, you could add a button to our Hello World applet to rotate the text. In the event callback for the button, you would call the rotate method on the model object whose reference you obtained initially. Then you would use the buffer technique to redraw the view.

Compiling .java Files

Keep the following points in mind when you compile the `.java` file that contains your applet or application code.

Setting the Class Path

When you compile an ION applet or application, the ION class files must be in the Java compiler's class path. Since ION is a *package*, the class files are stored in a directory structure. The ION class files are located in the following ION installation subdirectory of the `classes` directory:

```
Root_ION/com/rsi/ion
```

Where *`Root_ION`* indicates the path to the `classes` directory. If you have installed ION in the default directory, *`Root_ION`* would be:

```
On UNIX:
/usr/local/rsi/ion_1.6/ion_java/classes

On Windows:
C:\rsi\IDL60\products\ION20\ion_java\classes
```

Depending on your specific Java compiler, this can be accomplished by defining the system CLASSPATH environment variable. See "Setting the Class Path Environment Variable" on page 97 for more information. However, the recommended method is to specify the class path on the command line when you are compiling your program. See the following section for more information.

Setting the Class Path When Compiling

Because browsers react differently to a class path system definition, another way to specify the class path is to specify the ION path on the command line when you compile your Java program. If you have installed ION in the default directory, this might be similar to one of the following:

```
On UNIX:
javac -classpath ".;/usr/local/rsi/ion_1.6/ion_java/classes"
    myIonApp.java

On Windows:
javac -classpath "C:\rsi\IDL60\products\ION20\ion_java\classes"
    myIonApp.java
```

This method of specifying the class path has the added benefit of simulating the same environment that your clients will experience when running your application from your browser. This method does not rely on having a system environment variable

pointing to the ION `classes` directory, something your clients are unlikely to have defined.

Setting the Class Path Environment Variable

To define the CLASSPATH environment variable, you would set it using the following shell command on UNIX:

```
setenv CLASSPATH ".;/usr/local/rsi/ion_1.6/ion_java/classes"
```

or modify the class path environment variable in the System Environment dialog on Windows. The Java compiler will add the `com/rsi/ion` portion of the path when it looks for the package.

Once the CLASSPATH is set, you can compile your code with a shell command like the following:

```
javac myIONApp.java
```

where `myIONApp` is the name of your applet or application.

Warning

If you are running the client and the server on the same machine, setting the system CLASSPATH environment variable can result in errors similar to the following, appearing in your browser's Java console:

Netscape Java Console — #Applet exception:
error.java.lang.ClassFormatError:class already loaded

IE Java Console — Error getting package information: com/rsi/ion

To avoid such errors, specify the class path when compiling as described in the previous section.

Error Handling and ION Exceptions

When the ION Server detects an error, it throws an exception value you can detect and act upon using error-handling code. Consult the reference page for the method you are using to determine which exceptions ION can detect in a given situation.

Error handling is generally accomplished via a Java `try/catch` code segment. The following skeleton `try/catch` code illustrates how to catch exceptions and display an error message on the Java console. For a more detailed example, see "Simple Applet Example" on page 105.

Note

If an ION method (or any Java method, for that matter) throws a checked exception value, you must handle the exception in your code. The Java compiler will complain if you do not properly handle all possible exception values. Refer to a Java manual for more information on Java exception handling.

```
try{
   some ION command
}catch(IOException e) {
   // IO Error
      System.err.println("Error: Communication error:
      "te.getMessage());
      return;
}catch( IONIllegalCommandException e){
   // Illegal Command
      System.err.println("Error: Illegal Command:
      "te.getMessage());
      return;
}catch( IONSecurityException e){
   // Security Violation
      System.err.println("Error: Security Violation:
      "te.getMessage());
      return;
   }
}
```

Debug Mode

The "IONGrConnection / IONJGrConnection Class" on page 173 and the "IONGrDrawable / IONJGrDrawable Class" on page 188 both supply a debugMode() method that allows you to view the IDL command log output. Enable debug mode by adding the following to the Java code which establishes the connection to the ION Server:

```
connection.debugMode(true);
```

where *connection* is the IONGrConnection object or IONGrDrawable object.

When debug mode is in effect, holding down the Shift key and clicking on the ION drawing area associated with the connection, a separate window opens and displays the output that would typically appear in the IDL command log.

Tip

For those classes that do not have a debugMode() method, you can use the IONOutputListener Interface to return IDL output.

Debugging Your Application

When developing your applications, you can use the Java method, `System.out.println()` to print any type of Java program information. As the previous method can be used to view IDL command log output, this methods returns comparable Java program information. When running a standalone application (not an applet in a web browser), these log messages will be printed to your command line console. When running an applet from a browser, the information will be printed in your Java console. See "Check the Java Console Log" on page 291 for instructions on how to open the Java console.

Converting Between IDL and Java Bytes

It is important to understand the differences between byte data types in IDL and Java in order to ensure that byte arrays are transferred correctly between IDL and Java.

IDL and Java both have a basic byte data type, however, IDL's byte is unsigned and Java's is signed. Java does not support the concept of unsigned types. When a byte in Java is cast to an integer, the sign is preserved via sign extension. This can cause problems when transferring byte data between IDL and Java.

To understand the problem, consider the following unsigned IDL byte:

```
BYTE idlByte = 170
```

The binary representation of this byte is:

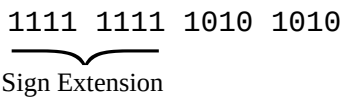
```
1010 1010
```

Unsigned 8-bit bytes give a numeric range of 0 to 255 while signed bytes have a range of –128 to 127. Signed numbers are represented using "two's complement." In two's complement, the highest bit is the sign-bit and determines whether the number is negative or positive. This is not a simple negation, however, and does change the unsigned value of the number.

Using Java's signed, two's complement numbers, this same set of bits corresponds to:

```
byte javaByte = -86;
```

When the value is cast to an integer, the sign-bit is extended to fill in the new bits so that the value of the number is preserved. If the integer is 16 bits long, the sign-extended byte becomes (in binary):



The signed value of this is still -86. Due to how two's complement numbers function, however, simply negating this number will not return the original unsigned value. If this was an unsigned data type, its 16-bit value would be 65450.

When a byte value is transferred from IDL to Java and the unsigned value is needed, the number must be converted to a positive number with the lower 8 bits staying the same. To accomplish this, use a bitmask to turn the high-order bits to 0 and preserve the low order bits:

```
byte javaByte = -86;
short javaShort = (short)javaByte; // cast the byte to a short
short unsignedValue = javaShort & 0x00FF;
```

After this conversion, `unsignedValue` is 170, the original value from IDL.

So, why does ION not do this under the hood? The simple answer is that we have taken the approach that Java rules apply on the Java side and IDL rules apply on the IDL side. Understanding the differences will help in application development and allow the developer to have more fine-grained control over how the application works.

Considerations Specific to ION Applets

When creating your ION applet, keep the following points in mind.

Tip

It's a good idea to shut down and restart the browser any time you make a changes to your HTML file or your class files.

Import the ION Package

In addition to the standard Java packages (and any other packages used in your applet), you must import the ION package with the statement:

```
import com.rsi.ion.*
```

ION Applets Extend the Java Applet Class

ION applets are subclassed from (they *extend*) the Java Applet class. When defining your applet class, use a statement similar to the following:

```
public class MyIONApplet extends Applet
```

where *MyIONApplet* is the name of your applet class.

See "Simple Applet Example" on page 105 for an example. For a basic overview of Applets, consult a Java reference.

Including Applets in HTML Pages

To include your compiled applet in an HTML page, use the <APPLET> tag with the NAME, CODE, WIDTH, and HEIGHT attributes:

```
<APPLET NAME="myIONApplet" CODE=myIONApplet.class
             WIDTH=300 HEIGHT=300 >
</APPLET>
```

For more information, see Chapter 4, "Using ION's Pre-Built Applets".

Locating the Class Files for use by ION Applets

ION applets must have access to the ION class files in order to run. While you can use the CODEBASE attribute to specify a relative path from the location of an HTML page containing an ION applet tag to the location of the class files, it is often easier to copy the class files (or provide a symbolic link, if your system supports symbolic links) to another directory located in or near the directory containing your HTML files.

For example, suppose you have located your HTML pages in a directory named `public_html`. You may wish to place the ION package, the ION ZIP file, and the ION JAR file in a subdirectory of `public_html` named `java`. If you then include any ION applet class files you create in the `java` directory, you could simply specify:

```
CODEBASE="./java"
```

in the `<APPLET>` tag used in your HTML page.

See "CODEBASE" on page 69 for further details.

Supporting Java Archive Files

When a web browser encounters an HTML page that contains a Java applet, the class files that make up the applet are downloaded from the web server into the browser. The applet is executed only after all of the necessary class files have been downloaded. Because a separate HTTP connection between the client and the server is established for each class file, the download time for a large applet (an applet with many class files) can be substantial.

To increase the download performance of Java applets, consider using a *Java Archive* file, or JAR file, detailed in number 3 of the following section. A JAR file can contain multiple class files, thus avoiding the need for multiple connections. A JAR file can also be compressed, further speeding the download process. Most modern browsers support the JAR format.

Browser Support of ION Class Library Versions

To support the different methods used by different browsers to download Java class files, ION provides three separate versions of the ION class library. These are:

1. The raw Java class files are contained in the `com/rsi/ion` directory structure within the `classes` directory of the ION distribution. Each file is downloaded to the browser via a separate connection to the server.

   Use raw Java class files with browsers that don't support the ARCHIVE attribute to the APPLET tag. For example, version 3 of Microsoft's Internet Explorer does not support the ARCHIVE attribute.

   Note

   To use this method, you must copy the `com` directory and all its subdirectories to your Web server since the raw Java class files are not copied to your Web server during ION installation.

2. An compressed file named `ion_16.zip` contains all of the Java class files included in the ION package. This ZIP file is located in the `classes` directory of the ION distribution, and can be downloaded via a single connection to the server.

   Use the ZIP file with browsers that support the ARCHIVE attribute and support compressed archive files. For example, version 4 of Netscape's Navigator supports the ARCHIVE attribute and compressed JAR files.

3. An uncompressed Java Archive (JAR) file named `ion_16.jar` contains the Java class files included in the ION package. This JAR file is located in the `classes` directory of the ION distribution, and can be downloaded via a single connection to the server.

   Use the JAR file with browsers that support uncompressed archive files. For example, version 3 and later of Netscape's Navigator supports uncompressed JAR files.

Supporting Multiple Browser Types

Note

This section is relevant only for ensuring support of browsers prior to Netscape 4 or Internet Explorer 5.

Use the following procedure to create a set of HTML pages that will use the most efficient download method for any of the three browser types defined above.

1. Ensure that the archive files are in the same directory. By default, they are located in the `classes` subdirectory of the ION distribution. This directory should be specified via the CODEBASE attribute to the APPLET tag. See "CODEBASE" on page 69 for more information.

2. Create two versions of each HTML page that contains an ION applet. One page should include a reference to the uncompressed archive file via the ARCHIVE attribute to the APPLET tag (`ARCHIVE="ion_16.zip"`). The other page should include a reference to the compressed archive file (`ARCHIVE="ion_16.jar"`). Browsers that do not support the ARCHIVE attribute will ignore it and download the unarchived files.

3. Create a "switch page" that includes JavaScript. The switch page determines which version of the browser is present and loads the appropriate HTML page.

```
<SCRIPT language="JavaScript">
<!--
   navigator.onerror = null;
   version = (  parseInt(navigator.appVersion) > 3 ? "4" : "3");
   if(version == "4"){
   // Version 4 can handle jar files, load the Jar page
      location.replace("JAR_page.html");
   } else {
      location.replace("ZIP_page.html");
}
// -->
</SCRIPT>
```

where *JAR_PAGE*.html is the name of the HTML page that references the ion_16.jar file and *ZIP_PAGE*.html is the name of the HTML page that references the ion_16.zip file. For example, you may name the page that references the JAR file myfile_j.html and the file that references the ZIP file myfile_z.html.

Simple Applet Example

The following Java code creates a simple applet that displays an IDL graphic. The example constructs an applet named Commands; the code is available in the examples/src directory in a file named commands.java.

Note

The characters "//" denote comments in Java code.

```
// -*-C++-*-
//
// commands.java
//
//

/*****************************************************************
 Copyright (c) 1997-2002, Research Systems, Inc.  All rights
 reserved. Unauthorized reproduction prohibited.

 (Of course, because these are examples, feel free to remove these
 lines and modify to suit your needs.)
*****************************************************************
/

import java.awt.*;
import java.applet.*;
import java.io.*;
import java.net.*;
import com.rsi.ion.*;   // Import the ION Package
```

```
public class commands extends Applet implements
IONDisconnectListener
{
// Instance Vars

   IONGrConnection c_ionCon;  // the ion connection
   IONGrDrawable   c_ionDrw;  // the ION drawable
   Dimension       c_dimApp;  // Size of drawing area
   int             c_bConnected=0; // 0 => !conn, 1 => conn, -1 =>
conn failed

// ******************************
// Init Method
// ******************************

  public void init(){
 // Create connection and drawable objects
    c_ionCon = new IONGrConnection();
    c_dimApp = getSize();
    c_ionDrw = new IONGrDrawable(c_dimApp.width, c_dimApp.height);

 // Add the drawable to the AWT tree
    setLayout(new GridLayout(1, 1));
    add(c_ionDrw);
  }
/*
 ************************************************
 * Inorder to display status messages at startup and also
 * to be able to disconnect when the page is not being viewed
 * we override the Applets start() and stop() methods
 ************************************************
 * start()
 *
 * Purpose:
 *   Overide the applet's start method.
 *   Connect to IONJ if not already connected.
 *
 * Note: in pre-ION1.4 releases, this method called repaint.
 * repaint then would call our paint method (now deleted from this
 * file). The paint method was responsible for connecting.
 * However, in some cases our paint method would not be called and
 * the applet would not get its data from the server.
 * We are now guaranteed that we will connect to the IONJ server
 * because start() will always be called when the applet starts.
 */
  public void start(){
    if(c_bConnected == 0)  // Not connected to ION, do so.
      connectToServer();
  }
```

```
/*
 ************************************************
 * stop()
 *
 * Purpose:
 *   Override the applet's stop method. This method
 *   Is called when the page is not being viewed. We
 *   disconnect from the server when this is the case.
 */
  public void stop(){
    if(c_bConnected == 1){
      c_ionCon.removeIONDisconnectListener(this);
      c_ionCon.disconnect();
      c_bConnected=0;
    }
  }

/*
 ************************************************
 * connectToServer()
 *
 * Purpose:
 *  Connects to the ION server, providing feedback to user
 */
  private void connectToServer(){

  // Write Status message
     writeMessage("Connecting to ION Java Server...");

  // Connect to the server
     try {
       c_ionCon.connect(this.getCodeBase().getHost());
     } catch(UnknownHostException eUn) {
         System.err.println("Error: Unknown Host.") ;
        writeMessage("Error:Unknown Host.");
         c_bConnected = -1;
        return;
     } catch(IOException eIO) {
         System.err.println("Error: Establishing Connection. ION
Java Server Down?");
        writeMessage("Error: Establishing Connection. ION Java
Server Down?");
         c_bConnected = -1;
        return;
     } catch(IONLicenseException eLic){
        System.err.println("Error: ION Java License Unavailable.")
;
        writeMessage("Error: ION Java License Unavailable.");
         c_bConnected = -1;
```

```
            return;
         }

        c_bConnected = 1;
        c_ionCon.addIONDisconnectListener(this);
     // Since we are only working with 8 bit, set decompose
        c_ionCon.setDecomposed(false);

     // Add the drawable to the connection
        c_ionCon.IONGr2addDrawable(c_ionDrw);

       writeMessage("Drawing Graphics..."); // message to screen
     // Issue IDL commands to generate a plot
        try {
        // Set the color table
           c_ionCon.executeIDLCommand("loadct, 15");

        // Create some data
           c_ionCon.executeIDLCommand("a = dist(30)");

        // Draw a contour plot
           c_ionCon.executeIDLCommand("show3, a");

       //Note that it is generally faster to package multiple
       //IDL commands into a single .pro to call.  This example
       //sends commands separately so that the code is easier to
    follow.
        } catch(Exception e) {
          String smsg;
          if(e instanceof IOException)
             smsg = "Communication error:"+e.getMessage();
          else if(e instanceof IONSecurityException )
             smsg = "ION Java security error";
          else if(e instanceof IONIllegalCommandException )
             smsg = "Illegal IDL Command detected on server.";
          else
             smsg = "Unknown error: "+e.getMessage();
          System.err.println("Error: "+smsg);
          writeMessage("Error: "+smsg);
          return;
        }
        writeMessage("Done");
     }

    /*
     ***********************************************
     * IONDisconnection()
     *
     * Purpose:
```

```
 *  Called when the connection is broken (can report reason).
 */
 public void IONDisconnection(int i){
  System.err.println("Server Connection Closed");
  writeMessage("Server Connection Closed");
  if(c_bConnected == 1)
     c_bConnected = 0;
 }
/*
 ******************************************
 * writeMessage()
 *
 * Purpose:
 *   Utility method that is used to write a string to the
 *   screen using Java.
 */
  private void writeMessage(String sMsg){
    showStatus(sMsg);
   System.out.println(sMsg);
  }
}
```

Further Examples

Example code illustrating ION features is included in the installed ION distribution. You will find example HTML files located in the `examples` directory in your installed ION distribution. The raw Java source files for the example ION classes are included in the `src` subdirectory of the `examples` directory. Also included in the `examples` directory are a number of IDL `.pro` files that are called by the ION demonstration applets. See “Running the ION Java Examples” on page 48 for more information.

Note

For the examples to function properly, you must have the ION Server running on your server machine. If you do not yet have the ION Server running on your system, visit RSI’s ION Web site and view ION examples there.

ION Applets and Scripting Languages

You can use scripting languages such as JavaScript and VBScript to control ION applets included on an HTML page by calling ION methods that are available to all applets. Communication between scripts and applets gives you a simple way to create interactive HTML pages that build on ION’s pre-built applets.

Browser and Script Language Differences

Two competing scripting languages are currently available for use in HTML pages — JavaScript and VBScript. JavaScript was developed by Netscape for use in its Navigator browser; VBScript was developed by Microsoft for use in its Internet Explorer browser. While the two scripting languages have much in common, they do differ in ways that are beyond the scope of this manual to describe. In the context of writing scripts that communicate with ION applets, the important differences are:

- Netscape browsers have a mechanism called “LiveConnect” that allows communication between JavaScript and applets.
- While Microsoft browsers support JavaScript as well as VBScript, they do not allow communication between JavaScript and applets. In Microsoft browsers, communication between scripts and applets must occur through VBScript.

The practical result of this situation is that in order to create HTML pages that allow users of both Netscape’s Navigator and Microsoft’s Internet Explorer to interact with ION applets via scripts, you must write HTML code that decides “on the fly” which scripting language to use.

Choosing Between JavaScript and VBScript

The simplest way to provide pages that use JavaScript for Netscape browsers and pages that use VBScript for Microsoft browsers is to use a “gateway” HTML page that loads one of two other HTML pages depending on the type of browser. The following HTML page uses JavaScript statements to detect whether the browser accessing the page is Netscape Navigator. If so, it loads a JavaScript version of the HTML page; otherwise it loads a VBScript version of the HTML page.

```
<HTML>
<! This page refers IE or Netscape to the proper ION example >
<SCRIPT language=JavaScript>
// <!--
var browser = navigator.appName;

if (browser.indexOf ("Netscape") != -1)
   location = "javascript.html";  // jump to JavaScript page
else
   location = "vbscript.html";    // jump to VBScript page
// -->
</SCRIPT>
</HTML>
```

Note

The script above assumes that the browser is either Navigator or Internet Explorer. Currently, the vast majority of browsers in use are one of these two; still, you may wish to make your own "gateway" HTML page more robust.

Methods Available

The following methods are available for communication between scripting languages and ION applets:

executeIDLCommand('*string*')

where *string* is a valid IDL command string. The `executeIDLCommand()` method allows you to execute any IDL command via a script, with IDL's output going to the specified applet's drawing area.

For example, if you have an IONSurfaceApplet named MYSURF, you could use the following JavaScript statement to change the color table when the user presses a button:

```
document.MYSURF.executeIDLCommand("LOADCT, 5");
```

See "Example: Using JavaScript" on page 111 for a more complete discussion.

disconnect()

Use this method to disconnect from the ION Server.

Example: Using JavaScript

The following HTML code demonstrates the use of JavaScript to interactively update an ION graphic. The example includes an IONGraphicApplet that displays a shaded surface, uses a JavaScript `select` object to create a pulldown list of rotation values, and adds a button to rotate the surface to the selected angle.

Note

The following script will work in Netscape's Navigator browser, but not in the Microsoft's Internet Explorer browser.

```
<!-- Define the HTML header. Note that the JavaScript is
     included in the HEAD section. -->
<HTML>
<HEAD>
 <TITLE>Simple JavaScript Applet Test</TITLE>
<!-- The script language is JavaScript. We declare the variable
rotation with an initial value of 30 degrees. -->
<SCRIPT language=JavaScript>
```

```
   var rotation = "30";

// The getSelectedValue() function returns the text associated
// with the value chosen from the pulldown list.

function getSelectedValue(sel) {
   return sel.options[sel.selectedIndex].text
 }

// The rot_surf() function retrieves the rotation value and
// executes the IDL command to re-draw the graphic. It is called
// when the button is clicked.

function rot_surf() {
   rotation = getSelectedValue(document.command_form.rot_value);
   document.SURFAPP.executeIDLCommand("SHADE_SURF, a,
       AZ="+rotation);
 }
</SCRIPT>
</HEAD>
<BODY>
<!-- JavaScript input controls must be contained in an
     HTML form. -->
<FORM NAME="command_form">

<!-- Create an IONGraphicApplet applet named "SURFAPP" that
generates some data and creates a shaded surface. Note that the
CODEBASE attribute is set to "../classes". This is the proper path
for the example as installed with the ION documentation
files.-->

<APPLET NAME="SURFAPP" CODE=com.rsi.ion.IONGraphicApplet.class
   CODEBASE="../classes" WIDTH=200 HEIGHT=200>
   <PARAM NAME="DEBUG_MODE" VALUE="YES">
   <PARAM NAME="SERVER_DISCONNECT" VALUE="NO">
   <PARAM NAME="DECOMPOSED_COLOR" VALUE="NO">
   <PARAM NAME="IDL_COMMAND_0"
          VALUE="a = EXP(-(SHIFT(DIST(30), 15, 15)/7)^2)">
   <PARAM NAME="IDL_COMMAND_1" VALUE="LOADCT, 5">
   <PARAM NAME="IDL_COMMAND_2" VALUE="SHADE_SURF, a">
</APPLET>
<BR>
<!-- Create the pulldown menu of rotation values -->
<SELECT NAME="rot_value" SIZE=1>
<OPTION VALUE=15>15
<OPTION VALUE=30 SELECTED>30
<OPTION VALUE=45>45
<OPTION VALUE=60>60
<OPTION VALUE=75>75
```

```
<OPTION VALUE=90>90
</SELECT>

<!-- Create the "Rotate Surface" button, which calls the
JavaScript function rot_surf().-->
<INPUT TYPE=BUTTON NAME="rot_button" VALUE="Rotate Surface"
onClick="rot_surf()">
</FORM>
</BODY>
</HTML>
```

See "Notes on the Differences Between the JavaScript and VBScript Versions" on page 115.

Example: Using VBScript

The following HTML code demonstrates the use of VBScript to interactively update an ION graphic. The example includes an IONGraphicApplet that displays a shaded surface, uses a VBScript `select` object to create a pulldown list of rotation values, and adds a button to rotate the surface to the selected angle. The line numbers are provided to aid in discussion; they are not part of the HTML code.

Note

The following script shown will work in Microsoft's Internet Explorer browser, but not in the Netscape Navigator browser.

```
<!-- Define the HTML header. Note that the VBScript is included in
the HEAD section. -->
<HTML>
<HEAD>
 <TITLE>Simple VBScript Applet Test</TITLE>
<!-- The script language is VBScript. We declare the variable
     rotation with an initial value of 30 degrees. -->
 <SCRIPT language=VBScript>
   Dim rotation
   rotation = "30"


// The rot_button_OnClick() subroutine retrieves the index of
// the value selected in the pulldown list, uses the index to
// retrieve the text value, and executes the IDL command to redraw
// the graphic.

 sub rot_button_OnClick()
   ind = document.command_form.rot_value.selectedIndex
   rotation = document.command_form.rot_value.options(ind).value
   document.SURFAPP.executeIDLCommand("SHADE_SURF, a,
        AZ="+rotation)
```

```
 end sub
 </SCRIPT>
</HEAD>
<BODY>

<!-- Create an IONGraphicApplet applet named "SURFAPP" that
generates some data and creates a shaded surface. Note that the
CODEBASE attribute is set to "../classes". This is the proper path
for the example as installed with the ION documentation
files. -->

<APPLET NAME="SURFAPP" CODE=com.rsi.ion.IONGraphicApplet.class
   CODEBASE="../classes" WIDTH=200 HEIGHT=200>
   <PARAM NAME="DEBUG_MODE" VALUE="YES">
   <PARAM NAME="SERVER_DISCONNECT" VALUE="NO">
   <PARAM NAME="DECOMPOSED_COLOR" VALUE="NO">
   <PARAM NAME="IDL_COMMAND_0"
          VALUE="a = EXP(-(SHIFT(DIST(30), 15, 15)/7)^2)">
   <PARAM NAME="IDL_COMMAND_1" VALUE="LOADCT, 5">
   <PARAM NAME="IDL_COMMAND_2" VALUE="SHADE_SURF, a">
</APPLET>

<!-- VBScript input controls must be contained in an
     HTML form. -->
<FORM NAME="command_form">
   <BR>

   <!-- Create the pulldown menu of rotation values. -->
   <SELECT NAME="rot_value" SIZE=1>
      <OPTION VALUE=15>15
      <OPTION VALUE=30 SELECTED>30
      <OPTION VALUE=45>45
      <OPTION VALUE=60>60
      <OPTION VALUE=75>75
      <OPTION VALUE=90>90
   </SELECT>

   <!-- Create the "Rotate Surface" button. The
         rot_button_OnClick() subroutine is called automatically
        when this button is clicked. -->
   <INPUT TYPE=BUTTON NAME="rot_button" VALUE="Rotate Surface">
</FORM>
</BODY>
</HTML>
```

Notes on the Differences Between the JavaScript and VBScript Versions

1. Interaction between the applet and the script language takes place in JavaScript statements in the Netscape Navigator version, and in VBScript statements in the Microsoft Internet Explorer version. The syntax of the scripting language is slightly different.
2. In the JavaScript version, the applet is included within the HTML FORM definition. Internet Explorer requires that the applet be located outside the FORM.

In JavaScript, you must explicitly tie a control (a button, for example) to a JavaScript function. VBScript automatically looks for a subroutine name based on the name of the button.

Tips and Tricks

This section includes suggestions that may be useful in some situations. Make sure your installation meets the criteria defined here before implementing any of these suggestions.

Local Netscape Users

If your installation is used only by a known set of users who all use Netscape's Navigator version 4 or later, you can eliminate the need to download the Java class files when an applet loads. Do the following:

1. Have each of your users install a copy of the *ion_release*`.jar` file in the `Program/java/classes` subdirectory of their local Netscape directory.
2. Remove the ARCHIVE attribute from the APPLET tag in your HTML code.

Note

The Java security mechanism requires that applet classes must be loaded from the server on which ION is running. This means that the approach described here will fail with a security error if the applet class files are not located in the `com/rsi/ion` subdirectory of the directory specified by the CODEBASE attribute.

Stop Methods

If your applet includes a stop() method, it will be invoked automatically when the browser leaves the browser window. In your applets, it is good practice to include a stop() method that closes the ION connection and does any other cleanup that may be necessary.

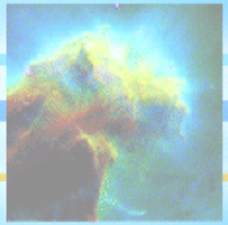
Client-side Animation

IDL's animation routines all rely on the IDL widget toolkit, and are thus not suitable for use with ION. You can, however, use IDL to create the individual frames of an animation and create an ION applet to build an array of frames and display the animation on the client side (in a browser or Java application).

An example application that does this sort of client-side animation is included in the ION distribution. Point your browser at the file `animation.html` in the `demo` subdirectory of the `examples` directory. The Java sources for the animation classes are included in the `src` subdirectory of the `classes` directory. Note that the animation demo relies on an IDL `.pro` file; see "Running the ION Java Examples" on page 48.

Tip

You can create an MPEG file on-the-fly with IDL and then supply a link to it. If the client browser has a common MPEG plug-in, you can play back the animation without requiring a special java applet. MPEG support in IDL requires a special license. For more information, contact your RSI sales representative.



Chapter 6:
ION Java Class and Method Reference

This chapter describes the ION Java classes and their methods. The following classes are covered in this chapter:

- IONCallableClient Class
- IONCommandDoneListener Interface
- IONContour / IONJContour Class
- IONDisconnectListener Interface
- IONGraphicsClient Class
- IONGrConnection / IONJGrConnection Class
- IONGrDrawable / IONJGrDrawable Class
- IONGrMap Class
- IONGrMapGrid Class
- IONGrPlot Class
- IONCanvas / IONJCanvas Class
- IONComplex Class
- IONDComplex Class
- IONDrawable Interface
- IONGraphicConnection Interface
- IONGrContour Class
- IONGrGraphic Class
- IONGrMapContinents Class
- IONGrMapImage Class
- IONGrSurface Class

- IONMap / IONJMap Class
- IONOffScreen Class
- IONPlot / IONJPlot Class
- IONVariable Class
- IONMouseListener Interface
- IONOutputListener Interface
- IONSurface / IONJSurface Class
- IONWindowingClient Class

Note

For descriptions of the ION Java applet classes, see Chapter 4, "Using ION's Pre-Built Applets".

How to Use this Chapter

The elements of the ION Java class library are documented alphabetically in this chapter. The page or pages describing each class include a description of the class declaration, which provides pointers to the Java class (or other ION class) that the class inherits from, if any. Note that this chapter does not provide documentation for the Java classes themselves; see your Java API reference materials for descriptions of the Java classes. Class methods are documented alphabetically (with the exception of the constructor method for the class, which is documented first) following the description of the class itself.

A description of each method follows its name. Beneath the general description of the method are a number of sections that describe the syntax for calling the method and its arguments (if any). These sections are described below.

Syntax

The “Syntax” section shows the proper syntax for calling the method.

Data Types

Java is a strongly-typed language, which means that input and output data variables must be created as or cast to the proper type before use. The “Syntax” description includes the data type of each variable specified. For example, the following is a syntax description for the ION method that sets the value of an IDL variable:

setIDLVariable(String *sName*, IONVariable *oVar*)

In this case, there are two arguments to the setIDLVariable method: *sName* and *oVar*. The word “String” defines *sName* as a variable of type string. Similarly, the word “IONVariable” defines *oVar* as a variable of type IONVariable.

Multiple Syntax Definitions

Many ION Java methods can be called in more than one way. In these cases, all of the available syntax definitions are listed together. For example, the following are all valid ways to call the setXValue method of the IONContour class:

setXValue(int *X[]*)

setXValue(float *X[]*)

setXValue(double *X[]*)

setXValue(String *sName*)

This means that the argument to the setXValue method can be either an integer, single-precision floating-point, or double-precision floating-point array, or a string value.

Optional Arguments

Arguments that are not required are included in the syntax definition enclosed in square brackets ([ ]). Italicized square brackets indicate an array, while non-italicized square brackets indicate that the enclosed arguments are optional. For example, the square brackets in this syntax definition indicate that the variable *X* is an array variable:

setXValue(int *X[]*)

The square brackets in the following syntax definition indicate that the *portNumber* argument is optional:

connect(String *hostname* [, int *portNumber*])

Case Sensitivity

ION object class and method names are displayed in mixed-case type. Unlike IDL, the Java language is case-sensitive — names of ION Java methods and classes must be entered with the same capitalization as shown in this reference section.

Italic Type

Arguments to ION procedures and functions — data or variables you must provide — are displayed in *italic* type.

Courier Type

Class declarations, syntax, and examples are shown in `courier`.

IDL Code

IDL functions, procedures, and keywords are displayed in UPPER CASE type. For example, the calling sequence for an IDL procedure looks like this:

CONTOUR, *Z* [, *X*, *Y*]

Arguments

The “Arguments” section describes each valid argument to the method. Note that these arguments are positional parameters that must be supplied in the order indicated by the method’s syntax.

Exceptions

The “Exceptions” section lists the ION exception values that are thrown when your error-handling code detects an error. For more information on exception handling, consult your Java manual. Refer to the following descriptions for each exception:

IOException

A network communication error was detected. Server is disconnected.

IONIllegalCommandException

The specified IDL command was illegal.

IONIsAnArrayException

The variable contains an array value.

IONLicenseException

An ION license could not be obtained.

IONNotAnArrayException

The IONVariable is not an array.

IONSecurityException

The specified IDL command is not allowed under the current ION security rules.

NumberFormatException

The variable is a string that cannot be converted.

UnknownHostException

The given hostname is unknown.

Example

Where appropriate, the “Example” section includes a short example that demonstrates the use of the method.

IONCallableClient Class

The IONCallableClient class provides mechanisms to handle communication with the server, execution of IDL commands, retrieval of IDL command log output, and getting and setting IDL variables on the ION Server.

In order to provide support for mouse operations (the IDL CURSOR procedure) the main thread that handles the Java event loop must not block during IDL command execution. If the main event thread blocked and the IDL server requested a mouse location, the client and the server would be in a deadlock condition. To prevent a deadlock condition, this class provides the sendIDLCommand() method which sends the command to the server for execution and returns, not waiting for the command to complete. The class is informed of the commands completion status through the IONCommandDoneListener interface.

Class Declaration

```
public class IONCallableClient
```

Methods

- **IONCallableClient() —** Constructs an object of the IONCallableClient class.
- **addIONCommandDoneListener() —** Adds a “command done” listener to the client object.
- **addIONDisconnectListener() —** Adds a “disconnect” listener to the client object.
- **addIONOutputListener() —** Adds an “output” listener to the client object.
- **connect() —** Connects to the server.
- **disconnect() —** Shuts down the ION Server and disconnects.
- **executeIDLCommand() —** Executes an IDL command on the ION Server. Control is not retuned until the command has been executed.
- **getClientVersion()** — Returns the current version of ION.
- **getConnectionType()** — Returns the type of connection in use.
- **getIDLVariable() —** Gets the value of an IDL variable on the ION Server.
- **removeIONCommandDoneListener() —** Removes a “command done” listener from the client object.

- **removeIONDisconnectListener() —** Removes a “disconnect” listener from the client object.
- **removeIONOutputListener() —** Removes a listener from the client object.
- **sendIDLCommand() —** Posts an IDL command to the ION Server. Control is returned as soon as the command is sent.
- **setConnectionMethod() —** Sets the type of connection for the client.
- **setConnectionTimeout()** — Sets the timeout for socket connections.
- **setIDLVariable() —** Sets the value of an IDL variable on the ION Server.

Subclasses

- IONGraphicsClient Class

Constants

- HTTP_CON
- SOCK_CON
- BEST_CON
- NO_CON

IONCallableClient()

The IONCallableClient() method constructs an IONCallableClient object. When it returns, all internal initialization is complete. No connection is made at this time.

Syntax

public IONCallableClient()

Arguments

None

Example

```
IONCallableClient client = new IONCallableClient();
```

addIONCommandDoneListener()

The addIONCommandDoneListener() method is used to register an object that implements the `IONCommandDoneListener` interface with this object.

Syntax

public final void addIONCommandDoneListener(IONCommandDoneListener *listener*)

Arguments

listener

An object that implements the IONCommandDoneListener interface. The listener is added to the internal listener list.

Exceptions

None

addIONDisconnectListener()

The addIONDisconnectListener() method adds an object that implements the IONDisconnectListener interface to the internal list of registered listeners. When the client/server connection is disconnected, callback methods are called on the objects that are registered with the IONCallableClient.

Syntax

public final void addIONDisconnectListener(IONDisconnectListener *listener*)

Arguments

listener

An object that implements the IONDisconnectListener interface. The listener is added to the internal listener list.

Exceptions

None.

addIONOutputListener()

The addIONOutputListener() method is used to add an object that implements the IONOutputListener interface to the internal list of listeners kept by this object. When any IDL output is sent from the server to the client, the output is sent to the objects contained in the listener list through the callback method defined by the IONOutputListener interface. This provides an efficient method of passing IDL output to the client and mimics the Java 1.1 event model.

Syntax

public final void addIONOutputListener(IONOutputListener *listener*)

Arguments

listener

This is an object that implements the IONOuptutListener interface. This interface defines the format of the callback method used to pass IDL output to the listener object. The listener is added to the internal listener list.

Exceptions

None.

connect()

The connect() method establishes a connection between the client and the ION Server. The client and the server make validity checks and the communication protocol is established. If hostname and port information for both the ION Server and the ION HTTP Tunnel Broker are supplied, the connection type is set automatically to “BEST_CON”. See “setConnectionMethod()” on page 132 for details on setting other connection types.

Syntax

public void connect(String *sHostname*)

public void connect(String *sHostname*, int *iPort*)

public void connect(String *sHostname*, int *iPort*, String *sHttpHost*, int *iHttpPort*)

Arguments

sHostname

The name of the host that the ION Server or HTTP broker is running on. If the class is being created as part of a Java applet, most web browsers require that the host name be the same host that is serving the applet. If the connection type is either "SOCK_CON" or "BEST_CON", this argument specifies the host on which the ION Server is running. If the connection type is "HTTP_CON" and the *sHttpHost* argument is not specified, this argument specifies the host on which the HTTP Tunnel Broker is running.

iPort

The port number to use when connecting to the ION Server. If this number is not provided the default port number is used. If the connection type is either "SOCK_CON" or "BEST_CON", this argument specifies the port on which the ION Server is running. If the connection type is "HTTP_CON" and the *iHttpPort* argument is not specified, this argument specifies the port on which the HTTP Tunnel Broker is running.

sHttpHost

The name of the host on which the ION HTTP Tunnel Broker is running. If all four arguments to the connect method are supplied, the connection type is automatically set to "BEST_CON".

iHttpPort

The port number to use when connecting to the ION HTTP Tunnel Broker. If all four arguments to the connect method are supplied, the connection type is automatically set to "BEST_CON".

Exceptions

IOException, UnknownHostException, IONLicenseException

disconnect()

Call the disconnect() method to shut down the ION Server (the daemon remains active), close the connection between the server and the client and free any resources that were being used by the connection. Once this method has been called, the object should be considered invalid and not used.

Syntax

public final void disconnect()

Arguments

None.

Exceptions

None.

executeIDLCommand()

The executeIDLCommand() method sends an IDL command to the ION Server for execution. The function returns when the command is complete on the server.

Syntax

public int executeIDLCommand(String *sCommand*)

Return Value

The function returns 0 if the IDL command executed successfully, or the value of the IDL system variable !ERROR if the IDL command did not execute successfully.

Arguments

sCommand

The IDL Command that is to be executed on the IDL server. The use of the “$” IDL command (to open a shell or command window) and the line continuation character ($) are prohibited (for security reasons, and because they can hang the server).

Exceptions

IOException, IONIllegalCommandException, IONSecurityException

getClientVersion()

The getClientVersion() method returns a string representing the current version of ION, for example, “ION 2.0”.

Syntax

public final String getClientVersion()

Return Value

The method returns a string containing the current version of ION.

Arguments

None.

Exceptions

None.

getConnectionType()

The getConnectionType() method returns the type of connection in use. See “Configuring The ION HTTP Tunnel Broker” on page 31 for details on connection types.

Syntax

public final int getConnectionType()

Return Value

The function returns one of the following values:

- HTTP_CON — The client uses the ION HTTP Tunnel Broker exclusively.
- SOCK_CON — The client uses a normal ION socket connection exclusively.
- BEST_CON — The client makes the best connection it can.

These values are defined as constants in the IONCallableClient class definition. The example below shows how to compare the returned value with the value defined in the IONCallableClient class.

Arguments

None.

Exceptions

None.

Example

To determine whether the connection in use is a socket-only connection, use a statement like the following:

```
if( getConnectionType() == IONCallableClient.SOCK_CON )
```

getIDLVariable()

The getIDLVariable() method requests the value of an IDL variable from the server. The value of the variable is then returned as an IONVariable object. If the variable does not exist on the server, it is created as an undefined type.

Syntax

public final IONVariable getIDLVariable(String *sName*)

Return Value

The function returns the value of the requested IDL variable in an IONVariable object.

Arguments

sName

The variable name whose value is desired.

Exceptions

IOException

removeIONCommandDoneListener()

The removeIONCommandDoneListener() method removes an object that implements the IONCommandDoneListener interface from the list of listeners maintained by this object. If the listener is not contained in the internal list of listeners, the method returns silently.

Syntax

public final void removeIONCommandDoneListener(IONCommandDoneListener *listener*)

Arguments

listener

An object that implements the IONCommandDoneListener interface that is to be removed from the internal listener list.

Exceptions

None.

removeIONDisconnectListener()

The removeIONDisconnectListener() method removes an object that implements the IONDisconnectListener interface from the internal Disconnect callback list. If the listener is not contained in the internal list of listeners, the method returns silently.

Syntax

public final void removeIONDisconnectListener(IONDisconnectListener *listener*)

Arguments

listener

The object that implements an IONDisconnectListener interface that should be removed from the listener callback list.

Exceptions

None.

removeIONOutputListener()

The removeIONOutputListener() method removes the given listener from the internal list of listeners. If the listener is not contained in the internal list of listeners, the method returns silently.

Syntax

public final void removeIONOutputListener(IONOutputListener *listener*)

Arguments

listener

The object that implements an IONOutputListener interface that should be removed from the listener callback list.

Exceptions

None.

sendIDLCommand()

The sendIDLCommand() method asynchronously sends an IDL command to the ION Server for execution. The IDL command is posted to the server for execution and the function immediately returns. Notification of the commands completion is performed via the IONCommandDoneListener interface.

Syntax

public void sendIDLCommand(String *sCommand*)

Arguments

sCommand

The IDL Command that is to be executed on the IDL server. The use of the “$” IDL command (to open a shell or command window) and the line continuation character ($) are prohibited (for security reasons, and because they can hang the server).

Exceptions

IOException

setConnectionMethod()

The setConnectionMethod() method sets the type of connection for the client. See "Configuring The ION HTTP Tunnel Broker" on page 31 for details on connection types.

Syntax

public final void setConnectionMethod(int *iType*)

Arguments

iType

Set the Type argument to one of the three following values:

- HTTP_CON — The client uses the ION HTTP Tunnel Broker exclusively.
- SOCK_CON — The client uses a normal ION socket connection exclusively.
- BEST_CON — The client first tries to use a SOCK_CON. If this times out, the client uses the HTTP_CON.

These values are defined as constants in the IONCallableClient class definition.

Example

To set the connection type to HTTP-only, use the following statement:

```
setConnectionType(IONCallableClient.HTTP_CON)
```

setConnectionTimeout()

The setConnectionTimout() method sets the timeout period of a socket connection.

Syntax

public final void setConnectionTimeout(long *iTime*)

Arguments

iTime

The number of milliseconds the connection should stay alive without any client requests.

Exceptions

None.

setIDLVariable()

The setIDLVariable() method sets the value of a variable in the ION Server. If the variable doesn't exist, it is created.

Syntax

public final void setIDLVariable(String *sName*, IONVariable *oVar*)

Arguments

sName

The name of the variable to set on the server.

oVar

An object of type IONVariable that contains the value of the variable.

Exceptions

IOException

IONCanvas / IONJCanvas Class

Objects of the IONCanvas class represents a visible drawing area in which graphic output can be drawn. The IONJCanvas class is the Swing implementation of the IONCanvas class.

Class Declaration

```
public class IONCanvas
             extends Canvas
             implements IONDrawable

public class IONJCanvas
             extends JComponent
             implements IONDrawable
```

Methods

- **IONCanvas() / IONJCanvas()** — Constructs an object of the IONCanvas / IONJCanvas class.

  Note

  The following methods have been deprecated in ION 1.4. These methods will continue to function as specified in ION 1.2, but it is recommended that you implement one of the Java methods, MouseListener or MouseMotionListener, which have more robust functionality.

- **addIONMouseListener()** — Adds a MouseListener object to the current canvas object.
- **getDownButtons()** — Reports mouse button status.
- **getImage()** — Returns the image that is being drawn.
- **getIONGraphics()** — Returns an ION graphics context for the device.
- **getMousePos()** — Reports position of the mouse cursor on the canvas object.
- **removeIONMouseListener()** — Removes a MouseListener object from the current canvas object.

See also the descriptions of the IONDrawable Interface and the Java Canvas class.

Subclasses

- IONGrDrawable / IONJGrDrawable Class

IONCanvas() / IONJCanvas()

The IONCanvas() method constructs an IONCanvas object of the given size. This object can then be placed in a Java AWT tree. The IONJCanvas() method constructs an IONJCanvas object.

Syntax

public IONCanvas(int *width*, int *height*)

public IONJCanvas(int *width*, int *height*)

Arguments

width

The width of the canvas.

height

The height of the canvas.

Exceptions

None.

addIONMouseListener()

The addIONMouseListener() method sets the object that implements the IONMouseListener interface as the current Mouse listener. When a mouse event of the requested type is detected, the mouse listener interface callback function is called. Only one mouse listener is active at one time. Note that only one mouse listener is allowed at a time. Any previously set mouse listener is removed.

Syntax

public final void addIONMouseListener(IONMouseListener *listener*, int *breq*)

Arguments

listener

Object that implements the mouse listener interface request.

breq

The mouse event type that is requested. This integer is a bit file that contains one or more of the following values:

- int IONMouseListener.ION_MOUSE_DOWN
- int IONMouseListener.ION_MOUSE_MOVE
- int IONMouseListener.ION_MOUSE_UP
- int IONMouseListener.ION_MOUSE_ANY

Exceptions

None.

getDownButtons()

The getDownButtons() method returns the current state of the mouse buttons in the canvas. The return value is a bit field where bit 1 is mouse button 1, bit 2 is mouse button 2 and bit three is mouse button 3. If a bit is set, the specific mouse button is down.

Syntax

public final int getDownButtons()

Arguments

None.

Exceptions

None.

Example

```
int iState = getDownButtons();
```

getImage()

The getImage() method returns the image of the current drawing area.

Syntax

public abstract Image getImage()

Arguments

None

Exceptions

None

Example

```
Image im = draw.getImage();
```

getIONGraphics()

The getIONGraphics() method returns a Graphics object that you can use to get graphics information on ION's drawing buffer or draw directly to. Unlike the getGraphics() method, getIONGraphics() allows you to affect the actual IDL drawable area. For example, you would use the getIONGraphics() method when manipulating the buffer using the COPY keyword to IDL's DEVICE procedure.

Syntax

public abstract Graphics getIONGraphics()

Arguments

None

Exceptions

None

Example

```
Graphics g = draw.getIONGraphics();
```

getMousePos()

The getMousePos() method returns the current location of the mouse cursor in the canvas.

Syntax

public Point getMousePos()

Arguments

None.

Exceptions

None.

Example

```
Point pt = getMousePos();
```

removeIONMouseListener()

The removeIONMouseListener() method removes a mouse listener from the object. If the given mouse listener is not the current listener, the function exits quietly.

Syntax

public final void removeIONMouseListener(IONMouseListener *listener*)

Arguments

listener

The listener to remove.

Exceptions

None.

Example

```
removeIONMouseListener(listener);
```

IONCommandDoneListener Interface

The IONCommandDoneListener interface defines the method that a class must implement to receive notification that an IDL command has completed. The listener object must be registered with the addIONCommandDoneListener() call.

Class Declaration

```
public interface IONCommandDoneListener
```

Methods

- **IONCommandComplete() —** Reports on the status of a completed command.

Implementing Classes

- IONGraphicApplet
- IONGrConnection / IONJGrConnection Class

IONCommandComplete()

Call the IONCommandComplete() method when a command that was sent to the IDL server is complete.

Syntax

public abstract void IONCommandComplete(int *iStatus*, int *iIDLStatus*)

Arguments

iStatus

A value that indicates the status of the processing of the IDL command. This value is one of the following constants that are part of this class:

- ION_COMM_OK - Command is OK.
- ION_COMM_SECURITY - Command security error.
- ION_COMM_INVALID - Command was invalid.

iIDLStatus

Indicates the success or failure of the execution of the IDL command. A value of 0 indicates that the command was successful. If the command was not successful, iIDLStaus contains the value of !ERROR in the IDL session.

Exceptions

None.

Example

```
public void IONCommandComplete(iStatus, iIDLStatus);
```

IONComplex Class

The IONComplex class represents a complex number.

Class Declaration

```
public class IONComplex
             extends Number
```

Methods

- **IONComplex()** — Constructs an object of the IONComplex class.
- **doubleValue()** — Returns the real portion of the number as a double.
- **floatValue()** — Returns the real portion of the number as a float.
- **getDImaginary()** — Returns the imaginary portion of the number as a double.
- **getImaginary()** — Returns the imaginary portion of the number as a float.
- **intValue()** — Returns the real portion of the number as an integer.
- **longValue()** — Returns the real portion of the number as a long.
- **toString()** — Returns the string value of the real portion of the number.

IONComplex()

The IONComplex() method constructs an object of the IONComplex class.

Syntax

public IONComplex(float *r*, float *i*)

Arguments

r

The real portion of the number.

i

The imaginary portion of the number.

Exceptions

None.

Example

```
IONComplex complexvar = new IONComplex(3.0, 2.0);
```

doubleValue()

The doubleValue() method returns the real portion of the complex number as a double-precision floating-point value.

Syntax

public final double doubleValue()

Arguments

None.

Exceptions

None.

Example

```
double d = complexvar.doubleValue();
```

floatValue()

The floatValue() method returns the real portion of the complex number as a single-precision floating-point value.

Syntax

public final float floatValue()

Arguments

None.

Exceptions

None.

Example

```
float f = complexvar.floatValue();
```

getDImaginary()

The getDImaginary() method returns the imaginary portion of the complex number as a double-precision floating-point value.

Syntax

public final double getDImaginary()

Arguments

None.

Exceptions

None.

Example

```
double d = complexvar.getDImaginary();
```

getImaginary()

The getImaginary() method returns the imaginary portion of the complex number as a single-precision floating-point value.

Syntax

public final float getImaginary()

Arguments

None.

Exceptions

None.

Example

```
float i = complexvar.getImaginary();
```

intValue()

The intValue() method returns the real portion of the complex number as an integer value.

Syntax

public final int intValue()

Arguments

None.

Exceptions

None.

Example

```
int i = complexvar.intValue();
```

longValue()

The longValue() method returns the real portion of the complex number as a long-integer value.

Syntax

public final long longValue()

Arguments

None.

Exceptions

None.

Example

```
long l = complexvar.longValue();
```

toString()

The toString() method returns the real portion of the complex number as a string value.

Syntax

public final String toString()

Arguments

None.

Exceptions

None.

Example

```
String s = complexvar.toString();
```

IONContour / IONJContour Class

The IONContour class extends the IONGrDrawable class and contains an IONGrContour object to provide an easy way of drawing IDL contours. It can be inserted into an AWT tree.

The IONJContour class extends the IONJGrDrawable class and contains an IONGrContour object. It can be inserted into a Swing component tree.

Class Declaration

```
public class IONContour
             extends IONGrDrawable

public class IONJContour
             extends IONJGrDrawable
```

Methods

- **IONContour() / IONJContour() —** Constructs an object of the IONContour/IONJContour class.
- **draw() —** Produces and displays the graphic on the drawing surface of this class.
- **getProperty() —** Gets the value of a property.
- **setNoErase()** — Specifies whether the object should be erased when another object is drawn.
- **setProperty() —** Sets a property for the graphic.
- **setXValue() —** Sets the X value of the contour.
- **setYValue() —** Sets the Y value of the contour.
- **setZValue() —** Sets the Z data of the contour.

IONContour() / IONJContour()

The IONContour() method constructs an object of the IONContour class. The IONJContour method constructs an object of the IONJContour class.

Syntax

Note

The following is the syntax for the IONContour() method. For the IONJContour() method, replace IONContour with IONJContour.

public IONContour(int *iWidth*, int *iHeight*)

public IONContour(int *iWidth*, int *iHeight*, int *Z[][]*)

public IONContour(int *iWidth*, int *iHeight*, float *Z[][]*)

public IONContour(int *iWidth*, int *iHeight*, double *Z[][]*)

public IONContour(int *iWidth*, int *iHeight*, int *Z[]*)

public IONContour(int *iWidth*, int *iHeight*, float *Z[]*)

public IONContour(int *iWidth*, int *iHeight*, double *Z[]*)

public IONContour(int *iWidth*, int *iHeight*, String *sName*)

public IONContour(int *iWidth*, int *iHeight*, int *Z[][]*, int *X[]*, int *Y[]*)

public IONContour(int *iWidth*, int *iHeight*, float *Z[][]*, float *X[]*, float *Y[]*)

public IONContour(int *iWidth*, int *iHeight*, double *Z[][]*, double *X[]*, double *Y[]*)

public IONContour(int *iWidth*, int *iHeight*, int Z[], int *X[]*, int *Y[]*)

public IONContour(int *iWidth*, int *iHeight*, float *Z[]*, float *X[]*, float *Y[]*)

public IONContour(int *iWidth*, int *iHeight*, double *Z[]*, double *X[]*, double *Y[]*)

public IONContour(int *iWidth*, int *iHeight*, String *sZName*, String *sXName*, String *sYName*)

Arguments

iWidth

The width of the plot in pixels.

iHeight

The height of the plot in pixels.

Z

The Z values (data) to use in the contour.

sName, sZName

The name of the IDL variable to use for the Z (data) values of the contour.

X

An array holding the values for the X coordinates of the grid.

Y

An array holding the values for the Y coordinates of the grid.

sXName

The name of the IDL variable holding the values for the X coordinates of the grid.

sYName

The name of the IDL variable holding the values for the Y coordinates of the grid.

Exceptions

None.

draw()

The draw() method produces and display a graphic in the drawing area that makes up this object.

Syntax

public void draw()

Arguments

None.

Exceptions

None.

getProperty()

The getProperty() method retrieves the current value of a property.

Syntax

public final IONVariable getProperty(String *sName*)

Argument

sName

The name of the property.

Properties Supported

The following IDL Contour properties are supported by IONContour.[get,set]Property. Refer to the IDL documentation on keywords available for use with the CONTOUR procedure for an explanation of each property:

BACKGROUND, CELL_FILL, CHARSIZE, CLIP, CLOSED, COLOR, C_ANNOTATION, C_CHARSIZE, C_COLORS, C_LABELS, C_LINESTYLE, C_ORIENTATION, C_SPACING, DATA, DEVICE, DOWNHILL, FILL, FOLLOW, FONT, IRREGULAR, LEVELS, MAX_VALUE, MIN_VALUE, NLEVELS, NOCLIP, NODATA, NOERASE, NORMAL, OVERPLOT, PATH_DATA_COORDS, PATH_FILENAME, PATH_INFO, PATH_XY, POLAR, POSITION, SUBTITLE, T3D, TICKLEN, TITLE, TRIANGULATION, XCHARSIZE/YCHARSIZE/ZCHARSIZE, XGRIDSTYLE/YGRIDSTYLE/ZGRIDSTYLE, XLOG, YLOG, XMARGIN/YMARGIN/ZMARGIN, XMINOR/YMINOR/ZMINOR, XRANGE/YRANGE/ZRANGE, XSTYLE/YSTYLE/ZSTYLE, XTICKFORMAT/YTICKFORMAT/ZTICKFORMAT, XTICKLEN/YTICKLEN/ZTICKLEN, XTICKNAME/YTICKNAME/ZTICKNAME, XTICKS/YTICKS/ZTICKS, XTICKV/YTICKV/ZTICKV, XTITLE/YTITLE/ZTITLE, XTICKINTERVAL/YTICKINTERVAL/ZTICKINTERVAL, XTICKLAYOUT/YTICKLAYOUT/ZTICKLAYOUT, XTICKUNITS/YTICKUNITS/ZTICKUNITS, XTICK_GET/YTICK_GET/ZTICK_GET, ZAXIS, ZVALUE

Exceptions

None.

Example

```
IONVariable value = getProperty(Property);
```

setNoErase()

The setNoErase() method of the IONContour class overrides setNoErase() in the IONGrDrawable class. The setNoErase() method of the IONJContour class overrides setNoErase() in the IONJGrDrawable class. See “setNoErase()” on page 195 for the syntax of this method.

setProperty()

The setProperty() method sets a property for the contour object.

Syntax

public final void setProperty(String *sName*, IONVariable *vValue*)

Arguments

sName

The name of the property to set.

vValue

The value of the property.

Properties Supported

The IDL Contour properties are supported by IONContour.[get,set]Property are the same as those covered in “getProperty()” on page 149. Refer to the IDL documentation on keywords available for use with the CONTOUR procedure for an explanation of each property.

Exceptions

None.

setXValue()

The setXValue() method resets the X value of the contour.

Syntax

public void setXValue(int *X[]*)

public void setXValue(float *X[]*)

public void setXValue(double *X[]*)

public void setXValue(String *sName*)

Arguments

X

The new X value of the contour.

sName

The name of an IDL variable that contains the new X value of the contour.

Exceptions

None.

setYValue()

The setYValue() method resets the Y value of the contour.

Syntax

public void setYValue(int *Y[]*)

public void setYValue(float *Y[]*)

public void setYValue(double *Y[]*)

public void setYValue(String *sName*)

Arguments

Y

The new Y value of the contour.

sName

The name of the IDL variable that contains the new Y value of the contour.

Exceptions

None.

setZValue()

The setZValue() method resets the Z value of the contour.

Syntax

public void setZValue(int *Z[]*)

public void setZValue(float *Z[]*)

public void setZValue(double *Z[]*)

public void setZValue(int *Z[][]*)

public void setZValue(float *Z[][]*)

public void setZValue(double *Z[][]*)

public void setZValue(String *sName*)

Argument

Z

The new Z value of the contour.

sName

The name of the IDL variable that contains the new Z value of the contour.

Exceptions

None.

IONDComplex Class

The IONDComplex class represents a double-precision complex number.

Class Declaration

```
public class IONDComplex
             extends Number
```

Methods

- **IONDComplex()** — Constructs an object of the IONComplex class.
- **doubleValue()** — Returns the double value of the real portion of the number.
- **floatValue()** — Returns the float value of the real portion of the number.
- **getDImaginary()** — Returns the imaginary value as a double.
- **getImaginary()** — Returns the imaginary value of the number.
- **intValue()** — Returns the int value of the real portion of the number.
- **longValue()** — Returns the long value of the real portion of the number.
- **toString()** — Returns the string value of the real portion of the number.

IONDComplex()

The IONDComplex() method constructs an object of the IONDComplex class.

Syntax

public IONDComplex(double *r*, double *i*)

Arguments

r

The real portion of the number.

i

The imaginary portion of the number.

Exceptions

None.

Example

```
IONDComplex dcomplexvar = new IONDComplex(3.0, 2.0);
```

doubleValue()

The doubleValue() method returns the real portion of the complex number as a double-precision floating-point value.

Syntax

public final double doubleValue()

Arguments

None.

Exceptions

None.

Example

```
double d = dcomplexvar.doubleValue();
```

floatValue()

The floatValue() method returns the real portion of the complex number as a single-precision floating-point value.

Syntax

public final float floatValue()

Arguments

None.

Exceptions

None.

Example

```
float f = dcomplexvar.floatValue();
```

getDImaginary()

The getDImaginary() method returns the imaginary portion of the complex number as a double-precision floating-point value.

Syntax

public final double getDImaginary()

Arguments

None.

Exceptions

None.

Example

```
double d = dcomplexvar.getDImaginary();
```

getImaginary()

The getImaginary() method returns the imaginary portion of the complex number as a single-precision floating-point value.

Syntax

public final float getImaginary()

Arguments

None.

Exceptions

None.

Example

```
float i = dcomplexvar.getImaginary();
```

intValue()

The intValue() method returns the real portion of the complex number as an integer value.

Syntax

public final int intValue()

Arguments

None.

Exceptions

None.

Example

```
int i = dcomplexvar.intValue();
```

longValue()

The longValue() method returns the real portion of the complex number as a long-integer value.

Syntax

public final long longValue()

Arguments

None.

Exceptions

None.

Example

```
long l = dcomplexvar.longValue();
```

toString()

The toString() method returns the real portion of the complex number as a string value.

Syntax

public final String toString()

Arguments

None.

Exceptions

None.

Example

```
String s = dcomplexvar.toString();
```

IONDisconnectListener Interface

The IONDisconnectListener interface defines a method that is called when the connection between the client and the server is disconnected. The reason for disconnection is defined by one of the constants that are a part of this interface.

To use this interface, your class must implement this interface and then register the listener using addIONDisconnectListener. See IONCallableClient Class from more information.

Class Declaration

```
public interface IONDisconnectListener
```

Methods

- **IONDisconnection() —** If registered, the method is called when client and server are disconnected.

Implementing Classes

None

Constants

- ION_DIS_OK - Normal disconnection due to disconnect() method being called.
- ION_DIS_ERR - Disconnection caused by an error. Normally due to an interruption in the communication channel.
- ION_DIS_SERVER - Disconnection due to server shutdown.

IONDisconnection()

If registered, the IONDisconnection() method is called when the connection between the client and the server is broken to report the reason for disconnection.

Syntax

public abstract void IONDisconnection(int iStatus)

Arguments

iStatus

An integer corresponding to the reason for the disconnection. These are defined by the constants mentioned in the previous section.

Exceptions

None

Example

```
import javax.swing.*;
import java.io.*;
import java.net.*;
import com.rsi.ion.*;

public class disconnectEx extends JApplet
                          implements IONDisconnectListener {

  IONGrConnection ionCon;
  IONGrDrawable iondraw;

  public void init() {
    ionCon=new IONGrConnection();
    ionCon.addIONDisconnectListener(this);

    // connect to server and do ION commands
  }

public void IONDisconnection(int iStatus) {
    if (iStatus != IONDisconnectListener.ION_DIS_OK)
       System.out.println("ION disconnection error.");

    // do anything else to clean-up

  }
}
```

IONDrawable Interface

The IONDrawable interface defines the methods that an object must implement to act as an ION drawable object. An IONDrawable is an object that can be drawn to by an IONGraphicsClient. IONCanvas and IONOffScreen are both implementations of IONDrawable.

Class Declaration

```
public interface IONDrawable
```

Methods

No public methods.

Implementing Classes

IONCanvas / IONJCanvas Class, IONOffScreen Class

- **createImage()** — Creates an offscreen image.
- **getImage()** — Returns the image that is being drawn.
- **getIONGraphics()** — Returns an ION graphics context for the device.

createImage()

Use the createImage() method to create an image of a given size.

Syntax

public abstract Image createImage(int *width*, int *height*)

Arguments

width

The width of the requested image

height

The height of the requested image

Exceptions

None

Example

```
Image im = draw.createImage(300, 300);
```

getImage()

The getImage() method returns the image of the current drawing area.

Syntax

public abstract Image getImage()

Arguments

None

Exceptions

None

Example

```
Image im = draw.getImage();
```

getIONGraphics()

The getIONGraphics() method returns a Graphics object that you can use to get graphics information on ION's drawing buffer or draw directly to. Unlike the getGraphics() method, getIONGraphics() allows you to affect the actual IDL drawable area. For example, you would use the getIONGraphics() method when manipulating the buffer using the COPY keyword to IDL's DEVICE procedure.

Syntax

public abstract Graphics getIONGraphics()

Arguments

None

Exceptions

None

Example

```
Graphics g = draw.getIONGraphics();
```

IONGraphicsClient Class

The IONGraphicsClient class provides mechanisms to handle the processing of a graphic primitive data set from the IDL server. Information sent by the server is read by mechanisms provided by the super class IONCallableClient.

Class Declaration

```
public class IONGraphicsClient
             extends IONCallableClient
             implements IONMouseListener, IONGR2PropListener
```

Methods

- **IONGraphicsClient() —** Constructs an object of the IONGraphicsClient class.
- **addIONDrawable() —** Adds an object that implements the IONDrawable interface (windows or off-screen images).
- **connect() —** Connects the client with the ION Server.
- **copyArea() —** Copies an area from one drawable to another.
- **getCurrentIndex() —** Gets the index of the current drawable.
- **getIONDrawableIndices() —** Gets a list of assigned drawable indices.
- **getNumIndices() —** Gets the number of drawable indices allocated.
- **readImage() —** Reads the current contents of the drawable
- **removeIONDrawable() —** Removes an object from the internal list of IONDrawables maintained by this object.
- **setDecomposed() —** Sets decomposed mode on the connection.
- **setIONDrawable() —** Sets the current drawable.

Subclasses

IONGrConnection / IONJGrConnection Class, IONWindowingClient Class

IONGraphicsClient()

The IONGraphicsClient() method constructs an object of the IONGraphicsClient class.

Syntax

public IONGraphicsClient()

Arguments

None.

Exceptions

None.

Example

```
IONGraphicsClient iclient = new IONGraphicsClient();
```

addIONDrawable()

The addIONDrawable() method adds an object that implements the IONDrawable interface to the internal list of drawing areas maintained by this object. An IONDrawable represents an area that graphic primitives can be rendered onto. When the window is added to this class, that drawing area is made the current drawing area being used for graphical output. The developer has the option of telling ION what index to use and also requesting that the method send information about the new drawable to the server. The function returns the window index number that is used by IDL to reference the drawing area.

Syntax

public final int addIONDrawable(IONDrawable *drawable*)

public final int addIONDrawable(IONDrawable *drawable*, int *index*)

public final int addIONDrawable(IONDrawable *drawable*, boolean *bSendAttr*)

public final int addIONDrawable(IONDrawable *drawable*, int *index*, boolean *bSendAttr*)

Arguments

drawable

An object that implements the IONDrawable interface.

index

The index to assign to the drawable. If no index is supplied, a free index is used.

bSendAttr

If true, the server is notified of the change to the current drawable.

Return Value

The function returns the window index number that is used by IDL to reference the drawable.

Exceptions

None.

Example

```
int iIndex = addIONDrawable( drawable);
int iIndex = addIONDrawable( drawable, index);
```

connect()

The connect() method establishes a connection between the client and the IDL server. The client and the server make validity checks and the communication protocol is established.

Syntax

public void connect(String *sHostname*)

public void connect(String *sHostname*, int *iPort*)

Arguments

sHostname

The name of the host that the ION Server is running on. If the class is being created as part of a Java applet, most web browsers require that the host name be the same host that the applet is being served from.

iPort

The port number to use when connecting to the IDL server. If this number is not provided the default port number is used.

Exceptions

IOException, UnknownHostException, IONLicenseException

copyArea()

The copyArea() method copies an area from one drawable to another.

Syntax

public void copyArea(int *iSource*, int *iDest*, int *x*, int *y*, int *width*, int *height*, int *x2*, int *y2*)

Arguments

iSource

The index of the source drawable.

iDest

The index of the destination drawable.

x, y

The lower left corner of the area to copy.

width, height

The dimensions of the copy area.

x2, y2

The location of the lower left corner of the copy area in the destination.

Exceptions

None.

getCurrentIndex()

The getCurrentIndex() method retrieves the index of the current drawable.

Syntax

public int getCurrentIndex()

Return Value

The current drawable index. If no drawable is current, -1 is returned.

Arguments

None.

Exceptions

None.

Example

```
int index = getCurrentIndex();
```

getIONDrawableIndices()

The getIONDrawableIndices() method fills an array with the indices of the available drawables.

Syntax

public void getIONDrawableIndices(int *iIndices[]*))

Arguments

iIndices

An array of length getNumIndices that will be filled with the index values.

Exceptions

None.

getNumIndices()

The getNumIndices() method returns the number of drawable indices currently allocated.

Syntax

public int getNumIndices()

Return Value

The number of indices.

Arguments

None.

Exceptions

None.

Example

```
int num = getNumIndices();
```

readImage()

Use the readImage() method to read the contents of the current drawable.

Syntax

readImage()

readImage(int *x0*, int *y0*, int *width*, int *height*)

Arguments

x0

The x start position of the rectangle to read

y0

The y start position of the rectangle to read

width

The width of the rectangle to read

height

The height of the rectangle to read

Exceptions

None

Example

```
Image im = readImage();
Image im = readImage( x0, y0, width, height);
```

removeIONDrawable()

The removeIONDrawable() method removes an object that implements the IONDrawable interface from the internal list of IONDrawable objects.

Syntax

public IONDrawable removeIONDrawable(IONDrawable *drawable*)

public IONDrawable removeIONDrawable(int *index*)

Return Value

This method returns a reference to the removed IONDrawable object.

Arguments

drawable

The drawable to remove.

index

The index of the drawable to remove.

Exceptions

None.

setDecomposed()

The setDecomposed() method sets an individual connection to the ION Server to use *decomposed color mode*. The decomposed color mode setting determines how ION will display graphics on a True color (24-bit or 32-bit color) device. If the argument is true (the default) a pixel value is treated as an RGB triplet. If the argument is false, the red component of the pixel is treated as an index into the current color table. For more information on decomposed color mode, see the documentation for the DECOMPOSED keyword to the DEVICE procedure in the *IDL Reference Guide*.

Once set, decomposed color mode applies to all drawables associated with a given connection.

Syntax

public final void setDecomposed(boolean *bDecomposed*)

Arguments

bDecomposed

If bDecomposed is set to True, pixel values are interpreted as RGB triplets. (This is the default behavior.) If bDecomposed is set to False, the first eight bits of the pixel value (the red portion) are used as an index value into the currently loaded IDL color table.

Exceptions

None.

setIONDrawable()

The setIONDrawable() method selects which IONDrawable to use from the internal list of IONDrawable objects.

Syntax

public final boolean setIONDrawable(int *iIndex*)

Arguments

iIndex

The index that was returned from the addIONDrawable() method when the object was added.

Return Value

The function returns the true on success, or false otherwise.

Exceptions

None.

IONGraphicConnection Interface

The IONGraphicConnection interface defines the common functionality in the IONGrConnection and IONJGrConnection classes.

Interface Declaration

```
public interface IONGraphicConnection
```

Implementing Classes

- IONGrConnection / IONJGrConnection Class

Methods

- **addDrawable()** — Adds an IONGrDrawable class to this connection.
- **debugMode()** — Enables/Disables debug mode.
- **executeIDLCommand()** — Executes a given IDL command on the ION Server. Control is not returned until the command has been executed.
- **getIDLVariable()** — Gets the value of an IDL variable on the ION Server.
- **removeDrawable()** — Removes an IONGrDrawable class from this connection.
- **sendIDLCommand()** — Sends an IDL command to the ION Server. Control is returned as soon as the command has been sent.
- **setDrawable()** — Sets the current drawable.
- **setIDLVariable()** — Sets the value of an IDL variable on the ION Server.

See “IONGrConnection / IONJGrConnection Class” on page 173 for information on these methods.

IONGrConnection / IONJGrConnection Class

The IONGrConnection class represents a connection between the client and the ION Server. It allows for the addition of multiple IONGrGraphic classes and has the primary function of acting as a communication module between the IONGrGraphic classes and the ION Server.

Class Declaration

```
public class IONGrConnection
             extends IONGraphicsClient
             implements IONGraphicConnection,
                        IONCommandDoneListener,
                        IONOutputListener

public class IONJGrConnection
             extends IONGraphicsClient
             implements IONGraphicConnection,
                        IONCommandDoneListener,
                        IONOutputListener
```

Methods

- **IONGrConnection() —** Constructs an object of the IONGrConnection class.
- **addDrawable() —** Adds an IONGrDrawable class to this connection.
- **connect() —** Connects with an ION Server.
- **debugMode() —** Enables/Disables debug mode.
- **disconnect() —** Disconnects with an ION Server.
- **executeIDLCommand() —** Executes a given IDL command on the ION Server. Control is not returned until the command has been executed.
- **getIDLVariable() —** Gets the value of an IDL variable on the ION Server.
- **removeDrawable() —** Removes an IONGrDrawable class from this connection.

- **sendIDLCommand() —** Sends an IDL command to the ION Server. Control is returned as soon as the command has been sent.
- **setDrawable() —** Sets the current drawable.
- **setIDLVariable() —** Sets the value of an IDL variable on the ION Server.

IONGrConnection()

The IONGrConnection() method constructs an object of the IONGrConnection class.

Syntax

public IONGrConnection()

Arguments

None.

Example

```
IONGrConnection con = new IONGrConnection();
```

addDrawable()

The addDrawable() method adds the specified ION graphic to the connection object. In turn, the graphic objects sets its reference back to the connection. Once added, the graphic can communicate with the ION Server and thus request graphics and information.

Note

When using an IONGrConnection class, it is recommended that you add a drawable method using addDrawable(). Do not use the parent class IONGraphicsClient.addIONDrawable since this method does not set the connection from the drawable method back to the connection.

Syntax

public int addDrawable(IONGrDrawable *ionGraphic*)

public int addDrawable(IONJGrDrawable *ionGraphic*)

Return Value

This method returns a reference to the added IONGrDrawable/IONJGrDrawable object.

Arguments

ionGraphic

An object of the IONGrDrawable class to add to the connection object.

Exceptions

None.

Example

```
IONGrDrawable draw;
con.addDrawable(draw);
```

connect()

See “connect()” on page 165.

debugMode()

The debugMode() method enables and disables the debug mode of the class. When debug mode is enabled, the command log output from the ION Server is displayed in a window when a Shift-click (shifted mouse button-press) event is detected on the drawing surface.

When debug mode is enabled, the class will buffer the output information for all registered drawables sent by the ION Server to the client class.

Syntax

public void debugMode(boolean *bEnable*)

Arguments

bEnable

If true, the debug mode is enabled, otherwise the debug mode is disabled.

Exceptions

None.

Example

```
con.debugMode(true);
```

disconnect()

See “disconnect()” on page 127.

executeIDLCommand()

The executeIDLCommand() method sends an IDL command to the ION Server for execution. Any graphical output resulting from the IDL command is displayed in the IONGrDrawable drawing area. Control is not returned to the application until the command has been executed.

Syntax

public int executeIDLCommand(String *sIDLCommand*)

Return Value

The function returns 0 if the IDL command executed successfully, or the value of the IDL system variable !ERROR if the IDL command did not execute successfully.

Arguments

sIDLCommand

A string containing a valid IDL command.

Exceptions

IOException, IONIllegalCommandException, IONSecurityException

Example

```
try{
    con.executeIDLCommand("PLOT, FINDGEN(10)");
}catch(IOException e) {
    System.err.println("IO error:"te.getMessage());
```

```
}catch(IONIllegalCommandException eIC) {
    System.err.println("Illegal Command error:"te.getMessage());
}catch(IONSecurityException eSE) {
    System.err.println("Security error:"te.getMessage());
}
```

getIDLVariable()

See "getIDLVariable()" on page 129.

removeDrawable()

The removeDrawable() method removes a graphic from the connection object. Once the graphic has been removed from the connection object, the graphic can no longer communicate with the ION Server.

Syntax

public IONDrawable removeDrawable(IONGrDrawable *ionGraphic*)

public IONDrawable removeDrawable(IONJGrDrawable *ionGraphic*)

public IONDrawable removeDrawable(int *iGraphic*)

Return Value

This method returns a reference to the removed IONGrDrawable object.

Arguments

ionGraphic

An object of the IONGrDrawable class that is being removed from the connection.

iGraphic

A zero-based integer index designating which IONGrDrawable object to remove from the connection (the IDL window index).

Exceptions

None.

Example

```
IONGrDrawable draw = con.removeDrawable(iongraphic);
```

```
IONGrDrawable draw = con.removeDrawable(1);
```

sendIDLCommand()

The sendIDLCommand() method asynchronously sends an IDL command to the ION Server. Control is returned to the application as soon as the command has been sent.

Syntax

public void sendIDLCommand(String *sIDLCommand*) throws IO Exception

Arguments

sIDLCommand

A string containing a valid IDL command.

Exceptions

IOException

setDrawable()

The setDrawable() method designates which IONGRDrawable object will receive graphical output from the ION Server.

Syntax

public boolean setDrawable(IONGrDrawable *ionGraphic*)

public boolean setDrawable(IONJGrDrawable *ionGraphic*)

public boolean setDrawable(int *iGraphic*)

Return Value

This routine returns False if the specified drawable is not registered with the connection, or True otherwise.

Arguments

ionGraphic

An instance of an IONGrDrawable object to set as the current drawable. This graphic must have been registered with the IONGrConnection object via the addDrawable() method.

iGraphic

A zero-based integer index designating which IONGrDrawable object to set as the current drawable. This graphic must have been registered with the IONGrConnection object via the addDrawable() method.

Exceptions

None.

Example

```
boolean bSuccess = con.setDrawable(ionGraphic);
boolean bSuccess = con.setDrawable(1);
```

setIDLVariable()

See “setIDLVariable()” on page 133.

IONGrContour Class

The IONGrContour class produces an IDL-generated contour in a drawing area. The class allows the user to enter data and set contour attributes at the program level.

Class Declaration

```
public class IONGrContour
             extends IONGrGraphic
```

Methods

- **IONGrContour()** — Constructs an object of the IONGrContour class.
- **draw()** — Produces and displays the graphic on the drawing surface of this class.
- **getProperty()** — Gets the value of a property.
- **setProperty()** — Sets a property for the graphic.
- **setNoErase()** — Specifies whether the object should be erased when another object is drawn.
- **setXValue()** — Sets the X value of the contour.
- **setYValue()** — Sets the Y value of the contour.
- **setZValue()** — Sets the Z data of the contour.

IONGrContour()

The IONGrContour() method constructs an IONGrContour object.

Syntax

IONGrContour()

IONGrContour(int *Z[][]*)

IONGrContour(float *Z[][]*)

IONGrContour(double *Z[][]*)

IONGrContour(int *Z[]*)

IONGrContour(float *Z[]*)

IONGrContour(double *Z[]*)

IONGrContour(String *sName*)

IONGrContour(int *Z[]*, int *X[]*, int *Y[]*)

IONGrContour(int *Z[][]*, int *X[]*, int *Y[]*)

IONGrContour(float *Z[]*, float *X[]*, float *Y[]*)

IONGrContour(float *Z[][]*, float *X[]*, float *Y[]*)

IONGrContour(double *Z[]*, double *X[]*, double *Y[]*)

IONGrContour(double *Z[][]*, double *X[]*, double *Y[]*)

IONGrContour(String *sZName*, String *sXName*, String *sYName*)

Arguments

Z

The Z values (data) to use in the contour.

sName, sZName

The name of the IDL variable to use for the Z (data) values of the surface.

X

An array holding the values for the X coordinates of the grid.

Y

An array holding the values for the Y coordinates of the grid.

sXName

The name of the IDL variable holding the values for the X coordinates of the grid.

sYName

The name of the IDL variable holding the values for the Y coordinates of the grid.

Exceptions

None.

draw()

Call the draw() method to produce and display a graphic in the drawing area that makes up this object.

Syntax

public void draw(IONGrConnection *grConn*)

Arguments

grConn

IONGrConnection used to issue the drawing commands to the server.

Exceptions

None.

getProperty()

The getProperty() method retrieves the current value of a property.

Syntax

public IONVariable getProperty(String *Property*)

Arguments

Property

The name of the property.

Return Value

The function returns the current value of a property.

Properties Supported

The following IDL Contour properties are supported by IONGrContour.[get,set]Property. Refer to the IDL documentation on keywords available for use with the CONTOUR procedure for an explanation of each property:

BACKGROUND, CELL_FILL, CHARSIZE, CLIP, CLOSED, COLOR, C_ANNOTATION, C_CHARSIZE, C_COLORS, C_LABELS, C_LINESTYLE, C_ORIENTATION, C_SPACING, DATA, DEVICE, DOWNHILL, FILL, FOLLOW, FONT, IRREGULAR, LEVELS, MAX_VALUE, MIN_VALUE, NLEVELS, NOCLIP, NODATA, NOERASE, NORMAL, OVERPLOT, PATH_DATA_COORDS, PATH_DOUBLE, PATH_FILENAME, PATH_INFO, PATH_XY, POLAR, POSITION, SUBTITLE, T3D, TICKLEN, TITLE, TRIANGULATION, XCHARSIZE/YCHARSIZE/ZCHARSIZE, XGRIDSTYLE/YGRIDSTYLE/ZGRIDSTYLE, XLOG, YLOG, XMARGIN/YMARGIN/ZMARGIN, XMINOR/YMINOR/ZMINOR, XRANGE/YRANGE/ZRANGE, XSTYLE/YSTYLE/ZSTYLE, XTICKFORMAT/YTICKFORMAT/ZTICKFORMAT, XTICKLEN/YTICKLEN/ZTICKLEN, XTICKNAME/YTICKNAME/ZTICKNAME, XTICKS/YTICKS/ZTICKS, XTICKV/YTICKV/ZTICKV, XTITLE/YTITLE/ZTITLE, XTICKINTERVAL/YTICKINTERVAL/ZTICKINTERVAL, XTICKLAYOUT/YTICKLAYOUT/ZTICKLAYOUT, XTICKUNITS/YTICKUNITS/ZTICKUNITS, XTICK_GET/YTICK_GET/ZTICK_GET, ZAXIS, ZLOG, ZVALUE

Exceptions

None.

Example

```
IONVariable value = getProperty(Property);
```

setProperty()

The setProperty() method sets a property for the contour object.

Syntax

public void setProperty(String *Property*, IONVariable *Value*)

Arguments

Property

The name of the property to set.

Value

The value of the property.

Properties Supported

The IDL Contour properties are supported by IONGrContour.[get,set]Property are the same as those covered in “getProperty()” on page 182. Refer to the IDL documentation on keywords available for use with the CONTOUR procedure for an explanation of each property.

Exceptions

None.

setNoErase()

The setNoErase() method of the IONGrContour class overrides setNoErase() in the IONGrGraphic class. See “setNoErase()” on page 201 for the description and syntax of this method.

setXValue()

The setXValue() method resets the X value of the contour.

Syntax

public void setXValue(int *X[]*)

public void setXValue(float *X[]*)

public void setXValue(double *X[]*)

public void setXValue(String *sName*)

Arguments

X

The new X value of the contour.

sName

The name of the IDL variable that contains the new X value of the surface.

Exceptions

None.

setYValue()

The setYValue() method resets the Y value of the contour.

Syntax

public void setYValue(int *Y[]*)

public void setYValue(float *Y[]*)

public void setYValue(double *Y[]*)

public void setYValue(String *sName*)

Arguments

Y

The new Y value of the contour.

sName

The name of the IDL variable that contains the new Y value of the contour.

Exceptions

None.

setZValue()

The setZValue() method resets the Z value of the contour.

Syntax

public void setZValue(int *Z[]*)

public void setZValue(float *Z[]*)

public void setZValue(double *Z[]*)

public void setZValue(int *Z[][]*)

public void setZValue(float *Z[][]*)

public void setZValue(double *Z[][]*)

public void setZValue(String *sName*)

Arguments

Z

The new Z value of the contour.

sName

The name of the IDL variable that contains the new Z value of the contour.

Exceptions

None.

IONGrDrawable / IONJGrDrawable Class

Objects of the IONGrDrawable class represent a drawing area for IDL-produced graphics that can be part of a Java AWT. The IONGrDrawable can act alone as a drawing area or it can contain many IONGrGraphic objects. The way in which multiple Graphic objects are displayed in the drawable can be controlled using setNoErase() and setMulti().

Class Declaration

```
public class IONGrDrawable
             extends IONCanvas
             implements java.awt.event.MousesListener

public class IONJGrDrawable
             extends IONJCanvas
             implements java.awt.event.MousesListener
```

Methods

- **IONGrDrawable() / IONJGrDrawable()** — Constructs an object of the IONGrDrawable class.
- **addGraphic()** — Adds a graphic object to be drawn.
- **debugMode()** — Enables/Disables the debug mode of the class.
- **draw()** — Draws all graphic objects in the drawable.
- **executeIDLCommand()** — Executes an IDL command on the ION Server.
- **getConnection()** — Gets the connection object associated with this drawable.
- **isConnected()** — Returns true if the drawable is associated with a connection.
- **removeGraphic()** — Removes a graphic object from the drawable.
- **resetMulti()** — Resets “multi mode” to one visible drawable at a time.
- **sendIDLCommand()** — Sends an IDL command to the ION Server.
- **setConnection()** — Associates this IONGrDrawable object with an IONGraphicConnection.

- **setMulti() —** Specifies how multiple graphic objects will be drawn in the server.
- **setNoErase() —** Specifies whether the drawable should be erased when new graphic is drawn.

Subclasses

IONContour / IONJContour Class, IONMap / IONJMap Class, IONPlot / IONJPlot Class, IONSurface / IONJSurface Class

IONGrDrawable() / IONJGrDrawable()

The IONGrDrawable() method constructs an IONGrDrawable object of a specified size.

Syntax

public IONGrDrawable(int *iWidth*, int *iHeight*)

public IONJGrDrawable(int *iWidth*, int *iHeight*)

Arguments

iWidth

The width of the drawing area.

iHeight

The height of the drawing area.

Exceptions

None.

addGraphic()

The addGraphic() method adds an IONGrGraphic to the drawable. Calling the draw() method causes all the graphics added in this manner to be displayed in the drawing area.

Syntax

public void addGraphic(IONGrGraphic *ionGraphic*)

Arguments

ionGraphic

Graphic object to add.

Exceptions

None.

Example

```
addGraphic(ionGraphic);
```

debugMode()

The debugMode() method enables and disables the debug mode of the class. When debug mode is enabled, the command log output from the ION Server is displayed in a window when a Shift-click (shifted mouse button-press) event is detected on the drawing surface.

When debug mode is enabled, the class will buffer the output information for all registered drawables sent by the ION Server to the client class.

Syntax

public void debugMode(boolean *bEnable*)

Arguments

bEnable

If true, the debug mode is enabled, otherwise the debug mode is disabled.

Example

```
con.debugMode(true);
```

draw()

The draw() method draws all the graphics objects associated with this drawable. If there are no graphics objects associated, nothing happens.

Syntax

public void draw()

Arguments

None.

Exceptions

None.

executeIDLCommand()

The executeIDLCommand() method sends an IDL command to the ION Server for execution. The call returns when the command has finished executing. Any resultant graphics is displayed in the IONGraphic drawing area.

Syntax

public int executeIDLCommand(String *sIDLCommand*)

Return Value

The function returns 0 if the IDL command executed successfully, or the value of the IDL system variable !ERROR if the IDL command did not execute successfully.

Arguments

sIDLCommand

A string containing a valid IDL command.

Exceptions

IOException, IONIllegalCommandException, IONSecurityException

getConnection()

The getConnection() method is used to retrieve the IONGraphicConnection object with which this object is associated.

Syntax

public IONGraphicConnection getConnection()

Return Value

Returns the IONGraphicConnection object that this object is associated with. If no connection is associated with this object null is returned.

Arguments

None.

Exceptions

None.

Example

```
IONGraphicConnection conn = getConnection();
```

isConnected()

The isConnected() method is used to determine whether the drawable is associated with an IONGraphicConnection.

Syntax

public boolean isConnected()

Return Value

The method returns true if the drawable is associated with an IONGraphicConnection, and false otherwise.

Arguments

None.

Exceptions

None.

Example

```
boolean connected = isConnected();
```

removeGraphic()

The removeGraphic() method removes an IONGrGraphic from the drawable.

Syntax

public boolean removeGraphic(IONGrGraphic *ionGraphic*)

Return Value

The method returns true on success or false if the specified graphic is not currently part of the system.

Arguments

ionGraphic

A graphic to remove from the drawable.

Exceptions

None.

Example

```
removeGraphic( ionGraphic);
```

resetMulti()

The resetMulti() method resets the !P.multi system variable to 0 (one plot at a time, using the entire drawing area).

Syntax

public void resetMulti()

Arguments

None.

Exceptions

None.

sendIDLCommand()

The sendIDLCommand() method asynchronously sends an IDL command to the ION Server for execution. The IDL command is posted to the server for execution and the function immediately returns. Notification of the commands completion is performed via the IONCommandDoneListener interface. Control is returned to the application as soon as the command has been sent.

Syntax

public void sendIDLCommand(String *sIDLCommand*)

Arguments

sIDLCommand

The IDL Command that is to be executed on the ION Server. The use of the spawn command and the line continuation character ($) is prohibited (for security reasons, and because they can hang the server).

Exceptions

IOException

setConnection()

The setConnection() method associates this IONGrDrawable object with an IONGraphicConnection.

Syntax

public void setConnection(IONGraphicConnection *ionConnection*)

Arguments

ionConnection

The connection with which to associate this IONGrDrawable object.

Exceptions

None.

setMulti()

The setMulti() method sets the !P.multi system variable that determines how multiple IDL plots or IONGrGraphics objects are displayed on the drawing area.

Syntax

public void setMulti(int *iMulti[]*)

Arguments

iMulti

Array defining the layout. See the IDL documentation for more information.

Exceptions

None.

setNoErase()

The setNoErase() method specifies whether or not the drawable should be erased between IONGrGraphic objects when the draw() method is called.

Syntax

public void setNoErase(boolean *bNoErase*)

Arguments

bNoErase

If true, the drawing area should not be erased.

Exceptions

None.

IONGrGraphic Class

The IONGrGraphic abstract class implements methods that are used by sub-classes to manage and store properties.

Class Declaration

```
public abstract class IONGrGraphic
                      extends Object
```

Methods

- **IONGrGraphic() —** Constructs an object of the IONGrGraphic class.
- **draw() —** Draws the object.
- **getProperty() —** Gets the value of the given property.
- **getPropertyNames()** — Gets the names of the properties.
- **getPropertyString() —** Gets a string that represents the properties. This string can then used with an IDL command.
- **setNoErase() —** Specifies whether the object should be erased when another object is drawn.
- **setProperty() —** Sets the value of a property in the property list.

Subclasses

IONGrContour Class, IONGrMap Class, IONGrMapContinents Class, IONGrMapGrid Class, IONGrMapImage Class, IONGrPlot Class, IONGrSurface Class

IONGrGraphic()

The IONGrGraphic() method constructs an object of the IONGrGraphic class.

Syntax

public IONGrGraphic()

Arguments

None.

Exceptions

None.

draw()

The draw() method is defined by sub-classes to issue the appropriate IDL command to draw the graphic object.

Syntax

public void draw(IONGraphicConnection *con*)

Arguments

con

IONGraphicConnection used to issue the drawing commands to the server.

Exceptions

None.

Example

```
draw(con);
```

getProperty()

The getProperty() method returns the value of a property. The property is returned as an object. It is the responsibility of the caller to cast the object to the correct type.

Syntax

public IONVariable getProperty(String *sName*)

Arguments

sName

The name of the property.

Exceptions

None.

Example

```
IONVariable = getProperty(sProperty);
```

getPropertyNames()

The getPropertyNames() method returns a string that contains the names of all the properties contained in the object. The string is formatted such that each property name makes up an IDL keyword. This string can be appended to an IDL graphics command string

Syntax

public final Enumeration getPropertyNames()

Arguments

None.

Exceptions

None.

getPropertyString()

The getPropertyString() method returns a string that contains the values of all the properties contained in the object. The string is formatted such that each property name makes up an IDL keyword and the value of the property is the value of the keyword. This string can be appended to an IDL graphics command string.

Note

This is a protected method, and can only be accessed from objects that subclass the IONGrGraphic class.

Syntax

protected final String getPropertyString()

Arguments

None.

Exceptions

None.

Example

```
String sProperties = getPropertyString();
```

registerProperty()

The registerProperty() method is used to register a property name as being valid. When the setProperty() or getProperty() methods are called, they check the validity of the object against the list of valid properties.

Note

This is a protected method, and can only be accessed from objects that subclass the IONGrGraphic class.

Syntax

registerProperty(String *PropertyName*)

Arguments

PropertyName

The name of the property.

Exceptions

None.

Example

```
protected registerProperty(PropertyName);
```

setNoErase()

The setNoErase() method is defined by subclasses to set the appropriate property for the graphic object that corresponds to the concept of 'no erase'.

Syntax

public void setNoErase(boolean *bFlag*)

Arguments

bFlag

If true, the object is not erased when other objects are drawn.

Exceptions

None.

Example

```
setNoErase(bFlag);
```

setProperty()

The setProperty() method is used to set the value of a property in the objects property list. If the property already exists in the property list, its value is replaced, otherwise the property is added to the property list.

Syntax

public void setProperty(String *sName*, IONVariable *vValue*)

Arguments

sName

The name of the property to set.

vValue

The value of the property. This must be an object or an array.

Exceptions

None.

Example

```
void setProperty(sProperty, value);
```

IONGrMap Class

IONGrMap is an IONGrGraphic that encapsulates the functionality of IDL's MAP_SET procedure. IONGrMap is used to set up a drawing area (IONGrDrawable) to display data on a map projection.

All MAP_SET keywords are accepted except GOODESHOMOLOSINE, ROBINSON, MILLER_CYLINDRICAL, NAME and REVERSE.

Class Declaration

```
public class IONGrMap
             extends IONGrGraphic
```

Methods

- **IONGrMap() —** Constructs a new map centered at (*lat*, *lon*) with rotation *rot.*
- **draw() —** Calls the MAP_SET procedure to draw the map projection.
- **getProperty()** — Retrieves the specified property.
- **setLat(), setLon() —** Sets the lat/lon on which to center the projection.
- **setProperty()** — Sets the value of the specified property.
- **setRotation() —** Sets the rotation of the map projection.

IONGrMap()

Constructs a new map centered at (*lat*, *lon*) with rotation *rot*.

Syntax

public IONGrMap()

public IONGrMap(int *lat*)

public IONGrMap(float *lat*)

public IONGrMap(double *lat*)

public IONGrMap(int *lat*, int *lon)*

public IONGrMap(float *lat*, float *lon*)

public IONGrMap(double *lat*, double *lon*)

public IONGrMap(int *lat*, int *lon*, int *rot)*

public IONGrMap(float *lat*, float *lon*, float *rot*)

public IONGrMap(double *lat*, double *lon*, double *rot*)

public IONGrMap(String *sLat*)

public IONGrMap(String *sLat*, String *sLon*)

public IONGrMap(String *sLat*, String *sLon*, String *sRot*)

Arguments

lat

The latitude of the point on the Earth's surface to be mapped to the center of the projection plane. Latitude is measured in degrees north of the equator, and *lat* must be in the range -90° ≤ *lat* ≤ 90°. The default is 0.

lon

The longitude of the point on the Earth's surface to be mapped to the center of the projection plane. Longitude is measured in degrees east of the Greenwich meridian, and *lon* must be in the range -180° ≤ *lon* ≤ 180°. The default is 0.

rot

The angle through which the North direction should be rotated around the line L between the Earth's center and the point (*lat, lon*). This angle is measured in degrees with the positive direction being clockwise around the line L, and must be in the range -180° ≤ *rot* ≤ 180°. The default is 0.

If the center of the map is at the North pole, North is in the direction *lon* + 180°. If the origin is at the South pole, North is in the direction *lon*.

sLat, sLon, sRot

Strings representing the latitude, longitude, and rotation.

Exceptions

None.

draw()

The draw() method calls the IDL MAP_SET procedure to draw the map projection.

Syntax

public void draw(IONGraphicConnection *grConn*)

Arguments

grConn

The name of the connection.

Exceptions

None.

getProperty()

The getProperty() method retrieves the value of the specified property.

Syntax

public final IONVariable getProperty(Sting *sName*)

Arguments

sName

The name of the property to retrieve.

Properties Supported

The following IDL map properties are supported by IONGrMap.[get,set]Property. Refer to the IDL documentation on keywords available for use with the MAP_SET procedure for an explanation of each property:

Projection Types: AITOFF, ALBERS, AZIMUTHAL, CONIC, CYLINDRICAL, GNOMIC, GOODESHOMOLOSINE, HAMMER, LAMBERT, MERCATOR, MILLER_CYLINDRICAL, MOLLEWIDE, ORTHOGRAPHIC, ROBINSON, SATELLITE, SINUSOIDAL, STEREOGRAPHIC, TRANSVERSE_MERCATOR

Map Characteristics: ADVANCE, CHARSIZE, CLIP, COLOR, CONTINENTS, CON_COLOR, HIRES, E_CONTINENTS, E_GRID, E_HORIZON, GLINESTYLE, GRID, HORIZON, LABEL, LATALIGN, LATDEL, LATLAB, LONDEL, LONLAB, MLINESTYLE, NAME, NOBORDER, NOERASE, REVERSE, TITLE, USA, XMARGIN, YMARGIN

Projection Parameters: CENTRAL_AZIMUTH, ELLIPSOID, ISOTROPIC, LIMIT, SAT_P, SCALE, STANDARD_PARALLELS

Graphics: POSITION, T3D, ZVALUE

Exceptions

None.

setLat(), setLon()

Sets the lat/lon on which to center the projection.

Syntax

public void setLat(int *lat*)

public void setLat(float *lat*)

public void setLat(double *lat*)

public void setLat(String *lat*)

public void setLon(int *lon*)

public void setLon(float *lon*)

public void setLon(double *lon*)

public void setLon(String *lon*)

Arguments

lat

The latitude of the point on the Earth's surface to be mapped to the center of the projection plane. Latitude is measured in degrees north of the equator, and *lat* must be in the range -90° ≤ *lat* ≤ 90°. The default is 0.

lon

The longitude of the point on the Earth's surface to be mapped to the center of the projection plane. Longitude is measured in degrees east of the Greenwich meridian, and *lon* must be in the range -180° ≤ *lon* ≤ 180°. The default is 0.

Exceptions

None.

setProperty()

The setProperty() method sets the specified property to the specified value.

Syntax

public final void setProperty(string *sName*, IONVariable *vValue*)

Arguments

sName

The name of the property to set.

vValue

The value of the property to set.

Properties Supported

The IDL Map properties are supported by IONGrMap.[get,set]Property are the same as those covered in "getProperty()" on page 205. Refer to the IDL documentation on keywords available for use with the MAP_SET procedure for an explanation of each property:

Exceptions

None.

setRotation()

Sets the rotation of the map projection.

Syntax

public void setRotation(int *rot*)

public void setRotation(float *rot*)

public void setRotation(double *rot*)

public void setRotation(String *rot*)

Arguments

rot

The angle through which the North direction should be rotated around the line L between the Earth's center and the point (*lat*, *lon*). This angle is measured in degrees with the positive direction being clockwise around the line L, and must be in the range -180° ≤ *rot* ≤ 180°. The default is 0.

If the center of the map is at the North pole, North is in the direction *lon* + 180°. If the origin is at the South pole, North is in the direction *lon*.

Exceptions

None.

IONGrMapContinents Class

An IONGrMapContinents object is an IONGrGraphic that encapsulates the functionality of IDL's MAP_CONTINENTS procedure, which is used to draw continental boundaries, filled continents, political boundaries, coastlines, and rivers over an existing map projection. This is used in conjunction with an IONGrMap.

All IDL MAP_CONTINENTS keywords are accepted except CONTINENTS, LIMIT, T3D, AND ZVALUE.

Class Declaration

```
public class IONGrMapContinents
             extends IONGrGraphic
```

Methods

- **IONGrMapContinents() —** Constructs an object of the IONGrMapContinents class.
- **draw() —** Calls the MAP_CONTINENTS procedure to add boundaries to the current map projection.
- **getProperty() —** Retrieves the specified property.
- **setProperty() —** Sets the value of the specified property.

IONGrMapContinents()

The IONGrMapContinents() method constructs an object of the IONGrMapContinents class.

Syntax

public IONGrMapContinents()

Arguments

None.

Exceptions

None.

draw()

The draw() method calls the IDL MAP_CONTINENTS procedure to add boundaries to the current map projection.

Syntax

public void draw(IONGraphicConnection *grConn*)

Arguments

grConn

The name of the connection.

Exceptions

None.

getProperty()

The getProperty() method retrieves the specified value of the property.

Syntax

public final IONVariable getProperty(Sting *sName*)

Arguments

sName

The name of the property to retrieve.

Properties Supported

The following IDL MAP_CONTINENTS properties are supported by the IONGrMapContinents.[get,set]Property. Refer to the IDL documentation on keywords available for use with the MAP_CONTINENTS procedure for an explanation of each property:

COASTS, COLOR, CONTINENTS, COUNTRIES, FILL_CONTINENTS, HIRES, LIMIT, MLINESTYLE, ORIENTATION, RIVERS, SPACING, T3D, USA, ZVALUE

Exceptions

None.

Example

setProperty()

The setProperty() method set the specified property to the specified value.

Syntax

public final void setProperty(string *sName*, IONVariable *vValue*)

Arguments

sName

The name of the property to set.

vValue

The value to which to set the property.

Properties Supported

The properties supported by IONGrMapContinents.[get,set]Property are the same as those supported by "getProperty()" on page 210. Refer to the IDL documentation on keywords available for use with the MAP_CONTINENTS procedure for an explanation of each property.

Exceptions

None.

IONGrMapGrid Class

An IONGrMapGrid object is an IONGrGraphic that encapsulates the functionality of IDL's MAP_GRID procedure, which is used to draw lat/lon lines on a map projection. This is used in conjunction with an IONGrMap.

Class Declaration

```
public class IONGrMapGrid
             extends IONGrGraphic
```

Methods

- **IONGrMapGrid() —** Constructs an object of the IONGrMapGrid class.
- **draw() —** Calls the MAP_GRID procedure to add a lat/lon grid to the current map projection.
- **getProperty() —** Retrieves the specified property.
- **setProperty() —** Sets the value of the specified property.

IONGrMapGrid()

The IONGrMapGrid() method constructs an object of the IONGrMapGrid class.

Syntax

public IONGrMapGrid()

Arguments

None.

Exceptions

None.

draw()

The draw() method calls the MAP_GRID procedure to add a lat/lon grid to the current map projection.

Syntax

public void draw(IONGraphicConnection *grConn*)

Arguments

grConn

The name of the connection.

Exceptions

None.

getProperty()

The getProperty() method retrieves the value of the specified property.

Syntax

public final IONVariable getProperty(Sting *sName*)

Argument

sName

The name of the property to retrieve.

Properties Supported

The following IDL MAP_GRID properties are supported by IONGrMapGrid.[get,set]Property. Refer to the IDL documentation on keywords available for use with the MAP_GRID procedure for an explanation of each property:

BOX_AXES, CHARSIZE, CLIP_TEXT, COLOR, FILL_HORIZON, GLINESTYLE, GLINETHICK, HORIZON, INCREMENT, LABEL, LATALIGN, LATDEL, LATLAB, LATNAMES, LATS, LONALIGN, LONDEL, LONLAB, LONNAMES, LONS, NO_GRID, ORIENTATION, T3D, ZVALUE

Exceptions

None.

setProperty()

The setProperty() method sets the specified property to the specified value.

Syntax

public final void setProperty(string *sName*, IONVariable *vValue*)

Arguments

sName

The name of the property to set.

vValue

The value to which to set the property.

Properties Supported

The IDL MAP_GRID properties supported by the IONGrMapGrid.[get,set]Property are the same as those supported by "getProperty()" on page 213. Refer to the IDL documentation on keywords available for use with the MAP_GRID procedure for an explanation of each property.

Exceptions

None.

IONGrMapImage Class

IONGrMapImage is an IONGrGraphic that encapsulates the functionality of IDL's MAP_IMAGE procedure, which projects an image onto a map projection. This is used in conjunction with an IONGrMap.

All IDL 5.4 MAP_IMAGE keywords are accepted.

Class Declaration

```
public class IONGrMapImage
             extends IONGrGraphic
```

Methods

- **IONGrMapImage() —** Constructs an object of the IONGrMapImage class.
- **draw() —** Calls the MAP_IMAGE procedure to project an image onto the current map projection.
- **getProperty()** — Retrieves the specified property.
- **setImage() —** Sets the image that will be projected. It can either be in the form of a two dimensional array or the name of the IDL variable.
- **setProperty()** — Sets the value of the specified property.
- **setStart() —** Defines the coordinates of the lower left corner of the image.

IONGrMapImage()

The IONGrMapImage() method constructs an object of the IONGrMapImage class.

Syntax

public IONGrMapImage()

public IONGrMapImage(byte *image[][]*)

public IONGrMapImage(int *image[][]*)

public IONGrMapImage(float *image[][]*)

public IONGrMapImage(double *image[][]*)

public IONGrMapImage(String *image*)

Arguments

image

A two-dimensional array containing the image to be overlaid on the map, or a variable containing an array.

Exceptions

None.

draw()

The draw() method calls the MAP_IMAGE procedure to project an image onto the current map projection.

Syntax

public void draw(IONGraphicConnection *grConn*)

Arguments

grConn

The name of the connection.

Exceptions

None.

getProperty()

The getProperty() method retrieves the specified value of the property.

Syntax

public final IONVariable getProperty(Sting *sName*)

Arguments

sName

The name of the property to retrieve.

Properties Supported

The following IDL MAP_IMAGE properties are supported by the IONGrMapImage.[get,set]Property. Refer to the IDL documentation on keywords available for use with the MAP_IMAGE procedure for an explanation of each property:

BILINEAR, COMPRESS, LATMAX, LATMIN, LONMAX, LONMIN, MAX_VALUE, MIN_VALUE, MISSING, SCALE

Exceptions

None.

setImage()

The setImage() method defines the image that will be projected. It can either be in the form of a two dimensional array or the name of the IDL variable.

Syntax

public void setImage(byte *image[][]*)

public void setImage(int *image[][]*)

public void setImage(float *image[][]*)

public void setImage(double *image[][]*)

public void setImage(String *image*)

Arguments

image

A two-dimensional array containing the image to be overlaid on the map, or a variable containing an array.

Exceptions

None.

setProperty()

The setProperty() method sets the specified property to the specified value.

Syntax

public final void setProperty(string *sName*, IONVariable *vValue*)

Arguments

sName

The name of the property to set.

vValue

The value of the property.

Properties Supported

The IDL map properties are supported by IONGrMapImage.[get,set]Property are the same as those covered in "getProperty()" on page 216. Refer to the IDL documentation on keywords available for use with the MAP_IMAGE procedure for an explanation of each property.

Exceptions

None.

setStart()

The setStart() method defines the coordinates of the lower left corner of the image.

Syntax

public void setStart(int *x*, int *y*)

public void setStart(float *x*, float *y*)

public void setStart(double *x*, double *y*)

Arguments

x

The x coordinate position of the left edge of the image.

y

The y coordinate position of the left edge of the image.

Exceptions

None.

IONGrPlot Class

The `IONGrPlot` class produces an IDL generated plot in a drawing area. The class allows the user to enter data and plot attributes at the program level.

Class Declaration

```
public class IONGrPlot
             extends IONGrGraphic
```

Methods

- **IONGrPlot() —** Constructs an object of the `IONPlot` class.
- **draw() —** Produces the output graphic and displays the graphic on the drawing surface of this class.
- **getProperty() —** Gets the value of a property.
- **setNoErase() —** Specifies whether the object should be erased when another object is drawn.
- **setProperty() —** Sets a property for the graphic.
- **setXValue() —** Sets the X value of the plot.
- **setYValue() —** Sets the Y value of the plot.

IONGrPlot()

The IONGrPlot() method constructs an object of the IONGrPlot class.

Syntax

public IONGrPlot()

public IONGrPlot(int *X[]*)

public IONGrPlot(float *X[]*)

public IONGrPlot(double *X[]*)

public IONGrPlot(String *sName*)

public IONGrPlot(int *X[]*, int *Y[]*)

public IONGrPlot(float *X[]*, float *Y[]*)

public IONGrPlot(double *X[]*, double *Y[]*)

public IONGrPlot(String *sXName*, String *sYName*)

Arguments

X

X values of the plot.

Y

Y values of the plot.

sXName

The name of an IDL variable to use for the X values in this plot.

sYName

The name of an IDL variable to use for the Y values in this plot.

Exceptions

None.

draw()

The draw() method displays the plot in the drawing area that makes up this object.

Syntax

public void draw(IONGraphicConnection *grConn*)

Arguments

grConn

IONGraphicConnection used to issue the drawing commands to the server.

Exceptions

None.

Example

```
draw(con);
```

getProperty()

The getProperty() method retrieves the current value of the specified property.

Syntax

```
public IONVariable getProperty(String Property)
```

Arguments

Property

The name of the property.

Return Value

The function returns the current value of a property.

Properties Supported

The following IDL Plot properties are supported by IONGrPlot.[get,set]Property. Refer to the IDL documentation on keywords available for use with the PLOT procedure for an explanation of each property:

BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, ISOTROPIC, LINESTYLE, MAX_VALUE, MIN_VALUE, NOCLIP, NODATA, NOERASE, NORMAL, NSUM, POLAR, POSITION, PSYM, SUBTITLE, SYMSIZE, T3D, TICKLEN, TITLE, XCHARSIZE/YCHARSIZE/ZCHARSIZE, XGRIDSTYLE/YGRIDSTYLE/ZGRIDSTYLE, XLOG, XMARGIN/YMARGIN/ZMARGIN, XMINOR/YMINOR/ZMINOR, XRANGE/YRANGE/ZRANGE, XSTYLE/YSTYLE/ZSTYLE, XTICKFORMAT/YTICKFORMAT/ZTICKFORMAT, XTICKINTERVAL/YTICKINTERVAL/ZTICKINTERVAL, XTICKLAYOUT/YTICKLAYOUT/ZTICKLAYOUT, XTICKLEN/YTICKLEN/ZTICKLEN, XTICKNAME/YTICKNAME/ZTICKNAME, XTICKS/YTICKS/ZTICKS, XTICKUNITS/YTICKUNITS/ZTICKUNITS, XTICKV/YTICKV/ZTICKV, XTICK_GET/YTICK_GET/ZTICK_GET, XTITLE/YTITLE/ZTITLE, YLOG, YNOZERO, ZVALUE

Exceptions

None.

Examples

```
IONVariable value = getProperty(Property);
```

setNoErase()

The setNoErase() method of the IONGrPlot class overrides setNoErase() in the IONGrGraphic class. See “setNoErase()” on page 201 for the syntax of this method.

setProperty()

The setProperty() method sets a property for the plot object.

Syntax

public void setProperty(String *Property*, IONVariable *Value*)

Arguments

Property

The name of the property to set.

Value

The value of the property.

Properties Supported

The properties supported by the IONGrPlot.[get,set]Property are the same as those supported by the “getProperty()” on page 222. Refer to the IDL documentation on keywords available for use with the PLOT procedure for an explanation of each property.

Exceptions

None.

Example

```
setProperty(Property, Value);
```

setXValue()

The setXValue() method resets the X value of the plot

Syntax

public void setXValue(int *X[]*)

public void setXValue(float *X[]*)

public void setXValue(double *X[]*)

public void setXValue(String *sXname*)

Arguments

X

The new X value of the plot.

sXname

The name of an IDL variable to use for the X value.

Exceptions

None.

setYValue()

The setYValue() method resets the Y value of the plot.

Syntax

public void setYValue(int *Y[]*)

public void setYValue(float *Y[]*)

public void setYValue(double *Y[]*)

public void setYValue(String *sYname*)

Arguments

Y

The new Y value of the plot.

sYname

The name of the IDL variable to use for the Y value.

Exceptions

None.

IONGrSurface Class

The IONGrSurface class produces an IDL-generated surface using SHADE_SURF or SURFACE in a drawing area. The class allows the user to enter data and set surface attributes at the program level.

Class Declaration

```
public class IONGrSurface
             extends IONGrGraphic
```

Methods

- **IONGrSurface() —** Constructs an object of the IONGrSurface class.
- **draw() —** Produces the output graphic and displays the graphic on the drawing surface of this class.
- **getProperty() —** Gets the value of a property.
- **setNoErase() —** Specifies whether the object should be erased when another object is drawn.
- **setProperty() —** Sets a property for the graphic.
- **setXValue() —** Sets the X value of the surface.
- **setYValue() —** Sets the Y value of the surface.
- **setZValue() —** Sets the Z data of the surface.

IONGrSurface()

The IONGrSurface() method constructs an object of the IONGrSurface class.

Syntax

public IONGrSurface()

public IONGrSurface(int *Z[][]*)

public IONGrSurface(float *Z[][]*)

public IONGrSurface(double *Z[][]*)

public IONGrSurface(int *Z[][]*, int *X[]*, int *Y[]*)

public IONGrSurface(float *Z[][]*, float *X[]*, float *Y[]*)

public IONGrSurface(double *Z[][]*, double *X[]*, double *Y[]*)

public IONGrSurface(String *sZname*)

public IONGrSurface(String *sZname*, String *sXname*, String *sYname*)

Arguments

Z

Z (data) values for the surface

sName, sZName

Name of the IDL variable to use for the Z (data) of the surface.

X

Array holding the values for the X coordinates of grid.

Y

Array holding the values for the Y coordinates of grid.

sXName

Name of the IDL variable holding the values for X coordinates of the grid.

sYName

Name of the IDL variable holding the values for Y coordinates of the grid.

Exceptions

None.

draw()

The draw() method displays the surface in the drawing area that makes up this object.

Syntax

public void draw(IONGraphicConnection *grConn*)

Arguments

grConn

IONGraphicConnection used to issue the drawing commands to the server.

Exceptions

None.

getProperty()

The getProperty() method retrieves the current value of the specified property.

Syntax

public IONVariable getProperty(String *sName*)

Arguments

sName

The name of the property.

Return Value

The function returns the current value of a property.

Properties Supported

The following IDL Surface properties are supported by IONGrSurface.[get,set]Property. Refer to the IDL documentation on keywords available for use with the SURFACE procedure for an explanation of each property:

AX, AZ, BACKGROUND, BOTTOM, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, HORIZONTAL, IMAGE, LEGO, LINESTYLE, LOWER_ONLY, MAX_VALUE, MIN_VALUE, NOCLIP, NODATA, NOERASE, NORMAL, PIXELS, POSITION, SAVE, SHADES, SKIRT, SUBTITLE, T3D, TICKLEN, TITLE, UPPER_ONLY, XLOG/YLOG/ZLOG, XCHARSIZE/YCHARSIZE/ZCHARSIZE, XGRIDSTYLE/YGRIDSTYLE/ZGRIDSTYLE, XMARGIN/YMARGIN/ZMARGIN, XMINOR/YMINOR/ZMINOR, XRANGE/YRANGE/ZRANGE, XSTYLE/YSTYLE/ZSTYLE, XTICKFORMAT/YTICKFORMAT/ZTICKFORMAT,

XTICKINTERVAL/YTICKINTERVAL/ZTICKINTERVAL, XTICKLAYOUT/YTICKLAYOUT/ZTICKLAYOUT, XTICKLEN/YTICKLEN/ZTICKLEN, XTICKNAME/YTICKNAME/ZTICKNAME, XTICKS/YTICKS/ZTICKS, XTICKUNITS/YTICKUNITS/ZTICKUNITS, XTICKV/YTICKV/ZTICKV, XTICK_GET/YTICK_GET/ZTICK_GET, XTITLE/YTITLE/ZTITLE, ZAXIS, ZVALUE

Exceptions

None

Example

```
IONVariable value = getProperty(Property);
```

setNoErase()

The setNoErase() method of the IONGrSurface class overrides setNoErase() in the IONGrGraphic class. See “setNoErase()” on page 201 for the syntax of this method.

setProperty()

The setProperty() method sets a property for the surface object.

Syntax

public void setProperty(String *sName*, IONVariable *v*)

Arguments

sName

The name of the property to set.

v

The value of the property.

Properties Supported

The properties supported by the IONGrSurface.[get,set]Property as the same as those supported by the “getProperty()” on page 228 Refer to the IDL documentation on

keywords available for use with the SURFACE procedure for an explanation of each property.

Exceptions

None.

setXValue()

The setXValue() method resets the X value of the surface.

Syntax

public void setXValue(int *X[]*)

public void setXValue(float *X[]*)

public void setXValue(double *X[]*)

public void setXValue(String *sName*)

Arguments

X

The new X value of the surface.

sName

The name of the IDL variable that contains the new X value of the surface.

Exceptions

None.

setYValue()

The setYValue() method resets the Y value of the surface.

Syntax

public void setYValue(int *Y[]*)

public void setYValue(float *Y[]*)

public void setYValue(double *Y[]*)

public void setYValue(String *sName*)

Arguments

Y

The new Y value of the surface.

sName

The name of the IDL variable that contains the new Y value of the surface.

Exceptions

None.

setZValue()

The setZValue() method resets the Z value of the surface.

Syntax

public void setZValue(int *Z[][]*)

public void setZValue(float *Z[][]*)

public void setZValue(double *Z[][]*)

public void setZValue(String *sName*)

Arguments

Z

The new Z value of the surface.

sName

The name of the IDL variable that contains the new Z value of the surface.

Exceptions

None.

IONMap / IONJMap Class

IONMap is an IONGrDrawable object that creates a map projection on which to display data. Displayed data can be any combination of IONGrMapImage, IONGrMapGrid, IONGrMapContinents and IONGrContour objects. Data is displayed in the order it is added.

The IONJMap class extends the IONJGrDrawable class and contains an IONGrMap object. It can be inserted into a component tree.

Class Declaration

```
public class IONMap
             extends IONGrDrawable

public class IONJMap
             extends IONJGrDrawable
```

Methods

- **IONMap() / IONJMap() —** Constructs a new map centered at (*lat*, *lon*) with rotation *rot*.
- **draw() —** Produces and displays the graphic on the drawing surface of this class.
- **getProperty() —** Retrieves the specified property.
- **setLat(), setLon() —** Sets the lat/lon on which to center the projection.
- **setProperty()** — Sets the value of the specified property.
- **setRotation() —** Sets the rotation of the map projection.

IONMap() / IONJMap()

The IONMap() method constructs a new map centered at (*lat*, *lon*) with rotation *rot*.

Syntax

Note

The following is the syntax for the IONMap() method. For the IONJMap() method, replace IONMap with IONJMap.

public IONMap(int *iWidth*, int *iHeight*)

public IONMap(int *iWidth*, int *iHeight*, int *lat*)

public IONMap(int *iWidth*, int *iHeight*, float *lat*)

public IONMap(int *iWidth*, int *iHeight*, double *lat*)

public IONMap(int *iWidth*, int *iHeight*, int *lat*, int *lon*)

public IONMap(int *iWidth*, int *iHeight*, float *lat*, float *lon*)

public IONMap(int *iWidth*, int *iHeight*, double *lat*, double *lon*)

public IONMap(int *iWidth*, int *iHeight*, int *lat*, int *lon*, int *rot*)

public IONMap(int *iWidth*, int *iHeight*, float *lat*, float *lon*, float *rot*)

public IONMap(int *iWidth*, int *iHeight*, double *lat*, double *lon*, double *rot*)

Arguments

iHeight

The height of the drawing area.

iWidth

The width of the drawing area.

lat

The latitude of the point on the Earth's surface to be mapped to the center of the projection plane. Latitude is measured in degrees north of the equator, and *lat* must be in the range $-90° \leq lat \leq 90°$. The default is 0.

lon

The longitude of the point on the Earth's surface to be mapped to the center of the projection plane. Longitude is measured in degrees east of the Greenwich meridian, and *lon* must be in the range -180° ≤ *lon* ≤ 180°. The default is 0.

rot

The angle through which the North direction should be rotated around the line L between the Earth's center and the point (*lat*, *lon*). This angle is measured in degrees with the positive direction being clockwise around the line L, and must be in the range -180° ≤ *rot* ≤ 180°. The default is 0.

If the center of the map is at the North pole, North is in the direction *lon* + 180°. If the origin is at the South pole, North is in the direction *lon*.

Exceptions

None.

draw()

The draw() method displays the map projection in the drawing area for this object.

Syntax

public void draw()

Arguments

None.

Exceptions

None.

getProperty()

The getProperty() method retrieves the value of the specified property.

Syntax

public final IONVariable getProperty(String *sName*)

Arguments

sName

The name of the property to retrieve.

Properties Supported

The following IDL map properties are supported by IONMap.[get,set]Property. Refer to the IDL documentation on keywords available for use with the MAP_SET procedure for an explanation of each property:

Projection Types: AITOFF, ALBERS, AZIMUTHAL, CONIC, CYLINDRICAL, GNOMIC, GOODESHOMOLOSINE, HAMMER, LAMBERT, MERCATOR, MILLER_CYLINDRICAL, MOLLEWIDE, ORTHOGRAPHIC, ROBINSON, SATELLITE, SINUSOIDAL, STEREOGRAPHIC, TRANSVERSE_MERCATOR

Map Characteristics: ADVANCE, CHARSIZE, CLIP, COLOR, CONTINENTS, CON_COLOR, HIRES, E_CONTINENTS, E_GRID, E_HORIZON, GLINESTYLE, GRID, HORIZON, LABEL, LATALIGN, LATDEL, LATLAB, LONDEL, LONLAB, MLINESTYLE, NAME, NOBORDER, NOERASE, REVERSE, TITLE, USA, XMARGIN, YMARGIN

Projection Parameters: CENTRAL_AZIMUTH, ELLIPSOID, ISOTROPIC, LIMIT, SAT_P, SCALE, STANDARD_PARALLELS

Graphics: POSITION, T3D, ZVALUE

Exceptions

None.

setLat(), setLon()

The setLat()/setLon() methods set the latitude and longitude for the map projection.

Syntax

public void setLat(int *lat*)

ublic void setLat(float *lat*)

public void setLat(double *lat*)

public void setLat(String *lat*)

public void setLon(int *lon*)

public void setLon(float *lon*)

public void setLon(double *lon*)

public void setLon(String *lon*)

Arguments

lat

The latitude of the point on the Earth's surface to be mapped to the center of the projection plane. Latitude is measured in degrees north of the equator, and *lat* must be in the range -90° ≤ *lat* ≤ 90°. The default is 0.

lon

The longitude of the point on the Earth's surface to be mapped to the center of the projection plane. Longitude is measured in degrees east of the Greenwich meridian, and *lon* must be in the range -180° ≤ *lon* ≤ 180°. The default is 0.

Exceptions

None.

setProperty()

The setProperty() method sets the specified property to the specified value.

Syntax

public final void setProperty(String *sName*, IONVariable *vValue*)

Arguments

sName

The name of the property to set.

vValue

The value to which to set the property.

Properties Supported

The properties supported by the IONMap.[get,set]Property are the same as those supported by "getProperty()" on page 234. Refer to the IDL documentation on keywords available for use with the MAP_SET procedure for an explanation of each property.

Exceptions

None.

setRotation()

The setRotation() method sets the rotation for the map projection.

Syntax

public void setRotation(int *rot*)

public void setRotation(float *rot*)

public void setRotation(double *rot*)

public void setRotation(String *rot*)

Arguments

rot

The angle through which the North direction should be rotated around the line L between the Earth's center and the point (*lat*, *lon*). This angle is measured in degrees with the positive direction being clockwise around the line L, and must be in the range -180° ≤ *rot* ≤ 180°. The default is 0.

If the center of the map is at the North pole, North is in the direction *lon* + 180°. If the origin is at the South pole, North is in the direction *lon*.

Exceptions

None.

IONMouseListener Interface

The IONMouseListener interface defines the callback methods an object must define to be notified of mouse events occurring on an object that implements the IONDrawable interface.

In ION 1.4, this interface uses the AWT event model. It is recommended that you use the AWT events directly (`java.awt.event.MouseListener` and/or `java.awt.event.MouseMotionListener`). These provide a more robust and complete solution.

Class Declaration

```
public interface IONMouseListener
```

Methods

- **mouseMoved() —** Called when the mouse moves.
- **mousePressed() —** Called when a mouse button down event occurs.
- **mouseReleased() —** Called when a mouse button up event occurs.

Implementing Classes

IONGraphicsClient Class

mouseMoved()

Call the mouseMoved() method when a mouse cursor is moved in a drawable. Note that the Mouse Listener must have been registered in the drawable prior to calling mouseMoved().

Syntax

public abstract void mouseMoved(IONDrawable *drawable*, int *X*, int *Y*, long *when*, int *mask*)

Arguments

drawable

The IONDrawable object that the event occurred in.

X

The X location of the mouse.

Y

The Y location of the mouse.

when

The time when the event happened.

mask

Current mouse button state.

Exceptions

None.

mousePressed()

Call the mousePressed() method when a mouse button is pressed in a drawable. Note that the Mouse Listener must have been registered in the drawable prior to calling mousePressed().

Syntax

public abstract void mousePressed(IONDrawable *drawable*, int *X*, int *Y*, long *when*, int *mask*)

Arguments

drawable

The IONDrawable object in which the event occurred.

X

The X location of the mouse.

Y

The Y location of the mouse.

when

The time when the event happened.

mask

The button that was pressed.

Exceptions

None.

mouseReleased()

Call the mouseReleased() method when a mouse button is released in a drawable. Note that the Mouse Listener must have been registered in the drawable prior to calling mouseReleased().

Syntax

public abstract void mouseReleased(IONDrawable *drawable*, int *X*, int *Y*, long *when*, int *mask*)

Arguments

drawable

The IONDrawable object in which the event occurred.

X

The X location of the mouse.

Y

The Y location of the mouse.

when

The time when the event happened.

mask

Current mouse button state. The left mouse button is represented by 1 (one), the middle mouse button by 2, and the right mouse button by 4.

In Unix versions of Java, it is impossible to determine which mouse button was released if more than one button was pressed before the button release. As a result, on Unix platforms ION reports the following button release events:

Buttons Pressed	Button Release Reported by ION
left and middle	left
left and right	left
middle and right	right
left, middle, and right	left

Button release events are reported correctly in Windows versions of Java.

Exceptions

None.

IONOffScreen Class

Objects of the IONOffScreen class represent an invisible drawing area on which graphic output can be placed.

Class Declaration

```
public class IONOffScreen
             extends Object
             implements IONDrawable
```

Methods

- **IONOffScreen() —** Constructs an object of the IONOffScreen class.
- **createImage() —** Creates an offscreen image.
- **getImage() —** Returns the image that is being drawn.
- **getIONGraphics() —** Returns an ION graphics context for the device.

See also the description of the IONDrawable Interface.

IONOffScreen()

The IONOffScreen() method constructs an object of the IONOffScreen class.

Syntax

public IONOffScreen(int *width*, int *height*, Component *comp*)

Arguments

width

The width of the drawing area.

height

The height of the drawing area.

comp

A visible used to create images. This needs to be a component that is already visible on the users screen in order for the OffScreen to be properly created.

Exceptions

None.

Example

```
IONOffScreen offscreen = new IONOffScreen();
```

createImage()

Use the createImage() method to create an image of a given size.

Syntax

public abstract Image createImage(int *width*, int *height*)

Arguments

width

The width of the requested image

height

The height of the requested image

Exceptions

None

Example

```
Image im = draw.createImage(300, 300);
```

getImage()

The getImage() method returns the image of the current drawing area.

Syntax

public abstract Image getImage()

Arguments

None

Exceptions

None

Example

```
Image im = draw.getImage();
```

getIONGraphics()

The getIONGraphics() method returns a Graphics object that you can use to get graphics information on ION's drawing buffer or draw directly to. Unlike the getGraphics() method, getIONGraphics() allows you to affect the actual IDL drawable area. For example, you would use the getIONGraphics() method when manipulating the buffer using the COPY keyword to IDL's DEVICE procedure.

Syntax

public abstract Graphics getIONGraphics()

Arguments

None

Exceptions

None

Example

```
Graphics g = draw.getIONGraphics();
```

IONOutputListener Interface

The IONOutputListener interface defines the method that a class must implement to receive ION Server output text. The object must register itself with the addIONOutputListener() call.

Class Declaration

```
public interface IONOutputListener
```

Methods

- **IONOutputText() —** Retrieves a line of text from the ION Server.

Implementing Classes

IONGrConnection / IONJGrConnection Class, IONMapApplet

Example

For a simple example using IONOutputListener, see the "Version" example on the page of "Basic ION Java Applets" provided with the ION Java installation. See "Running the ION Java Examples" on page 48 for more information.

IONOutputText()

The IONOutputText() method is called when a line of output text is available from the ION Server.

Syntax

public abstract void IONOutputText(String *sLine*)

Arguments

sLine

A line of output text from the ION Server.

Exceptions

None.

IONPlot / IONJPlot Class

The IONPlot class extends the IONGrDrawable class and contains an IONGrPlot to provide a easy way of drawing IDL plots. It can be inserted into an AWT tree.

The IONJPlot class extends the IONJGrDrawable class and contains an IONGrPlot object. It can be inserted into a component tree.

Class Declaration

```
public class IONPlot
             extends IONGrDrawable

public class IONJPlot
             extends IONJGrDrawable
```

Methods

- **IONPlot() / IONJPlot()** — Constructs an object of the IONPlot class.
- **draw()** — Produces and displays the graphic on the drawing surface of this class.
- **getProperty()** — Gets the value of a property.
- **setProperty()** — Sets a property for the graphic.
- **setXValue()** — Sets the X value of the plot.
- **setYValue()** — Sets the Y value of the plot.

IONPlot() / IONJPlot()

The IONPlot() method constructs an object of the IONPlot class.

Syntax

Note

The following is the syntax for the IONPlot() method. For the IONJPlot() method, replace IONPlot with IONJPlot.

public IONPlot(int *iWidth*, int *iHeight*)

public IONPlot(int *iWidth*, int *iHeight*, int *X[]*)

public IONPlot(int *iWidth*, int *iHeight*, float *X[]*)

public IONPlot(int *iWidth*, int *iHeight*, double *X[]*)

public IONPlot(int *iWidth*, int *iHeight*, String *sName*)

public IONPlot(int *iWidth*, int *iHeight*, int *X[]*, int *Y[]*)

public IONPlot(int *iWidth*, int *iHeight*, float *X[]*, float *Y[]*)

public IONPlot(int *iWidth*, int *iHeight*, double *X[]*, double *Y[]*)

public IONPlot(int *iWidth*, int *iHeight*, String *sXName*, String *sYName*)

Arguments

iWidth

The width of the plot.

iHeight

The height of the plot.

X

The X values of the plot.

Y

The Y values of the plot.

sXName

The name of an IDL variable to use for the X values of this plot.

sYName

The name of an IDL variable to use for the Y values of this plot.

Exceptions

None.

draw()

The draw() method produces and displays a graphic in the drawing area that makes up this object.

Syntax

public void draw()

Arguments

None.

Exceptions

None.

getProperty()

The getProperty() method retrieves the current value of the specified property.

Syntax

public final IONVariable getProperty(String *sName*)

Arguments

sName

The name of the property.

Properties Supported

The following IDL Plot properties are supported by IONPlot.[get,set]Property. Refer to the IDL documentation on keywords available for use with the PLOT procedure for an explanation of each property:

BACKGROUND, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, ISOTROPIC, LINESTYLE, MAX_VALUE, MIN_VALUE, NOCLIP, NODATA, NOERASE, NORMAL, NSUM, POLAR, POSITION, PSYM, SUBTITLE, SYMSIZE, T3D, TICKLEN, TITLE, XCHARSIZE/YCHARSIZE/ZCHARSIZE, XGRIDSTYLE/YGRIDSTYLE/ZGRIDSTYLE, XLOG, XMARGIN/YMARGIN/ZMARGIN, XMINOR/YMINOR/ZMINOR, XRANGE/YRANGE/ZRANGE, XSTYLE/YSTYLE/ZSTYLE,

XTICKFORMAT/YTICKFORMAT/ZTICKFORMAT, XTICKINTERVAL/YTICKINTERVAL/ZTICKINTERVAL, XTICKLAYOUT/YTICKLAYOUT/ZTICKLAYOUT, XTICKLEN/YTICKLEN/ZTICKLEN, XTICKNAME/YTICKNAME/ZTICKNAME, XTICKS/YTICKS/ZTICKS, XTICKUNITS/YTICKUNITS/ZTICKUNITS, XTICKV/YTICKV/ZTICKV, XTICK_GET/YTICK_GET/ZTICK_GET, XTITLE/YTITLE/ZTITLE, YLOG, YNOZERO, ZVALUE

Exceptions

None.

Example

```
IONVariable value = getProperty(Property);
```

setProperty()

The setProperty() method sets a property for the plot object.

Syntax

public final void setProperty(String *sName*, IONVariable *vValue*)

Arguments

sName

The name of the property to set.

vValue

The value of the property.

Properties Supported

The IDL Plot properties supported by IONPlot.[get,set]Property are the same as those supported by “getProperty()” on page 248. Refer to the IDL documentation on keywords available for use with the PLOT procedure for an explanation of each property.

Exceptions

None.

setXValue()

The setXValue() method resets the X value of the plot.

Syntax

public void setXValue(int *X[]*)

public void setXValue(float *X[]*)

public void setXValue(double *X[]*)

public void setXValue(String *sName*)

Arguments

X

The new X value of the plot.

sName

The name of an IDL variable to use for the X value.

Exceptions

None

setYValue()

The setYValue() method resets the Y value of the plot.

Syntax

public void setYValue(int *Y[]*)

public void setYValue(float *Y[]*)

public void setYValue(double *Y[]*)

public void setYValue(String *sName*)

Argument

Y

The new Y value of the plot.

sName

The name of the IDL variable to use for the Y value.

Exceptions

None.

IONSurface / IONJSurface Class

The IONSurface class extends the IONGrDrawable class and contains an IONGrSurface object to provide a easy way of drawing IDL surfaces. It can be inserted into an AWT tree.

The IONJSurface class extends the IONJGrDrawable class and contains an IONGrSurface object. It can be inserted into a component tree.

Class Declaration

```
public class IONSurface
             extends IONGrDrawable

public class IONJSurface
             extends IONJGrDrawable
```

Methods

- **IONSurface() / IONJSurface()** — Constructs an object of the IONSurface class.
- **draw()** — Produces and displays the graphic on the drawing surface of this class.
- **getProperty()** — Gets the value of a property.
- **setNoErase()** — Specifies whether the object should be erased when another object is drawn.
- **setProperty()** — Sets a property for the graphic.
- **setXValue()** — Sets the X value of the surface.
- **setYValue()** — Sets the Y value of the surface.
- **setZValue()** — Sets the Z data of the surface.

IONSurface() / IONJSurface()

The IONSurface() method constructs an object of the IONSurface class.

Syntax

Note

The following is the syntax for the IONSurface() method. For the IONJSurface() method, replace IONSurface with IONJSurface.

public IONSurface(int *iWidth*, int *iHeight*)

public IONSurface(int *iWidth*, int *iHeight*, int *Z[][]*)

public IONSurface(int *iWidth*, int *iHeight*, float *Z[][]*)

public IONSurface(int *iWidth*, int *iHeight*, double *Z[][]*)

public IONSurface(int *iWidth*, int *iHeight*, String *sName*)

public IONSurface(int *iWidth*, int *iHeight*, int *Z[][]*, int *X[]*, int *Y[]*)

public IONSurface(int *iWidth*, int *iHeight*, float *Z[][]*, float *X[]*, float *Y[]*)

public IONSurface(int *iWidth*, int *iHeight*, double *Z[][]*, double *X[]*, double *Y[]*)

public IONSurface(int *iWidth*, int *iHeight*, String *sZName*, String *sXName*, String *sYName*)

Arguments

iWidth

The width of the plot.

iHeight

The height of the plot.

Z

The Z (data) values for the surface.

sName, sZName

The name of the IDL variable to use for the Z (data) values of the surface.

X

An array holding the values for the X coordinates of the grid.

Y

An array holding the values for the Y coordinates of the grid.

sXName

The name of the IDL variable holding the values for the X coordinates of the grid.

sYName

The name of the IDL variable holding the values for the Y coordinates of the grid.

Exceptions

None.

draw()

The draw() method produces and displays a graphic in the drawing area that makes up this object.

Syntax

public void draw()

Arguments

None.

Exceptions

None.

getProperty()

The getProperty() method retrieves the current value of the specified property.

Syntax

public final IONVariable getProperty(String *sName*)

Arguments

sName

The name of the property.

Properties Supported

The following IDL Surface properties are supported by IONSurface.[get,set]Property. Refer to the IDL documentation on keywords available for use with the SURFACE procedure for an explanation of each property:

AX, AZ, BACKGROUND, BOTTOM, CHARSIZE, CLIP, COLOR, DATA, DEVICE, FONT, HORIZONTAL, IMAGE, LEGO, LINESTYLE, LOWER_ONLY, MAX_VALUE, MIN_VALUE, NOCLIP, NODATA, NOERASE, NORMAL, PIXELS, POSITION, SAVE, SHADES, SKIRT, SUBTITLE, T3D, TICKLEN, TITLE, UPPER_ONLY, XLOG/YLOG/ZLOG, XCHARSIZE/YCHARSIZE/ZCHARSIZE, XGRIDSTYLE/YGRIDSTYLE/ZGRIDSTYLE, XMARGIN/YMARGIN/ZMARGIN, XMINOR/YMINOR/ZMINOR, XRANGE/YRANGE/ZRANGE, XSTYLE/YSTYLE/ZSTYLE, XTICKFORMAT/YTICKFORMAT/ZTICKFORMAT, XTICKINTERVAL/YTICKINTERVAL/ZTICKINTERVAL, XTICKLAYOUT/YTICKLAYOUT/ZTICKLAYOUT, XTICKLEN/YTICKLEN/ZTICKLEN, XTICKNAME/YTICKNAME/ZTICKNAME, XTICKS/YTICKS/ZTICKS, XTICKUNITS/YTICKUNITS/ZTICKUNITS, XTICKV/YTICKV/ZTICKV, XTICK_GET/YTICK_GET/ZTICK_GET, XTITLE/YTITLE/ZTITLE, ZAXIS, ZVALUE

Exceptions

None.

Example

```
IONVariable value = getProperty(Property);
```

setNoErase()

The setNoErase() method of the IONSurface class overrides setNoErase() in the IONGrDrawable class. The setNoErase() method of the IONJSurface class overrides

setNoErase() in the IONJGrDrawable class. See “setNoErase()” on page 195 for the syntax of this method.

setProperty()

The setProperty() method sets a property for the plot object.

Syntax

public final void setProperty(String *sName*, IONVariable *vValue*)

Arguments

sName

The name of the property to set.

vValue

The value to which to set the property.

Properties Supported

The properties supported by the IONSurface.[get,set]Property are the same as those supported by the “getProperty()” on page 254. Refer to the IDL documentation on keywords available for use with the SURFACE procedure for an explanation of each property:

Exceptions

None.

setXValue()

The setXValue() method resets the X value of the surface.

Syntax

public void setXValue(int *X[]*)

public void setXValue(float *X[]*)

public void setXValue(double *X[]*)

public void setXValue(String *sName*)

Arguments

X

The new X value of the surface.

sName

The name of the IDL variable that contains the new X value of the surface.

Exceptions

None.

setYValue()

The setYValue() method resets the Y value of the surface.

Syntax

public void setYValue(int *Y[]*)

public void setYValue(float *Y[]*)

public void setYValue(double *Y[]*)

public void setYValue(String *sName*)

Arguments

Y

The new Y value of the surface.

sName

The name of the IDL variable that contains the new Y value of the surface.

Exceptions

None.

setZValue()

The setZValue() method resets the Z value of the surface.

Syntax

public void setZValue(int *Z[][]*)

public void setZValue(float *Z[][]*)

public void setZValue(double *Z[][]*)

public void setZValue(String *sName*)

Arguments

Z

The new Z value of the surface.

sName

The name of the IDL variable that contains the new Z value of the surface.

Exceptions

None.

IONVariable Class

Objects of the IONVariable class provide a client-side representation of an IDL variable. IONVariable objects are used to read and write data between the IDL server and clients.

Note

IDL and Java both have a basic byte data type, however, IDL's byte is unsigned and Java's is signed. Java does not support the concept of unsigned types. When a byte in Java is cast to an integer, the sign is preserved via sign extension. This can cause problems when transferring byte data between IDL and Java. For information on how to properly convert an IDL byte to a Java byte, see "Converting Between IDL and Java Bytes" on page 100.

Class Declaration

```
public class IONVariable
             extends Object
```

Constants

The following constants are used to identify data types:

Type	Description
TYPE_UNDEFINED	Variable is of IDL type undefined
TYPE_BYTE	Variable is of IDL type byte
TYPE_INT	Variable is of IDL type int
TYPE_LONG	Variable is of IDL type long
TYPE_FLOAT	Variable is of IDL type float
TYPE_DOUBLE	Variable is of IDL type double
TYPE_STRING	Variable is of IDL type string
TYPE_COMPLEX	Variable is of IDL type complex
TYPE_DCOMPLEX	Variable is of IDL type double complex

Methods

- **IONVariable()** — Constructs an object of the IONVariable class.
- **arrayDimensions()** — Returns an int array that contains the array's dimensions.
- **getByte()** — Returns the byte value of the variable.
- **getByteArray()** — Returns the byte array of the variable.
- **getComplexArray()** — Returns the array of IONComplex values.
- **getDComplexArray()** — Returns the array of IONDComplex values.
- **getDImaginary()** — Returns the imaginary value of a double complex variable.
- **getDouble()** — Returns the double value of the variable.
- **getDimensionedByteArray()** — Returns the byte array value of the variable as an array with the same dimensions as the variable.
- **getDimensionedDoubleArray()** — Returns the double array value of the variable as an array with the same dimensions as the variable.
- **getDimensionedFloatArray()** — Returns the float array value of the variable as an array with the same dimensions as the variable.
- **getDimensionedIntArray()** — Returns the integer array value of the variable as an array with the same dimensions as the variable.
- **getDimensionedShortArray()** — Returns the short array value of the variable as an array with the same dimensions as the variable.
- **getDouble()** — Returns the double value of the variable.
- **getDoubleArray()** — Returns the double array value of the variable.
- **getFloat()** — Returns the float value of the variable.
- **getFloatArray()** — Returns the float array of the variable.
- **getImaginary()** — Returns the imaginary value of a complex variable.
- **getInt()** — Returns the int value of the variable.
- **getIntArray()** — Returns the int array of the variable.
- **getShort()** — Returns the short value of the variable.
- **getShortArray()** — Returns the short array value of the variable.

- **getString() —** Returns the string value of the variable.
- **getStringArray() —** Returns the string array value of the variable.
- **isArray() —** Returns true of the variable is an array.
- **toString() —** Returns a string that represents the variable value.
- **type() —** Returns the type of the variable.

IONVariable()

The IONVariable() method constructs an object of the specified IDL data type. The variable can be either a scalar or an array.

Syntax

Scalars

public IONVariable()

public IONVariable(byte *b*)

public IONVariable(short *s*)

public IONVariable(int *i*)

public IONVariable(float *f*)

public IONVariable(double *d*)

public IONVariable(String *s*)

public IONVariable(String *s*, boolean *b*)

public IONVariable(IONDComplex *cmplx*)

Arrays

public IONVariable(byte *b[]*, int *dims[]*)

public IONVariable(short *i[]*, int *dims[]*)

public IONVariable(int *i[]*, int *dims[]*)

public IONVariable(float *f[]*, int *dims[]*)

public IONVariable(double *d[]*, int *dims[]*)

public IONVariable(String *s[]*, int *dims[]*)

public IONVariable(IONComplex *cmplx[]*, int *dims[]*)

public IONVariable(IONDComplex *cmplx[]*, int *dims[]*)

Arguments

Most arguments are straightforward. If no arguments are specified, the IONVariable object corresponds to an IDL variable of type “Undefined”. Type “short” corresponds to IDL type “integer”, and type “int” corresponds to IDL type “long integer”. The

size of arrays and the array dimension array are determined through the use of the Java array length property.

Example

To create an IONVariable object of type float:

```
IONVariable oVariable = new IONVariable(1234.5678);
```

To create an IONVariable object of type float array of size (100,100,3):

```
float[] farr = new float[100*100*3];
int dims[] = new int[3];
dims[0] = 100;
dims[1] = 100;
dims[2] = 3;
oVariable = new IONVariable(farr, dims);
```

arrayDimensions()

The arrayDimensions() method returns an int array that contains the size of the dimensions of the array variable. If the variable is not an array, an exception is thrown.

Syntax

public final int[] arrayDimensions()

Return Value

The function returns an int array that contains the size of each dimension in the corresponding element of the array. The number of dimensions available can be determined through the length property of the returned array.

Arguments

None.

Exceptions

IONNotAnArrayException

Example

```
try {
int dims[] = arrayDimensions();
```

```
}catch(IONNotAnArrayException e){
   System.err.println("Variable is not an array");
}
```

getByte()

The getByte() method returns the byte value of the variable. If the value is not of type byte, the scalar value is converted to a byte.

Syntax

public final byte getByte()

Return Value

The method returns the byte value of the variable.

Arguments

None.

Exceptions

IONIsAnArrayException, NumberFormatException

Example

```
try {
byte b = myVariable.getByte();
}catch(IONIsAnArrayException e){
   System.err.println("Variable is an array");
}catch(NumberFormatException e){
   System.err.println("String Cannot be converted");
}
```

getByteArray()

The getByteArray() method returns the byte array value of the variable. If the value is not of type byte, a “java.lang.ClassCastException” exception will be thrown.

Syntax

public final byte[] getByteArray()

Return Value

The method returns the byte array value of the variable.

Arguments

None.

Exceptions

IONNotAnArrayException

Example

```
try {
byte b[] = myVariable.getByteArray();
}catch(IONNotAnArrayException e){
   System.err.println("Variable is not an array");
}
```

getComplexArray()

The getComplexArray() method returns the value of the complex array variable.

Syntax

public final IONComplex[] getComplexArray()

Return Value

The method returns the value of the complex array variable.

Arguments

None.

Exceptions

IONNotAnArrayException

Example

```
try {
IONComplex c[] = myVariable.getComplexArray();
}catch(IONNotAnArrayException e){
   System.err.println("Variable is not an array");
}
```

getDComplexArray()

The getDComplexArray() method returns the value of the double complex array variable. If the value is not of type double complex, a "java.lang.ClassCastException" exception will be thrown.

Syntax

public final IONDComplex[] getDComplexArray()

Return Value

The method returns the value of the double complex array variable.

Arguments

None.

Exceptions

IONNotAnArrayException

Example

```
try {
IONDComplex dc[] = myVariable.getDComplexArray();
}catch(IONNotAnArrayException e){
   System.err.println("Variable is not an array");
}
```

getDImaginary()

The getDImaginary() method returns the imaginary value of the double complex variable. If the value is not of type double complex, zero is returned.

Syntax

public final double getDImaginary()

Return Value

The method returns the imaginary value of the double complex variable.

Arguments

None.

Exceptions

IONIsAnArrayException

Example

```
try {
double i = myVariable.getDImaginary();
}catch(IONIsAnArrayException e){
   System.err.println("Variable is an array");
}
```

getDimensionedByteArray()

Returns the byte array value of the variable. The result contains the same number of dimensions as the variable.

Syntax

public final Object getDimensionedByteArray

(where Object can be a 1- to 8-dimensional array of Java primitive type 'byte')

Return

The method returns the multidimensional byte array value of the variable.

Arguments

None.

Exceptions

IONNotAnArrayException

Example

```
IONVariable myVariable = c_ionCon.getIDLVariable("my3dByteArr");
byte b3d[][][];
try {
b3d = (byte[][][])myVariable.getDimensionedByteArray();
} catch(IONNotAnArrayException e) {
   System.err.println("Variable is not an array");
}
```

getDimensionedDoubleArray()

Returns the double array value of the variable. The result contains the same number of dimensions as the variable.

Syntax

public final Object getDimensionedDoubleArray

(where Object can be a 1- to 8-dimensional array of Java primitive type 'double')

Return

The method returns the multidimensional double array value of the variable.

Arguments

None.

Exceptions

IONNotAnArrayException

Example

```
IONVariable myVariable = c_ionCon.getIDLVariable("my3dDoubleArr");
double d3d[][][];
try {
d3d = (double[][][])myVariable.getDimensionedDoubleArray();
} catch(IONNotAnArrayException e) {
   System.err.println("Variable is not an array");
}
```

getDimensionedFloatArray()

Returns the float array value of the variable. The result contains the same number of dimensions as the variable.

Syntax

public final Object getDimensionedFloatArray

(where Object can be a 1- to 8-dimensional array of Java primitive type 'float')

Return

The method returns the multidimensional float array value of the variable.

Arguments

None.

Exceptions

IONNotAnArrayException

Example

```
IONVariable myVariable = c_ionCon.getIDLVariable("my3dFloatArr");
float f3d[][][];
try {
f3d = (float[][][])myVariable.getDimensionedFloatArray();
} catch(IONNotAnArrayException e) {
   System.err.println("Variable is not an array");
}
```

getDimensionedIntArray()

Returns the integer array value of the variable. The result contains the same number of dimensions as the variable.

Syntax

public final Object getDimensionedIntArray

(where Object can be a 1- to 8-dimensional array of Java primitive type 'int')

Return

The method returns the multidimensional int array value of the variable.

Arguments

None.

Exceptions

IONNotAnArrayException

Example

```
IONVariable myVariable =
c_ionCon.getIDLVariable("my3dIdlLongArr");
int i3d[][][];
try {
i3d = (int[][][])myVariable.getDimensionedIntArray();
} catch(IONNotAnArrayException e) {
System.err.println("Variable is not an array");
}
```

getDimensionedShortArray()

Returns the short array value of the variable. The result contains the same number of dimensions as the variable.

Syntax

public final Object getDimensionedShortArray

(where Object can be a 1- to 8-dimensional array of Java primitive type 'short')

Return

The method returns the multidimensional short array value of the variable.

Arguments

None.

Exceptions

IONNotAnArrayException

Example

```
IONVariable myVariable = c_ionCon.getIDLVariable("my3dIdlIntArr");
short short3d[][][];
try {
short3d = (short[][][])myVariable.getDimensionedShortArray();
} catch(IONNotAnArrayException e) {
   System.err.println("Variable is not an array");
}
```

getDouble()

The getDouble() method returns the double value of the variable. If the value is not of type double, the scalar value is converted to a double.

Syntax

public final double getDouble()

Return Value

The method returns the double value of the variable.

Arguments

None.

Exceptions

IONIsAnArrayException, NumberFormatException

Example

```
try {
double d = myVariable.getDouble();
}catch(IONIsAnArrayException e){
   System.err.println("Variable is an array");
}catch(NumberFormatException e){
   System.err.println("String Cannot be converted");
}
```

getDoubleArray()

The getDoubleArray() method returns the double array value of the variable. If the value is not of type double, a “java.lang.ClassCastException” exception will be thrown.

Syntax

public final double[] getDoubleArray()

Return Value

The method returns the double array value of the variable.

Arguments

None.

Exceptions

IONNotAnArrayException

Example

```
try {
double d[] = myVariable.getDoubleArray();
}catch(IONNotAnArrayException e){
   System.err.println("Variable is not an array");
}
```

getFloat()

The getFloat() method returns the float value of the variable. If the value is not of type float, the scalar value is converted to a float.

Syntax

public final float getFloat()

Return Value

The method returns the float value of the variable.

Arguments

None.

Exceptions

IONIsAnArrayException, NumberFormatException

Example

```
try {
float f = myVariable.getFloat();
}catch(IONIsAnArrayException e){
   System.err.println("Variable is an array");
}catch(NumberFormatException e){
   System.err.println("String Cannot be converted");
}
```

getFloatArray()

The getFloatArray() method returns the float array value of the variable. If the value is not of type float, “java.lang.ClassCastException” exception will be thrown.

Syntax

public final float[] getFloatArray()

Return Value

The method returns the float array value of the variable.

Arguments

None.

Exceptions

IONNotAnArrayException

Example

```
try {
float f[] = myVariable.getFloatArray();
}catch(IONNotAnArrayException e){
   System.err.println("Variable is not an array");
}
```

getImaginary()

The getImaginary() method returns the imaginary value of the complex variable. If the value is not of type complex, zero is returned.

Syntax

public final float getImaginary()

Return Value

The method returns the imaginary value of the complex variable.

Arguments

None.

Exceptions

IONIsAnArrayException

Example

```
try {
float i = myVariable.getImaginary();
}catch(IONIsAnArrayException e){
   System.err.println("Variable is an array");
}
```

getInt()

The getInt() method returns the int value of the variable. If the value is not of type int (IDL type long), the scalar value is converted to an int.

Syntax

public final int getInt()

Return Value

The method returns the int value of the variable.

Arguments

None.

Exceptions

IONIsAnArrayException, NumberFormatException

Example

```
try {
int i = myVariable.getInt();
}catch(IONIsAnArrayException e){
   System.err.println("Variable is an array");
}catch(NumberFormatException e){
   System.err.println("String Cannot be converted");
}
```

getIntArray()

The getIntArray() method returns the int array value of the variable. If the value is not of type int, “java.lang.ClassCastException” exception will be thrown.

Syntax

public final int[] getIntArray()

Return Value

The method returns the int array value of the variable.

Arguments

None.

Exceptions

IONNotAnArrayException

Example

```
try {
int i[] = myVariable.getIntArray();
}catch(IONNotAnArrayException e){
   System.err.println("Variable is not an array");
}
```

getShort()

The getShort() method returns the short value of the variable. If the value is not of type short (IDL type int), the scalar value is converted to a short.

Syntax

public final short getShort()

Return Value

The method returns the short value of the variable.

Arguments

None.

Exceptions

IONIsAnArrayException, NumberFormatException

Example

```
try {
short s = myVariable.getShort();
}catch(IONIsAnArrayException e){
   System.err.println("Variable is an array");
}catch(NumberFormatException e){
   System.err.println("String Cannot be converted");
}
```

getShortArray()

The getShortArray() method returns the short array value of the variable. If the value is not of type short, “java.lang.ClassCastException” exception will be thrown.

Syntax

public final short[] getShortArray()

Return Value

The method returns the short array value of the variable.

Arguments

None.

Exceptions

IONNotAnArrayException

Example

```
try {
short s[] = myVariable.getShortArray();
}catch(IONNotAnArrayException e){
   System.err.println("Variable is not an array");
}
```

getString()

The getString() method returns the string value of the variable. If the value is not of type string, the scalar value is converted to a string.

Syntax

public final String getString()

Return Value

The method returns the string value of the variable.

Arguments

None.

Exceptions

IONIsAnArrayException

Example

```
try {
String st = myVariable.getString();
}catch(IONIsAnArrayException e){
   System.err.println("Variable is an array");
}
```

getStringArray()

The getStringArray() method returns the string array value of the variable. If the value is not of type string, “java.lang.ClassCastException” exception will be thrown.

Syntax

public final String[] getStringArray()

Return Value

The method returns the string array value of the variable.

Arguments

None.

Exceptions

IONNotAnArrayException

Example

```
try {
String st[] = myVariable.getStringArray();
}catch(IONNotAnArrayException e){
   System.err.println("Variable is not an array");
}
```

isArray()

The isArray() method determines if the value of the variable is an array.

Syntax

public final boolean isArray()

Return Value

This method returns true if the variable is an array and false if the variable is not.

Arguments

None.

Exceptions

None.

Example

```
boolean bIsArray = myVariable.isArray();
```

toString()

The toString() method returns a string representation of the variables value.

Syntax

public final String toString()

Return Value

A string that represents the value of the variable. The string is in a format that can be understood by IDL.

Arguments

None.

Exceptions

None.

Example

```
String s = myVariable.toString();
```

type()

The type() method returns the type of the value the variable contains. This return value is one of the constant type codes which are a part of this object.

Syntax

public final int type()

Return Value

This function returns the type code of the variable.

Arguments

None.

Exceptions

None.

Example

```
int typeCode = myVariable.type();
```

IONWindowingClient Class

The IONWindowingClient class provides mechanisms to handle the processing of the windowing commands that are part of an IDL Direct Graphics driver. This includes the creation, deletion, showing, hiding and iconization of windows on the client.

Class Declaration

```
public class IONWindowingClient
             extends IONGraphicsClient
```

Methods

- **IONWindowingClient() —** Constructs an object of the IONWindowingClient class.
- **connect() —** Connects to the server.
- **createWindow() —** Creates a window on the client
- **deleteWindow() —** Deletes a given window or pixmap
- **showWindow() —** Shows/hides a window

IONWindowingClient()

The IONWindowingClient() method constructs an IONWindowingClient object. The connect method (from IONGraphicsClient) must be called to establish a connection between the client and the server.

Syntax

public IONWindowingClient(Component *comp*)

Arguments

comp

A Java AWT Component that is used to reference the display being used for the graphics. This is needed for creating offscreen images.

Exceptions

None.

connect()

See “connect()” on page 125.

createWindow()

The createWindow() method creates a drawing area of the given size, places that area in its own window frame, make the window the current destination for graphics output and returns the IDL window index of the new window. If a title is not specified, the default IDL windowing convention is used (IDL 0, IDL 1, ...).

Syntax

public int createWindow(int *xsize*, int *ysize*)

public int createWindow(int *xsize*, int *ysize*, String *title*)

public int createWindow(int *index*, int *xsize*, int *ysize*)

public int createWindow(int *index*, int *xsize*, int *ysize*, String *title*)

public int createWindow(int *index*, int *xsize*, int *ysize*, int *xpos*, int *ypos*, String *title*)

Return Value

This method returns the IDL window index of the newly created window.

Arguments

xsize

The width in pixels of the window to be created

ysize

The height in pixels of the window to be created

title

The title of the window to be created

index

The desired IDL window index of the window

Exceptions

None

Example

```
IONWindowingClient ionWin = New IONWindowingClient();
int index = ionWin.createWindow( xsize, ysize, title);
```

For another detailed example, see the `window.java` example located in the `examples/src` directory.

deleteWindow()

Use the deleteWindow() method to delete the window/pixmap that is referenced by the given IDL Window index.

Syntax

public void deleteWindow(int *index*)

Arguments

index

The IDL Window index of the window/pixmap to destroy

Exceptions

None

showWindow()

Use the showWindow() method to raise or lower the Z order of the given window.

Syntax

public void showWindow(int *index*, boolean *show*)

Arguments

index

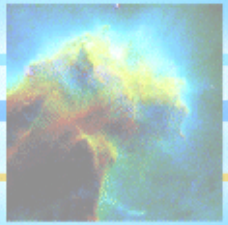
The IDL Window index of the window to iconize

show

Flag used indicate if the window should be shown or hidden

Exceptions

None



Chapter 7: Troubleshooting

Using ION applets over the World Wide Web requires interaction between your Web server, the ION Server, and IDL. This section discusses some possible ION Server and IDL problems you may encounter:

- Avoiding Conflicting ION Installations
- Checking Web Server Communication
- Troubleshooting ION Service Problems
- Troubleshooting Applets that Fail to Display
- Troubleshooting “Not Found” Errors
- Troubleshooting Licensing Errors
- Setting the IDL Path
- Troubleshooting Security Errors
- Encountering Browser Timeouts with Java Errors

Avoiding Conflicting ION Installations

Unexpected errors occur when attempting to use previous ION classes with the ION 2.0 server or vice versa. To avoid conflicts, remove the previous ION installations's service, and all previous ION class, .zip, and .jar files from your system before installing ION 2.0.

Removing Previous ION Service

To remove the previous ION Service, complete the following steps:

On Windows

1. Open the ION Properties utility, wionprop.exe, located in *RSI-DIR*\ion\ion_java\bin directory where *RSI-DIR* is the directory where you have installed your outdated ION version.
2. Stop the ION Service by selecting the "Stop" button.
3. Remove the ION Service by selecting the "Remove" button.

After removing the previous ION service, uninstall your previous ION version.

On UNIX

1. At the shell prompt, change to the previous ION installation directory, *RSI-DIR*/ion/ion_java/bin where *RSI-DIR* is the directory where you have installed your outdated ION version.
2. Enter `iondown` to stop the ION daemon.

After removing the previous ION service, uninstall your previous ION version by deleting the directory tree.

Check the CLASSPATH Variable

If you have configured a system variable for CLASSPATH, make sure it is not referencing your previous ION version. See "Setting the Class Path" on page 96 for more information.

Check the ION Version

From within an application, you can see which version of ION your application is using by adding a call to the getClientVersion() method of the "IONCallableClient Class" on page 122. Make sure the resulting string value indicating the current version of ION is ION 2.0.

Checking Web Server Communication

Make sure you are able to communicate with the Web Server. For example, using the Apache Web server, you can enter a URL such as `http://localhost` or `http://`*`hostname`*, where *`hostname`* is the hostname or Windows machine name of your computer. If your Web server is properly configured, you should receive a page stating that the Apache Web server has been successfully installed on your site. Consult your Web server documentation to find out how to test your Web server.

Troubleshooting ION Service Problems

Make Sure the ION Service is Started

On Windows, select **Start** → **Programs** → **Research Systems ION 2.0** → **ION Java Status**. In the "ION Status" dialog, enter the hostname of your computer and click **Query**. If you receive a message, "Unable to connect to the ION Java Daemon," you should make sure the service is started and that the port number is correct. Use the "Control" tab of the ION Java Properties utility to start the ION service and to check the port number. See "The ION Java Properties Dialog" on page 11.

On UNIX, change to the *`RSI-DIR`*`/ion_2.0/ion_java/bin` directory and enter `ionstat` to query the ION Service. If you receive the message, "Error: Unable to locate ION Java Server," see "Starting the ION Daemon on UNIX" on page 22 for instructions on how to start the ION Service.

Check File Permissions

On Windows NT systems, only users with the administrator privileges are allowed to start and stop the ION service using the Services Control Panel. If you have Administrator privileges but continue to experience problems when trying to start or stop the ION service using the utility described in the section, "The ION Java Properties Dialog" on page 11, you may need to use the Windows task manager to end the process or reboot the server to resolve the problem.

Troubleshoot Port Number Problems

If the ION Service is not running on the default port number (7085), then the applet code must specify the port number. See "Connecting to the ION Server" on page 72 information about including a port number connection parameter within an applet.

Enable SOCKS Proxy to Resolve Firewall Connection Problems

If the client machine is located behind a firewall and the socket connection between the ION server and the client machine cannot be established, the user on the client machine should verify that the SOCKS proxy is enabled in their browser.

Verify SOCKS Proxy on Netscape Navigator

1. From the browser, select **Edit** → **Preferences**.
2. Expand the **Advanced** heading and select **Proxies**.
3. In the Proxies section of the Preferences dialog, select the "Manual proxy configuration" option and click **View** to view the SOCKS proxy settings.

Verify SOCKS Proxy on Internet Explorer

1. From the browser, select **Tools** → **Internet Options**.
2. Select the "Connections" tab and click the **LAN Settings** button.
3. In the "Proxy server" field, check the "Use a proxy server" box if necessary and click the **Advanced** button to view the SOCKS proxy settings.

On each browser, you will see a list of the proxies supported by the browser as well as other information your site uses to implement the proxy. If the SOCKS proxy field is blank, check with your System Administrator to see if your site supports the SOCKS proxy. If the SOCKS proxy is available, input the correct information.

Troubleshooting Applets that Fail to Display

When an applet fails to display, first make sure the ION Service is started. See "Checking Web Server Communication" on page 289. If you are on a slow connection (modem), you may need to wait for all of ION Java's class files to be downloaded to your browser. These Java class files are required by ION Java for proper operation. If neither of these issues seem to be the problem, check the following sections for additional possibilities.

Enable Java in Your Browser

Most web browsers include a setting that enables the use of Java applets in HTML pages. Make sure your browser is configured to allow Java to load by completing the following steps for your browser.

Enable Java in Netscape Navigator

1. In the browser, select **Edit** → **Preferences**.

2. Click on **Advanced** and make sure the “Enable Java” check box is selected.

Enable Java in Internet Explorer

1. In the browser, select **Tools** → **Internet Options**.
2. Select the “Advanced” tab.
3. At the bottom of the scroll window, make sure “JIT compiler for virtual machine enabled” is selected.

Restart the Browser

If an ION applet fails to load as expected, even though you are using the correct CODEBASE, you might need to shut down and restart the browser. It’s a good idea to shut down and restart the browser any time you make changes to your HTML or class files.

Check the Java Console Log

If your applet fails to function properly, always check the Java console. To open the java console, complete the following steps for your browser.

Open the Netscape Navigator Java Console

1. Open the Netscape browser.
2. Select **Communicator** → **Tools** → **Java Console**.

Open the Internet Explorer Java Console

1. Open the IE browser.
2. Select **View** → **Java Console**. If Java Console is not an active option, complete the following steps:
   A. Select **Tools** → **Internet Options** and click the **Advanced** tab.
   B. Select the “Java Console Enabled” and “Java Logging Enabled” options located at the bottom of the scroll window. Apply the changes.
   C. Restart your browser.
   D. Open the Java Console by selecting **View** → **Java Console**.

Note

If you are running the client and the server on the same machine, setting the system CLASSPATH environment variable can result in errors similar to the following, appearing in your browser’s Java console:

Netscape Java Console — #Applet exception:
error.java.lang.ClassFormatError:class already loaded
IE Java Console — Error getting package information: com/rsi/ion

To avoid such errors, specify the class path when compiling as described in the section, “Compiling .java Files” on page 96.

Check the Debug Window

It is also helpful to set the ION applet Debug Mode, which allows you to check for IDL command log output or java program output for errors. See “Debug Mode” on page 99 for more information.

Note

Applets may also fail display because of security errors. See “Troubleshooting Security Errors” on page 294 for more information.

Troubleshooting “Not Found” Errors

Check the Location of Class Files

If you encounter an error that looks like:

```
Applet xxx can't start: class xxx not found
```

in the message area of your browser or in the Java Console, check to make sure that the ION package (the directory hierarchy `com/rsi/ion/*`) or the appropriate ION archive file is located either in the same directory as the HTML page that contains the applet or in the directory specified by the CODEBASE attribute of the APPLET tag. See “Locating the Class Files for use by ION Applets” on page 102 and “Supporting Java Archive Files” on page 103 for details.

Note

Class names are case sensitive. Within your Java code, calling `customaction.class` when the file has been saved as `CustomAction.class` and can produce a “class not found” error.

Check File Permissions

The ION Daemon runs with the user and group ID of the user who started it. This means that the daemon will have the same file access permissions as that user. While it is not necessary to start the ION Daemon as a particularly privileged user, make

sure that the access permissions for the ION class files and any class files you create are such that the ION Daemon has read permission.

If your applet does not run and the Java Console shows something like the following:

```
# Applet exception: class myApplet not found
```

where you know that the `myApplet.class` file exists and is located in the designated place, you may have a file permissions problem.

Troubleshooting Licensing Errors

If you get a license manager error on Windows stating that the license file cannot be found, your ION Java installation may not have been properly licensed. Review the licensing instructions in the Installing and Licensing IDL manual and make sure you have properly licensed ION.

Note

Since Web Servers do not read any system environment variables, you cannot use the LM_LICENSE_FILE environment variable to point to where you have located your license file. When licensing ION Java, you must place your license file in the default location *`RSI-DIR`*`\license\license.dat` where *`RSI-DIR`* is the name of the main installation directory you selected to install ION Java. On UNIX, if you have a network license that you wish your ION installation to use, you can copy the network license file to the default location *`RSI-DIR`*`\license\license.dat` on the machine on which you are running ION.

Note

Licensing errors appear on the server machine, not the client machine.

Setting the IDL Path

The IDL Search Path is used to specify the search path used by IDL for `.pro` and `.sav` files. If you call user-written IDL routines from an applet, make sure that the `.pro` files are located in IDL's path. You can set the IDL Path using one of the following methods:

- Place the directory that contains your `.pro` files in the path specified by the IDL_PATH environment variable by adding your directory in the "Path" tab of the Preferences dialog, accessed by selecting **File → Preferences** in the IDL Development Environment.

- Explicitly alter the value of the IDL system variable !PATH within your applet code.
- Set the IDL Path on Windows using the “Locations” tab of ION Java Properties dialog. See “The ION Java Properties Dialog” on page 11.

To specify multiple IDL Path search directories, separate each directory with a semicolon (Windows) or a colon (UNIX). Place the “+” symbol at the beginning of a directory to indicate that all subdirectories of a specified directory should be searched. For example the following Windows IDL Search Path specifies that the directory `C:\Program Files\Apache Group\apache2\htdocs\IONJava` and all its subdirectories be searched as well as the `C:\java_source` directory:

```
+C:\Program Files\Apache Group\apache2\htdocs\
IONJava;C:\java_source
```

On UNIX, the following IDL Search Path specifies that the directory `/usr/local/apache2/htdocs/ionjava` and all of its subdirectories be searched as well as the `/home/java` directory:

```
+/usr/local/apache2/htdocs/ionjava:/home/java
```

If ION attempts to compile and run a `.pro` file that is not in the path, no output will be generated but no error will be displayed. The best way to catch errors like this is to enable the ION applet Debug Mode and check the IDL command log output. See “Debug Mode” on page 99 for more information.

Troubleshooting Security Errors

Use http:// Instead of file:// with Internet Explorer

Security errors result when attempting to open HTML files containing applets using the IE browser’s **File** → **Open** command or when double-clicking an .html file to open it in the browser. When using Internet Explorer, always use an URL to specify the file location, for example, `http://www.mydomain.com/htmlfiles/index.html`.

Check Security Command Settings

Applets that violate the screening process specified by security command settings will result in errors. See “Command Security” on page 34 to see if your applet is failing the ION Service’s security criteria.

Encountering Browser Timeouts with Java Errors

If you do encounter an error when running a Java applet, some browsers' Java virtual machines will "hang," requiring you to shut down and restart the browser. It is generally a good idea to restart your browser after a Java error.

When the error is in an ION applet, there is a chance that the connection to the ION Server is still active when you close your browser. In this case, your browser may not start again immediately; it will wait for the ION socket connection to time out before shutting down and allowing you to start the browser again.

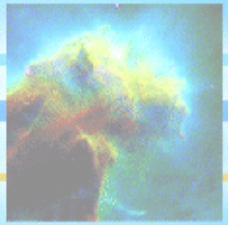
On Unix systems, you can use the kill command to prematurely kill the browser process and close the socket connection. On Windows NT systems, use the "Processes" tab of the Task Manager dialog to end a browser process. If you do not manually kill the browser process, the socket connection will automatically time out in 60 seconds.

ION Server Timeout

The ION Server may time out if you leave an applet unattended. You can change the amount of time the ION Server will wait before timing out. For more information, see Chapter 1, "Configuring ION Java".

JDK 1.2 Required for Clients

Clients may encounter problems if they are using a pre-JDK 1.2 virtual machine. JDK 1.2, 1.3, or 1.4 is required for connections to ION Java 1.6. User's can check their browser configurations for information on which version they are using.



Index

Symbols

$ (line continuation character), 44

A

addDrawable() method, 174
addGraphic() method, 189
addIONCommandDoneListener() method, 124
addIONDisconnectListener() method, 124
addIONDrawable() method, 164
addIONMouseListener() method, 135
addIONOutputListener() method, 125
ALIGN attribute, 68
ALT attribute, 68
APPLET tag
  ALIGN attribute, 68
  ALT attribute, 68
APPLET tag (*continued*)
  ARCHIVE attribute, 69
  CODE attribute, 69
  CODEBASE attribute, 69
  HEIGHT attribute, 70
  HSPACE attribute, 70
  NAME attribute, 70
  VSPACE attribute, 70
  WIDTH attribute, 70
applets
  attributes, 68
  compiling, 96
  controlling with scripts, 109
  creating, 102
  debugging, 74
  including in HTML pages, 102
  ION pre-built, 43
  IONContourApplet, 61, 78

applets (*continued*)
  IONGraphicApplet, 61, 76
  IONPlotApplet, 62, 84
  IONSurfaceApplet, 62, 86
  sharing connections, 74
applications, performance, 46
ARCHIVE attribute, 69
arrayDimensions() method, 263
attributes
  ALIGN, 68
  ALT, 68
  ARCHIVE, 69
  CODE, 69
  CODEBASE, 69
  HEIGHT, 70
  HSPACE, 70
  NAME, 70
  PARAM tags, 72
  VSPACE, 70
  WIDTH, 70
AWT, 64
AYSNC_COMMANDS parameter, 76

B

bandwidth, 46
byte data type, converting between IDL and Java, 100

C

character size, setting, 93
class files
  class path, 96
  location, 53
class path, 96
client (applet) verification, 35
CODE attribute, 69
CODEBASE attribute, 69
color (ION device), 90
command line parameters (ION daemon), 22
command security, 34
compiling applets, 96
configuring ION daemon, Windows, 11
connect() method
  IONCallableClient class, 125
  IONGraphicsClient class, 165
  IONGrConnection class, 175
  IONWindowingClient class, 283
connecting to the ION server, 72
connections
  limit, 35
  maximum number, 24, 33
  sharing, 74
contour plots, 61, 78
contour_property parameter, 79
COPY keyword (ION device), 90
copyArea() method, 167
createImage() method, 160, 243
createWindow() method, 283
creating ION applets, 102
current font, 92

D

daemon, 41
debug mode, 99
DEBUG_MODE parameter, 74
debugging, 74
debugging applications, 99
debugMode() method, 99
  IONGrConnection class, 175
  IONGrDrawable class, 190
DECOMPOSED keyword (ION device), 90
DECOMPOSED_COLOR parameter, 76
deleteWindow() method, 284
disconnect() method
  for scripts, 111
  IONCallableClient class, 127
  IONGrConnection class, 176
doubleValue() method, 142, 154

draw() method
IONContour class, 148
IONGrContour class, 182
IONGrDrawable class, 191
IONGrGraphic class, 198
IONGrMap class, 204
IONGrMapContinents class, 210
IONGrMapGrid class, 212
IONGrMapImage class, 216
IONGrPlot class, 221
IONGrSurface class, 227
IONMap class, 234
IONPlot class, 248
IONSurface class, 254
drawing, 63

E

error handling, 98
examples
running applets, 67
simple applet, 105
using JavaScript, 111
using VBScript, 113
exceptions, handling, 98
exclude commands, 23
exclude file, 22
executeIDLCommand() method
for scripts, 111
IONCallableClient class, 127
IONGrConnection class, 176
IONGrDrawable class, 191
execution of IDL commands, 122

F

filtering, 34
floatValue() method
IONComplex class, 142
IONDComplex class, 154
FONT keyword (ION device), 91
fonts, available, 92
fonts, specifying, 91

G

GET_CURRENT_FONT keyword (ION device), 92
GET_FONTNAMES keyword (ION device), 92
GET_GRAPHICS_FUNCTION keyword (ION device), 92
GET_SCREEN_SIZE keyword (ION device), 92
getByte() method, 264
getByteArray() method, 264
getClientVersion() method, 128
getComplexArray() method, 265
getConnection() method, 192
getConnectionType() method, 128
getCurrentIndex() method, 167
getDComplexArray() method, 266
getDImaginary() method
IONComplex class, 143
IONDComplex class, 155
IONVariable class, 267
getDouble() method, 271
getDoubleArray() method, 272
getDownButtons() method, 136
getFloat() method, 272
getFloatArray() method, 274
getGraphics() method, 137, 161, 244
getIDLVariable() method, 129
getImage() method, 137, 161, 244
getImaginary() method
IONComplex class, 143
IONDComplex class, 155
IONVariable class, 274
getInt() method, 275
getIntArray() method, 276
getIONDrawableIndices() method, 168

getMousePos() method, 138
getNumIndices() method, 168
getProperty() method, 216
  IONContour class, 149
  IONGrContour class, 182
  IONGrGraphic class, 198
  IONGrPlot class, 222
  IONGrSurface class, 228
  IONMap class, 234
  IONPlot class, 248
  IONSurface class, 254
getPropertyNames() method, 199
getPropertyString() method, 199
getShort() method, 277
getShortArray() method, 277
getString() method, 278
getStringArray() method, 279
graphics devices, ION, 90
graphics java classes, 43

H

HEIGHT attribute, 70
HSPACE attribute, 70
HTTP, ION Tunnel Broker, 31, 41

I

IDL
  command execution, 122
  command log output, 99
  search path, 15
  Widgets, 44
IDL_COMMAND parameter, 76
images, size of, 46
importing the ION package, 102
include commands, 23
include file, 23
including applets in HTML pages, 102
intValue() method
  IONComplex class, 144
  IONDComplex class, 156
ION
  class files, 53
  connecting to the server, 72
  controlling applets with scripts, 109
  error handling, 98
  graphics device, 90
    keywords accepted, 90
  graphics objects
    drawing, 63
    getting properties, 63
    setting properties, 63
    setting values, 63
  IDL limitations, 44
  low-level classes, 57
  pre-built applets, 61, 67
  server limitations, 44
  using graphics classes, 63
ION daemon, 41
  checking status (UNIX), 27
  checking status (Windows), 19
  client verification, 35
  command line parameters, 22
  configuring on UNIX, 22
  configuring on Windows, 11
  overview, 10
  port number, 13, 24
  security, 35
  security tokens, 25
  shutting down, 27
  starting on UNIX, 22
  starting on Windows, 12
  starting with Services Manager, 21
ION device
  COPY keyword, 90
  DECOMPOSED keyword, 90
  FONT keyword, 91
  GET_CURRENT_FONT keyword, 92
  GET_FONTNAMES keyword, 92

ION device (*continued*)
GET_GRAPHICS_FUNCTION keyword, 92
GET_SCREEN_SIZE keyword, 92
keywords accepted, 90
SET_CHARACTER_SIZE keyword, 92
SET_GRAPHICS_FUNCTION keyword, 93
ION Graphics Java Classes, 43
ION HTTP Tunnel Broker, 31, 41
ION Java Properties dialog, 11
ION Low-Level Java Classes, 43
ION methods available, 111
ION package, importing, 102
ION server, 40
connection limit, 35
security, 34
security files, 34
security system, 41
ION Service *See also* ION daemon.
ION Tunnel Broker, port number, 32
ION_CONNECTION_NAME parameter, 74
ion_httpd command, 32
IONCallableClient class, 57, 122, 124
addIONCommandDoneListener() method, 124
addIONDisconnectListener() method, 124
addIONOutputListener() method, 125
connect() method, 125
disconnect() method, 127
executeIDLCommand() method, 127
getCConnectionType() method, 128
getClientVersion() method, 128
getIDLVariable() method, 129
IONCallableClient() method, 123
removeIONCommandDoneListener() method, 130
removeIONDisconnectListener() method, 130
removeIONOutputListener() method, 131
sendIDLCommand() method, 131
setConnectionMethod() method, 132
IONCallableClient class, (*continued*)
setConnectionTimeout() method, 132
setIDLVariable() method, 133
IONCallableClient() method, 123
IONCanvas class, 58, 134
addIONMouseListener() method, 135
getDownButtons() method, 136
getGraphics() method, 137
getImage() method, 137
getMousePos() method, 138
IONCanvas() method, 135
removeIONMouseListener() method, 138
IONCanvas() method, 135
IONCommandComplete() method, 139
IONCommandDoneListener interface, 58, 139
IONCommandDoneListener interface class, IONCommandComplete() method, 139
IONComplex class, 58, 141
doubleValue() method, 142
floatValue() method, 142
getDImaginary() method, 143
getImaginary() method, 143
intValue() method, 144
IONComplex() method, 141
longValue() method, 144
toString() method, 145
IONComplex() method, 141
IONContour class, 59, 146
draw() method, 148
getProperty() method, 149
IONContour() method, 146
setNoErase() method, 150
setProperty() method, 150
setXValue() method, 150
setYValue() method, 151
setZValue() method, 152
IONContour() method, 146
IONContourApplet, 61, 78
IONdaemon, command line parameters, 20

IONDComplex class, 58, 153
  doubleValue() method, 154
  floatValue() method, 154
  getDImaginary() method, 155
  getImaginary() method, 155
  intValue() method, 156
  IONDComplex() method, 153
  longValue() method, 156
  toString() method, 157
IONDComplex() method, 153
IONDisconnection() method, 158
IONDisconnectListener interface class, 158
  IONDisconnection() method, 158
iondown utility, 27
IONDrawable class, 57
IONDrawable interface, 160
  createImage() method, 160
  getGraphics() method, 161
  getImage() method, 161
IONGR2Drawable interface, 58
IONGraphicApplet, 61, 76
IONGraphicsClient, readImage() method, 169
IONGraphicsClient class, 57, 163
  addIONDrawable() method, 164
  connect() method, 165
  copyArea() method, 167
  getCurrentIndex() method, 167
  getIONDrawableIndices() method, 168
  getNumIndices() method, 168
  IONGraphicsClient() method, 164
  removeIONDrawable() method, 170
  setDecomposed() method, 170
  setIONDrawable() method, 171
IONGraphicsClient() method, 164
IONGrConnection class, 59, 172, 173
  addDrawable() method, 174
  debugMode() method, 175
  executeIDLCommand() method, 176
  IONGrConnection() method, 174
  removeDrawable() method, 177
  sendIDLCommand() method, 178
IONGrConnection class, (*continued*)
  setDrawable() method, 178
  setYValue() method, 186
IONGrConnection() method, 174
IONGrContour class, 60, 180
  draw() method, 182
  getProperty() method, 182
  IONGrContour() method, 180
  setNoErase() method, 184
  setProperty() method, 183
  setXValue() method, 184
  setZValue() method, 186
IONGrContour() method, 180
IONGrDrawable class, 59, 188
  addGraphic() method, 189
  debugMode() method, 190
  draw() method, 191
  executeIDLCommand() method, 191
  getConnection() method, 192
  IONGrDrawable() method, 189
  isConnected() method, 192
  removeGraphic() method, 193
  resetMulti() method, 193
  sendIDLCommand() method, 194
  setMulti() method, 195
  setNoErase() method, 195
IONGrDrawable() method, 189
IONGrGraphic class, 60, 197
  getProperty() method, 198
  getPropertyNames() method, 199
  getPropertyString() method, 199
  IONGrGraphic() method, 197, 198
  registerProperty() method, 200
  setNoErase() method, 201
  setProperty() method, 201
IONGrGraphic() method, 197
IONGrMap class, 60
  draw() method, 204
  getProperty() method, 205
  IONGrMap(), 203

IONGrMap class, 60 (*continued*)
setProperty() method, 207
IONGrMapContinents class, 60
draw() method, 210
getProperty() method, 210
IONGrMapContinents() method, 209
setProperty() method, 211
IONGrMapContinents() method, 209
IONGrMapGrid class, 60
draw() method, 212
getProperty() method, 213
IONGrMapGrid() method, 212
setProperty() method, 214
IONGrMapGrid() method, 212
IONGrMapImage class, 61, 216, 218
draw() method, 216
getProperty() method, 216
IONGrMapImage class, 215
setImage() method, 217
setProperty() method, 218
setStart() method, 218
IONGrMapImage() method, 215
IONGrPlot class, 61, 220
draw() method, 221
getProperty() method, 222
IONGrPlot() method, 220
setNoErase() method, 223
setProperty() method, 223
setXValue() method, 224
setYValue() method, 224
IONGrPlot() method, 220
IONGrSurface class, 61, 226
draw() method, 227
getProperty() method, 228
IONGrSurface() method, 226
setNoErase() method, 229
setProperty() method, 229
setXValue() method, 230
setYValue() method, 230
setZValue() method, 231
IONGrSurface() method, 226
IONJContour class, 146, 252
IONJContour() method, 146
IONJContour() method, 146
IONJMap class, 232
IONJPlot class, 246
IONMap class, 59
draw() method, 234
getProperty() method, 234
setProperty() method, 236
IONMouseListener interface class, 58, 238
mouseMoved() method, 238
mousePressed() method, 239
mouseReleased() method, 240
IONOffScreen class, 58, 242
createImage() method, 243
getGraphics() method, 244
getImage() method, 244
IONOffScreen() method, 242
IONOffScreen() method, 242
IONOutputListener interface, 58, 245
IONOutputListener interface class, IONOutputText() method, 245
IONOutputText() method, 245
IONPlot class, 59, 246
draw() method, 248
getProperty() method, 248
IONPlot() method, 246
setProperty() method, 249
setXValue() method, 250
setYValue() method, 250
IONPlot() method, 246
IONPlotApplet, 62, 84
ionstat utility, 27
IONSurface class, 60, 252
draw() method, 254
getProperty() method, 254
IONSurface() method, 253
setNoErase() method, 255
setProperty() method, 256
setXValue() method, 257
setYValue() method, 257

IONSurface class, 60, 252 (*continued*)
setZValue() method, 258
IONSurface() method, 253
IONSurfaceApplet, 62, 86
IONVariable class, 58, 259
arrayDimensions() method, 263
getByte() method, 264
getByteArray() method, 264
getComplexArray() method, 265
getDComplexArray() method, 266
getDImaginary() method, 267
getDouble() method, 271
getDoubleArray() method, 272
getFloat() method, 272
getFloatArray() method, 274
getImaginary() method, 274
getInt() method, 275
getIntArray() method
, 276
getShort() method, 277
getShortArray() method, 277
getString() method, 278
getStringArray() method, 279
IONVariable() method, 262
isArray() method, 279
toString() method, 280
type() method, 280
IONVariable() method, 262
IONWindowingClient class, 57, 282
createWindow() method, 283
deleteWindow() method, 284
IONWindowingClient() method, 282
showWindow() method, 284
IONWindowingClient() method, 282
isArray() method, 279
isConnected() method, 192

J

jar files, 53, 103
Java applets, pre-built, 61, 67
Java archive files, 103
Java classes, ION low-level, 57
java console, 291
JavaScript, 109
JavaScript and VBScript, differences between, 115

L

limitations
IDL, 44
server, 44
line continuation character, 44
LINK_URL parameter, 75
LiveConnect (Netscape browers), 110
log file, 24, 24, 33
longValue() method
IONComplex class, 144
IONDComplex class, 156
low-level Java classes, 43

M

maximum number of connections, 24, 33, 35
mouse operations, 122
mouseMoved() method, 238
mousePressed() method, 239
mouseReleased() method, 240

N

NAME attribute, 70

O

object graphics, 94
object references, 95
output log file, 24, 24, 33

P

packages, archive files, 103
PARAM Tags, 72
parameters
  ASYNC_COMMANDS, 76
  contour_property, 79
  DEBUG_MODE, 74
  DECOMPOSED_COLOR, 76
  IDL_COMMAND, 76
  ION_CONNECTION_NAME, 74
  LINK_URL, 75
  plot_property, 84
  PORT_NUMBER, 72
  SERVER_DISCONNECT, 72
  SERVER_NAME, 72
  surface_property, 87
  X_VALUES, 78, 84, 86
  Y_VALUES, 78, 84, 86
  Z_VALUES, 78, 86
path, IDL search, 15
performance, 46
pixel copy operation (ION device), 90
plot_property parameter, 84
plotting, 62, 84
port number, 13, 24, 32
PORT_NUMBER parameter, 72
Pre-Built ION Client Applets, 43

R

readImage() method, 169
registerProperty() method, 200
removeDrawable() method, 177
removeGraphic() method, 193
removeIONCommandDoneListener() method, 130
removeIONDisconnectListener() method, 130
removeIONDrawable() method, 170
removeIONMouseListener() method, 138
removeIONOutputListener() method, 131
resetMulti() method, 193

S

screen size, retrieving, 92
scripting languages, 109, 111
  differences, 115
search path, IDL, 15
security, 34
  exclude commands, 23
  exclude file, 22
  include commands, 23
  include file, 23
  ION server, 41
  lists, 25
sendIDLCommand() method
  IONCallableClient class, 131
  IONGrConnection class, 178
  IONGrDrawable class, 194
server, 40
SERVER_DISCONNECT parameter, 72
SERVER_NAME parameter, 72
Services Manager (Windows)
  ION daemon, services manager, 21
SET_CHARACTER_SIZE keyword (ION device), 92
SET_GRAPHICS_FUNCTION keyword (ION device), 93
SET_PLOT routine, 90
setConnectionMethod() method, 132
setConnectionTimeout() method, 132
setDecomposed() method, 170
setDrawable() method, 178
setIDLVariable() method, 133
setImage() method, 217
setIONDrawable() method, 171
setMulti() method, 195
setNoErase() method
  IONContour class, 150
  IONGrContour class, 184
  IONGrDrawable class, 195

setNoErase() method (*continued*)
IONGrGraphic class, 201
IONGrPlot class, 223
IONGrSurface class, 229
IONSurface class, 255
setProperty() method, 218
IONContour class, 150
IONGrContour class, 183
IONGrGraphic class, 201
IONGrPlot class, 223
IONGrSurface class, 229
IONMap class, 236
IONPlot class, 249
IONSurface class, 256
setStart() method, 218
setXValue() method
IONContour class, 150
IONGrContour class, 184
IONGrPlot class, 224
IONGrSurface class, 230
IONPlot class, 250
IONSurface class, 257
setYValue() method
IONContour class, 151
IONGrConnection class, 186
IONGrPlot class, 224
IONGrSurface class, 230
IONPlot class, 250
IONSurface class, 257
setZValue() method
IONContour class, 152
IONGrContour class, 186
IONGrSurface class, 231
IONSurface class, 258
showWindow() method, 284
shutting down the ION daemon, 27
simple applet example, 105
status, utility, 19
status, checking on UNIX, 27
status, checking on Windows, 19
surface plots, 62, 86
surface_property parameter, 87
Swing, 64

T

tips and tricks (building applets), 115
toString() method, 157
IONComplex class, 145
IONVariable class, 280
true-color displays, 90
Tunnel Broker, 41
configuring, 31
configuring on UNIX, 24
configuring on Windows, 18
connection types, 31
ion_httpd command, 32
starting, 32
type() method, 280

U

URL (CODEBASE attribute), 70
URL, linking to, 75

V

VBScript, 109
VSPACE attribute, 70

W

WIDTH attribute, 70

X

X_VALUES parameter, 78, 84, 86
X-Y plots, 62, 84

Y

Y_VALUES parameter, 78, 84, 86

Z

Z_VALUES parameter, 78, 86
zip file (of Java class files), 53