# NewsEdge Java API Guide

**Version 1.0**

Revision: October 2007

# Table of Contents

# Chapter 1: About This Guide

Moody’s Analytics builds and operates digital media distribution platforms for publishing and financial services.

This document describes the NewsEdge Java API, which allows you to search and download news stories and metadata in various formats. You can find information about news delivery formats such as NewsML, XMLNews, and NITF, in separate documents for each format. This document includes the following chapters:

"Quick Start,” page 7 (step-by-step tutorials for the most common tasks to help you get started)

“How To ...," page 18 (code snippets and instructions for many routine tasks)

"API," page 45 (documentation of all the classes, methods, and constants relevant to client applications)

“Quick Start Listings,” page 131 (Java command-line samples for tasks listed in the Quick Start chapter)

“Architectural Overview,” page 139 (architectural overview of NewsEdge Java API)

"Glossary,” page 143

This guide is intended for experienced Java developers who want to integrate NewsEdge news feeds into existing applications or build new custom applications around the news feeds. It is not necessary to read this document to be able to work with Moody’s Analytics’ delivery formats, since each has its own, self-contained manual.

## Getting Started

"Quick Start," page 7 provides step-by step tutorials on the four basic tasks required to obtain news from the NewsEdge News Server:

Connecting to the server

Searching historical news

Filtering real-time news

Retrieving the full text of a story

## Prerequisites

In order to use the NewsEdge Java API, you must have the following:

Java 1.4 or greater SDK

Experience developing Java applications

Internet access

Connection information supplied by NewsEdge Customer Support (username, password, and host)

## Document Conventions

This document uses the following conventions:

Class, object and method names appear in boldface. When a class, object or method appears as a separate entry (in other words, not within the text of a paragraph), it is left-justified, with exception information below it in a smaller font, and an explanation of the item centered, as in the following example:

MediaConnection (java.net.InetAddress address, gari.net.ProxyData)
throws MediaException, java.io.IOException

The constructor opens a connection to the media server.

Code samples appear in a courier font, as follows:

```
MediaConnection connection = new MediaConnection(hostName);
connection.requestImage(key);
MediaData image = connection.readImageResponse();
```

# Chapter 2: Quick Start

This chapter focuses on the four basic tasks required to obtain news from the NewsEdge News Server:

Connect to and ping the News Server

Perform a full text search of historical articles

Set up a filter and read real-time articles

Retrieve the full text of an article

The code examples in this chapter are intentionally simple and do not include error handling and other elements of robust production code.

After completing this Quick Start, refer to the chapters "How To ..." page 18 for other common tasks, and "API" page 45 for a complete list of the API classes.

## Before You Begin

Here's what you need to work through the examples:

Java development environment (version 1.4 or higher)

A copy of the NewsEdge Java API (gari.jar, provided by NewsEdge Customer Support) added to the Java load path

A copy of the Apache Foundation's Log4J logging library (log4j.jar, available from http://logging.apache.org/) added to the Java load path.

A copy of the Log4J property file (log4j.properties, provided by NewsEdge Customer Support) in the Java source directory of your application to configure logging out (see "How to Configure Logging" page 25 for a sample listing); if this is not present, warning messages appear, but your applications will still run

A connection to the Internet (with information about any proxies)

The hostname for your NewsEdge News Server (provided by NewsEdge Customer Support)

A username and password for connecting to the NewsEdge News Server (provided by NewsEdge Customer Support)

Experience writing, compiling, and running Java command-line applications

## Task 1: Connecting to the News Server

The following code sample is a short static method that you can use to connect to a NewsEdge News Server, using the username, hostname, and password supplied by NewsEdge Customer Support. The method returns a gari.inap.InapClient object (page 46):

```
private static InapClient
amcConnect(String username, String password, String hostname)
throws IOException, QuipException {

        LoginData login = new LoginData(username, password,
        "JAPI");
        InetAddress address = InetAddress.getByName(hostname);
        return new InapClient(address, Channel.STARTUP_SERVICE,
        login);
}
```

Note that this method may not work if there is a proxy server between your client and the NewsEdge News Server. For more information on how to handle proxies, see “How to Connect Through a Proxy,” page 21. If firewall rules prevent you from using the default port, it is also possible to connect to the News Server through port 80 (see “How to Connect Through Port 80,” page 22).

### Code Walk-Through

First, create a gari.inap.LoginData object (page 48) that contains username, password, and a string identifying the API version. Note that although you must include a value for the API version, the actual content does not matter: you can simply use “JAPI”, or a combination of “JAPI” and your company name (for example, “JAPIAcmeFinance”).

```
LoginData login = new LoginData(username, password, "JAPI");
```

Next, look up the IP address of the News Server. If the code cannot resolve the hostname, it throws a java.lang.IOException.

```
InetAddress address = InetAddress.getByName(hostname);
```

Finally, use these two pieces of information to create a gari.inap.InapClient object, which is the basic connection object for a NewsEdge News Server. You must also import the gari.net.Channel class (page 76) to get the STARTUP_SERVICE constant. The code throws a java.lang.IOException for a general networking problem, or a gari.net.QuipException (page 83) for a NewsEdge-specific problem (such as a bad password).

```
return new InapClient(address, Channel.STARTUP_SERVICE, login);
```

## How to Use the Code

To test this method, create a connection and test that it is active. Make sure that you replace USERNAME, PASSWORD, and HOSTNAME with your actual connection information, as supplied by NewsEdge Customer Support:

```
InapClient client = amcConnect(USERNAME, PASSWORD, HOSTNAME);

// Test if the connection is active
System.out.println("Connected to NewsEdge News Server.");
System.out.println("  Is Connected: " + client.isConnected());
```

## Example Application

For a complete, short Java command-line application to connect to and ping a NewsEdge News Server and display an error message (on failure) or some information from the server (on success), see “Listing: Connecting to the News Server,” page 131.

## Task 2: Searching Historical News

Now that you have connected to the NewsEdge News Server, you may want to perform a full text search through past articles stored on the server. Note that searching for historical news requires the gari.inap.InapClient object you created in “Task 1: Connecting to the News Server,” page 8.

The API supports many more search parameters, such as date ranges, sources, and so on, that this section does not describe. For more information on search parameters, see “Historical News, Notes,” page 28.

The following is a short static method for searching past articles:

```
private static ExtendedEnumeration
doHistorySearch(NewsChannel channel, String pattern)
throws IOException {

       HistorySearch search = new HistorySearch(pattern);
       search.doReadHeadlines(channel);
       HeadlineAnswer answer = new
       HeadlineAnswer(channel.receive());
       return answer.getElements();
}
```

Create the gari.news.NewsChannel object, where client is the gari.inap.InapClient object created in “Task 1: Connecting to the News Server,” page 8:

```
NewsChannel channel = new NewsChannel(client);
```

## Code Walk-Through

First, you must create a gari.news.HistorySearch object (page 95) using the full text search pattern in the pattern variable:

```
HistorySearch search = new HistorySearch(pattern);
```

The simplest search pattern is simply a series of keywords, such as “Wisconsin dairy” or “San Francisco” (case does not matter). You can create much more sophisticated searches, including ticker symbols and other information, as described in “Historical News, Notes,” page 28.

After creating a gari.news.HistorySearch object, have the HistorySearch object read its results into the NewsChannel. (The code throws a java.lang.IOException on error.)

```
search.doReadHeadlines(channel);
```

The channel's receive method will cause it to receive the search results into a buffer in a raw, wire format. The gari.news.HeadlineAnswer class (page 89) parses the results for the client application, by taking the buffer returned as an argument for its constructor:

```
HeadlineAnswer answer = new HeadlineAnswer(channel.receive());
```

Finally, the HeadlineAnswer object provides a gari.util.ExtendedEnumeration object (page 128) that the client application can use to iterate through the results:

```
return answer.getElements();
```

## How to Use the Code

To use this method, first create an InapClient object as described in “Task 1: Connecting to the News Server,” page 8, and then use it to create a NewsChannel (page 102) by passing the client as an argument to the channel's constructor:

```
InapClient client = amcConnect(USERNAME, PASSWORD, HOSTNAME);
NewsChannel channel = new NewsChannel(client);
```

Next, pass the channel and your search pattern to the doHistorySearch method (for more information about search pattern syntax, see “How to Search Historical Headlines, Notes,” page 28):

```
ExtendedEnumeration result = doHistorySearch(channel, "boise");
```

Finally, iterate through the enumeration, casting each element to the gari.news.Headline class (page 85):

```
while (searchResult.hasMoreElements()) {
       Headline metadata = (Headline) searchResult.nextElement();
       System.out.println("Headline: " + metadata.getText());
```

```
        System.out.println("Resource id: " +
        metadata.getResourceID()
                + "\n");
}
```

The Headline class contains several types of metadata about the article, including the headline (getText), the article's unique identifier (getResourceId), associated company stock ticker symbols (getSymbols), subject codes (getCodes), and others. For more information, see “Metadata and Content” (page 36).

## Example Application

For a complete, short Java command-line application to search historical news, see “Listing: Searching Historical News,” page 132.

## Task 3: Obtaining and Filtering Real-Time News

In addition to searching past news, your application can receive real-time news as it is released, by specifying a filter to indicate which real-time news stories you want to obtain. The code example provided here uses “*” as a wildcard value to obtain all the available news stories. You can specify other filters (see “How to Filter Real-Time Headlines,” page 35), but for the purposes of the sample, use the wildcard value – otherwise, it might take a long time to find a story that matches a specific filter.

Filtering real-time news has some similarities to searching historical news, but instead of receiving search results immediately, the application must set up an implementation of java.util.Observer to be notified whenever a batch of new stories arrives.

The following is a short static method for starting a real-time news filter through a gari.news.Profile object (page 103):

```
public static void startFilter
(NewsChannel channel, String pattern, Observer observer)
```

```
throws IOException {

       Profile profile = new Profile("Sample profile", "pattern");
       profile.setNumber(1);

       ProfileManager manager = new ProfileManager(profile,
       channel);
       manager.addObserver(new ProfileFilter());
}
```

WARNING: This code replaces any profile you currently have in slot 1. If you have an existing profile, change the argument in setNumber to use a different slot.

Create the gari.news.NewsChannel object (page 102), where client is the gari.inap.InapClient object created in “Task 1: Connecting to the News Server,” page 8:

```
NewsChannel channel = new NewsChannel(client);
```

The observer argument is any class that implements the java.util.Observer interface. To implement the interface, the class must include a public update method to process incoming news stories, like the following:

```
public void update(Observable src, Object arg) {
       HeadlineAnswer answer = ((ProfileHeadlines)
       arg).getHeadlines();
       ExtendedEnumeration headlines = answer.getElements();
       while (headlines.hasMoreElements()) {
              Headline metadata = (Headline)
       headlines.nextElement();
              // do something with the Headline object
       }
}
```

Each time a new batch of stories arrives, the API invokes this method so that your client application can process them. Because the notification takes place on a separate thread, your application can continue with other work while waiting for news to arrive.

## Code Walk-Through

First, create a new gari.news.Profile object. The first parameter is a label for the profile, and the second one is a search filter (“*” matches everything):

```
Profile profile = new Profile("Sample profile", "*");
```

Next, assign the profile to one of the available profile slots (see “How to Discover Your Entitlements,” page 26 to determine your allocated slots). It replaces any profile already using that slot:

```
profile.setNumber(1);
```

Set up a gari.news.ProfileManager (page 111) to watch for incoming news through the profile and associate the profile with a news channel:

```
ProfileManager manager = new ProfileManager(profile, channel);
```

Finally, register your client application's observer with the manager:

```
manager.addObserver(new ProfileFilter());
```

The API now invokes the observer's update method every time a new batch of matching news stories arrives.

## How to Use the Code

To use this method, first create an InapClient object as described in “Task 1: Connecting to the News Server,” page 8, and then use it to create a NewsChannel:

```
InapClient client = amcConnect(USERNAME, PASSWORD, HOSTNAME);
NewsChannel channel = new NewsChannel(client);
```

Next, create a class that implements the java.util.Observer interface. The update method receives an argument of type gari.news.ProfileHeadlines (page 110) that the client application can use to get a HeadlineAnswer object, as in the history search:

```
public void update(Observable src, Object arg) {
        HeadlineAnswer answer = ((ProfileHeadlines)
        arg).getHeadlines();
        ExtendedEnumeration headlines = answer.getElements();
        while (headlines.hasMoreElements()) {
           Headline metadata = (Headline) headlines.nextElement();
           System.out.println("Headline: " + metadata.getText());
           System.out.println("Resource id: " +
                                metadata.getResourceID() + "\n");
        }
}
```

Finally, pass the channel and your search pattern to the startFilter method (for more information about search pattern syntax, see “How to Search Historical Headlines, Notes,” page 28):

```
startFilter(channel, "*", observer);
```

The observer receives batches of stories until you end the application.

## Example Application

For a complete, short Java command-line application to obtain and filter real-time news, see “Listing: Obtaining and Filtering Real Time News,” page 134.

## Task 4: Retrieving the Full Text of an Article

The two previous examples in this chapter show how the client application can receive gari.news.Headline objects (page 85) either through a history search or by filtering incoming news in real-time through a Profile filter.

The Headline object contains metadata such as the text of the headline, company stock ticker symbols, and subject codes, but does not contain the actual text of a news story. The getResourceID() method in the Headline class returns a resource identifier that is the key for retrieving the full text.

Given a NewsChannel (see earlier examples) and a ResourceID (page 116) the following simple static method returns a string containing the story text:

```
private static String
getArticle(NewsChannel channel, ResourceID resourceId)
throws IOException {
       StoryRequest request = new StoryRequest(resourceId);
       request.doReadStory(channel);
       StoryAnswer answer = new StoryAnswer(channel.receive());
       return answer.getText();
}
```

By default, story text is delivered in the XMLNews XML format. For information on how to select alternative formats (such as HTML), see “How to Obtain the Content of a Story,” page 42.

## Code Walk-Through

First, create a new gari.news.StoryRequest object (page 120) with the story's unique resource identifier:

```
StoryRequest request = new StoryRequest(resourceId);
```

Next, use the request's doReadStory method to read the story text into a buffer in the news channel:

```
request.doReadStory(channel);
```

(This throws a java.io.IOException if there is an error reading the story text.)

Next, create a new gari.news.StoryAnswer object (page 119) to parse the text from the news channel's buffer:

```
StoryAnswer answer = new StoryAnswer(channel.receive());
```

Finally, use the answer's getText() method to retrieve the text of the story as a string (which can then be saved to disk, passed to an XML parser, and so on):

```
return answer.getText();
```

## How to Use the Code

To use this method, first create an InapClient object as described in "Task 1: Connecting to the News Server," page 8, and then use it to create a NewsChannel:

```
InapClient client = amcConnect(USERNAME, PASSWORD, HOSTNAME);
NewsChannel channel = new NewsChannel(client);
```

Either initiate a history search, or start a real-time profile filter, as described in "Task 2: Searching Historical News" (page 9) or "Task 3: Obtaining and Filtering Real-Time News" (page 12). In either case, the application eventually ends up iterating through an enumeration of gari.news.Headline objects.

Extract the resource identifier from the Headline object:

```
ResourceID resourceId = headline.getResourceID();
```

Pass the channel and the resourceId to the getArticle method. The return value is a string containing the article (on error, the method throws a java.io.Ioexception):

```
String article = getArticle(channel, resourceId);
```

## Example Application

For a complete, short Java command-line application to retrieve the full text of an article, see the source code listing "Listing: Retrieving the Full Text of an Article," page 136.

# Chapter 3: How To ...

This chapter provides brief explanations and code snippets for common tasks. It does not provide detailed code walk-throughs as in the “Quick Start” chapter, so you may wish to refer to the “API” chapter (page 45) for additional information.

This chapter covers the following common tasks:

- "How to Obtain the Provider of a Story" (page 38)
- "How to Obtain the Service of a Story" (page 38)
- "How to Obtain the Publication Date/Time of a Story" (page 39)
- "How to Obtain the Short Identifier for a Story" (page 40)
- "How to Obtain the Headline Text of a Story" (page 41)
- "How to List the Available Wires" (page 41)

## Connecting and Authentication

Creating a connection to the NewsEdge News Server is the basis for all other interaction with the server. This section describes how to make a basic connection to the server, how to connect through a proxy, and how to connect through port 80 and 443 (for firewall traversal).

In addition to creating a connection to the News Server, there are several News Server management tasks you may perform, including changing your connection password and monitoring the connection's health.

Note that you can have only one connection with the News Server active at a time for each account. Creating a new connection closes an existing one.

### How to Connect to the Server

#### *Synopsis*

```
LoginData login = new LoginData(username, password, "JAPI");
InetAddress address = InetAddress.getByName(hostname);
client = new InapClient(address, Channel.STARTUP_SERVICE, login);
NewsChannel channel = new NewsChannel(client);
```

### Prerequisites (all available from NewsEdge Customer Support)

Hostname of News Server

Username

Password

### Classes Used

gari.inap.LoginData (page 48)

gari.net.Channel (page 76)

gari.inap.InapClient (page 46)

gari.net.QuipException (page 83)

gari.net.TimeoutException (page 84)

gari.news.NewsChannel (page 102)

java.io.IOException

java.net.InetAddress

### Notes

The InapClient constructor requires an IP address (java.net.InetAddress), not a string or hostname. The third argument to the InapClient constructor does not currently matter – you may want to use a combination of "JAPI" and your company name as a convention (for example, "JAPIAcmeFinance").

The method throws QuipException or IOException. The client can distinguish TimeoutException as a subclass of QuipException if there is an error. The TimeoutException indicates that the library could not connect within time limits.

By default, the method uses TCP port 6973. Channel.StartupService and Channel.NewsService are identical services. For information on connecting through other ports, see "How to Connect Through Port 80 or Port 443," page 22.

The client application performs most of its operations through a NewsChannel  object. However, in addition to creating a news channel, the client application also uses the InapClient object to change its password (see "How to Change Your Connection Password," page 23) and to set up observers to monitor the connection health (see "How to Monitor the Connection's Health," page 24). Users can create multiple channels, if there is only one underlying InapClient object.

For an introductory walk-through of connecting to the News Server, refer to "Task 1: Connecting to the News Server," page 8 in the "Quick Start" chapter.

For information on how to connect through an Internet proxy, see "How to Connect Through a Proxy," page 21.

## How to Connect Through a Proxy

### *Synopsis*

```
ProxyData proxy = new ProxyData("proxy.example.org", "8080");
client = new InapClient(address, Channel.STARTUP_SERVICE, login,
                        proxy);
```

### *Prerequisites*

Hostname of News Server

Username

Password

Hostname or IP address of your proxy

TCP port used by your proxy (e.g., 8080)

Username and password, if required by your proxy

### *Classes Used*

gari.net.ProxyData (page 77)

gari.inap.InapClient (page 46)

gari.util.detectProxy (optional) (page 129)

### *Notes*

The gari.util.ProxyData class allows you to configure proxy data.

There is also a constructor that accepts a username and password (see “ProxyData Class,” page 77), if your proxy is password-protected:

```
ProxyData proxy = new ProxyData("proxy.example.org", "8080",
       "username", "password");
```

The gari.inap.InapClient class has a constructor that accepts the proxy data as a fourth argument (see “Task 1: Connecting to the News Server,” page 8 and “How to Connect to the Server,” page 19).

If you are working with JDK 1.3 or 1.4, you can use the gari.util.detectProxy class to auto-detect proxies that are not password protected. Use any well-known HTTP URL:

```
detectProxy proxyTest = new detectProxy(new
       URL("http://www.example.org"));
```

The isProxySet method indicates whether a proxy was detected. The getProxyHost and getProxyPort return values that can be passed to the ProxyData constructor. This method is designed for use mainly by Java applets.

## How to Connect Through Port 80 or Port 443

### *Synopsis*

```
InapClient client = InapClient(address, Channel.HTTP_SERVICE,
       login);
```

Prerequisites:

Hostname of News Server

Username

Password

### Classes Used

gari.inap.LoginData (page 48)

gari.net.Channel (page 76)

gari.inap.InapClient (page 46)

### Notes

Connecting to the news server through port 80 using "Channel.HTTP_SERVICE" as the service and TCP port 80, or through port 443 using "Channel.HTTPS_SERVICE" as the service and TCP port 443, often allows connections through corporate firewalls, which might block outgoing connections to the default port (6973) on the news server.

While HTTPS_SERVICE uses the same port as secure, encrypted SSL/TLS connections for web sites – to take advantage of open ports in firewalls – it does not actually use SSL/TLS. There is no difference in encryption or security using HTTP_SERVICE or HTTPS_SERVICE.

## How to Change Your Connection Password

### Synopsis

```
LoginData newLogin = new LoginData(USERNAME, OLD_PASSWORD,
                                   "JAPI", NEW_PASSWORD);
client.change_password(newLogin);
```

### Prerequisites

An InapClient  object (see "How to Connect to the Server," page 19

Your existing username and password

Your new password

### *Classes Used*

gari.inap.LoginData (page 48)

gari.inap.InapClient (page 46)

gari.net.QuipException (page 83)

### *Notes*

Create a Login object as for the connection, using the old password as the second argument to the constructor, but adding the new password as the fourth argument.

The method throws a java.io.IOException or a gari.net.QuipException if the change is unsuccessful.

## How to Monitor the Connection's Health

### *Synopsis (main code)*

```
client.addQuipEventListener(new MyListener());
```

### *Synopsis (listener)*

```
class MyListener implements QuipEventListener
{
       public void QuipEventNotification (QuipEvent event)
       {
          System.err.println("Event: " + event.getType() + " (" +
                              event.getMessage() + ")");
       }
}
```

### *Prerequisites*

An InapClient object (see “How to Connect to the Server,” page 19)

### *Classes Used*

gari.inap.InapClient (page 46)

gari.net.QuipEventListener (interface) (page 75)

gari.net.QuipEvent (page 81)

### *Notes*

The library can inform the client application of major events in the connection, such as a disconnect or a system error. Whenever there is a connection event, the library invokes the listener's QuipEventNotification method to report it. (See "QuipEvent Class, Methods," page 81 for a list of event types and methods available.)

While it is possible to use this interface for rudimentary logging of events, it is generally better for client applications to configure the library's own logging support through the Apache Foundation Log4J library (see "How to Configure Logging", page 25).

## How to Configure Logging

### *Synopsis (Property File)*

```
log4j.rootLogger=DEBUG, A1
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-4r [%t] %-5p %c %x -
      %m%n
log4j.logger.demo=INFO
```

### *Prerequisites*

File named log4j.properties in the root class directory of the client application

### *Notes*

The NewsEdge Java API uses the Apache Foundation Log4J logging library to make tracing, management, and debugging information available to users. Log4J uses six different logging levels, from least to most severe:

1. TRACE
2. DEBUG
3. INFO
4. WARN
5. ERROR
6. FATAL

As is usual with Log4J, loggers are named after the packages and classes in the library. Adding the following line to the log4j.properties file, for example, disables all logging messages below the ERROR level for the entire library (every class in a package beginning with "gari"):

```
log4j.logger.gari=ERROR
```

The following line enables all log messages for the gari.net package (since TRACE is the lowest level):

```
log4j.logger.gari.net=TRACE
```

There are other ways for an application to configure logging beyond the use of a properties file. For more information, see the Log4J web site http://logging.apache.org/log4j/

## How to Discover Your Entitlements

### *Synopsis*

```
UserNewsInfoRequest.doUserNewsInfoRequest(channel);
UserNewsInfo info = new UserNewsInfo(channel.receive());
```

### *Prerequisites*

A NewsChannel object (see "How to Connect to the Server," page 19)

### Classes Used

gari.news.NewsChannel (page 102)

gari.news.UserNewsInfoRequest (page 124)

gari.news.UserNewsInfo (page 122)

gari.news.ParsingException (page 127)

### Notes

The UserNewsInfo constructor (page 122) throws a ParsingException (page 127) if the result is scrambled. It has methods to get entitlement information such as the update frequency, maximum profiles, and so on.

## Historical News

Historical news refers to past articles retained on the NewsEdge News Server. This section describes how to search historical headlines, how to change the number of search results returned and how to limit results by date. For additional information on searching historical news, see "Task 2: Searching Historical News," page 9.

## How to Search Historical Headlines

### Synopsis

```
HistorySearch search = new HistorySearch("*");
search.doReadHeadlines(channel);
HeadlineAnswer answer = new HeadlineAnswer(channel.receive());
ExtendedEnumeration e = answer.getElements();
while (e.hasMoreElements()) {
       Headline headline = (Headline)e.nextElement();
       // process the Headline object
}
```

### Prerequisites

A full-text search pattern for the stories you want to find (explained below)

A NewsChannel object (see "Task 1: Connecting to the News Server," page 8)

### Classes Used

gari.news.NewsChannel (page 102)

gari.news.HistorySearch (page 95)

gari.news.HeadlineAnswer (page 89)

gari.util.ExtendedEnumeration (page 128)

gari.news.Headline (page 85)

### Notes

The client application can specify a pattern for a history search. The following table describes the pattern matching symbols:

| Pattern | Description |
|---|---|
| * | Match anything. |
| word1 word2 | Match word1 or word2. |
| +word | The specified word is required in the results. |
| -word | The specified word is not allowed in the results. |
| /symbol | Searches for a stock ticker symbol (e.g., "/MSFT")<br>NOTE: Use only the symbol. Do not include the exchange. |
| "a phrase" | Match the words within the quotation marks as a phrase. |

To search for stories containing the words "oil" or "refinery" anywhere, use the following:

```
HistorySearch search = new HistorySearch("oil refinery");
```

To search for stories containing the words “oil” and “refinery”, use the following:

```
HistorySearch search = new HistorySearch("+oil +refinery");
```

To search for stories containing the words “oil” but not “refinery”, use the following:

```
HistorySearch search = new HistorySearch("+oil -refinery");
```

To search for the exact phrase “oil refinery” use the following:

```
HistorySearch search = new HistorySearch("\"oil refinery\"");
```

To search for stories about Exxon (ticker symbol “XOM” on the NYSE), use the following:

```
HistorySearch search = new HistorySearch("/XOM");
```

ExtendedEnumeration is derived from the regular java.util.Enumeration but adds a count() method. The client application can treat it as a regular enumeration.

## How to Change the Number of Search Results Returned

### *Synopsis*

```
HistorySearch search = new HistorySearch("*");
search.setMaxHits(2);
search.setGetExactHitCount(true);
HeadlineAnswer answer = new HeadlineAnswer(channel.receive());
ExtendedEnumeration e = answer.getElements();
while (e.hasMoreElements()) {
       Headline headline = (Headline)e.nextElement();
      // process the Headline object
}
```

### *Prerequisites*

A full-text search pattern for the stories you want to find (explained below)

A NewsChannel object (see “Task 1: Connecting to the News Server,” page 8)

### *Classes Used*

gari.news.HistorySearch (page 95)

### *Notes*

The setGetExactHitCount method call ensures that no more than the maximum number of search items specified are returned. If you do not specify true for the setGetExactHitCount method, the actual number of results returned may be slightly higher than the value you specify.

For information on how to page through results, see “HeadlineContext Class,” page 90.

## How to Limit Search Results by Date

### *Synopsis*

```
HistorySearch search = new HistorySearch("*");
search.setDateRange(">20070101 00:00 <20070131 23:59");
HeadlineAnswer answer = new HeadlineAnswer(channel.receive());
ExtendedEnumeration e = answer.getElements();
while (e.hasMoreElements()) {
       Headline headline = (Headline)e.nextElement();
      // process the Headline object
}
```

### *Prerequisites*

A full-text search pattern for the stories you want to find (explained below)

A NewsChannel object (see “Task 1: Connecting to the News Server,” page 8)

### Classes Used

gari.news.HistorySearch (page 95)

### Notes

The example searches for articles published in January 2007. The format is "<YYYYMMDD" for articles published before or on a date, ">YYYYMMDD" for articles published on or after a date, or both combined for a range ">YYYYMMDD <YYYYMMDD".

You also may add time in "HH:SS" format after the date (with a space in between the date and time) to specify a time in 24-hour format, using the U.S. Eastern time zone.

To search for articles published on or after January 1, 2007, use the following:

```
search.setDateRange(">20070101");
```

To search for articles published on or before 5:00 pm (Eastern) on March 1, 2007, use the following:

```
search.setDateRange("<20070301 17:00");
```

To search for articles published on or after 5:00 pm (Eastern) on February 28, 2007, use the following:

```
search.setDateRange(">20070228 17:00");
```

To search for articles published between 5:00 pm (Eastern) on February 28 and 5:00 pm on March 1, 2007, use the following:

```
search.setDateRange(">20070228 17:00 <20070301 17:00");
```

For more information on search patterns, see "How to Search Historical Headlines, Notes," page 28.

## How to Limit a Search by Wires

### Synopsis #1 (history search)

```
search.setWires("[ BW PR");
search.doReadHeadlines(channel);
```

```
HeadlineAnswer answer = new HeadlineAnswer(channel.receive());
```

### *Synopsis #2 (real-time news)*

```
profile.setWires("[ BW PR");
ProfileManager manager = new ProfileManager(profile, channel);
manager.addObserver(new MyObserver());
```

Note: See "How to List the Available Wires," page 41.

### *Prerequisites*

In order to limit a search by wires, you require the two-letter abbreviation for the wires you want to include or exclude. See "How to List the Available Wires," page 41.

### *Classes Used*

gari.news.MetaHeadlineQuery (page 99, and the base class for gari.news.Profile and gari.news.HistorySearch)

### *Notes*

The MetaHeadlineQuery is the common base class for Profile and HistorySearch. This technique works for both historical and real-time news.

Before executing a history search, or starting a real-time profile, you can use the setWires method (page 100) to select which wires are included or excluded.

The synopses use the two-letter wire identifiers available through WireInfo getBaseWire method (page 125), not the eight-character provider/service identifiers available through gari.news.ResourceID (page 116).

The list of available wires is part of entitlement information (see "How to Discover Your Entitlements," page 26 and "How to List the Available Wires," page 41).

The argument to the method is a string beginning with "[" (include) or "]" (exclude), followed by a list (which may or may not be space-separated) of two-letter identifiers. A blank string means include all wires.

The following are examples of the argument:

"" — include all wires

"[ PR" — include only PR Newswire

"] PR" — include everything except PR Newswire

"[ PR BW DJ" — include only PR Newswire, Business Wire, and Dow Jones

"] PR BW DJ" — include everything except PR Newswire, and Dow Jones

## Real-Time News

Real-time news refers to news stories as they arrive from the providers. This section describes how to filter real news as it arrives. The Moody's Analytics contract or product you are using determines what news is available to you. It is possible to backfill a feed by including recent stories as well as newly- arrived ones.

### How to Start a Real-Time News Feed

#### *Synopsis #1 (main client code)*

```
Profile profile = new Profile("Test", "*");
// will overwrite any existing profile in this slot ...
profile.setNumber(1);
profile.setBackfill(true);
ProfileManager manager = new ProfileManager(profile, channel);
manager.addObserver(new MyObserver());
```

#### *Synopsis #2 (observer class)*

```
public class MyObserver implements Observer {
      public void update (Observable o, Object arg)
```

```
        {
                ProfileHeadlines pheadlines = (ProfileHeadlines)arg;
                HeadlineAnswer answer = pheadlines.getHeadlines();
                ExtendedEnumeration e = answer.getElements();
                while (e.hasMoreElements()) {
                        Headline headline = (Headline)e.nextElement();
                        // process headline
         }
}
```

### Prerequisites

A NewsChannel object (see “Task 1: Connecting to the News Server,” page 8)

### Classes Used

gari.news.NewsChannel (page 102)

gari.news.Profile (page 103)

gari.news.ProfileManager (page 111)

gari.news.ProfileHeadlines (page 110)

gari.news.HeadlineAnswer (page 89)

gari.news.Headline (page 85)

gari.util.ExtendedEnumeration (page 128)

java.util.Observer (interface)

java.util.Observable (interface)

### Notes

Use any profile name that is meaningful to the client application. (This example uses “Test”). The library sends incoming stories to the observer class asynchronously: the rest of the application can continue running.

The client application can discover the maximum number of profile slots available through the UserInfo class (see “How to Discover Your Entitlements,” page 26).

Note that uploading a profile to the server overwrites any existing profile in the slot. If you have an existing profile, change the argument in setNumber to use a different slot.

For additional information, see “Task 3: Obtaining and Filtering Real-Time News,” page 12.

## How to Filter Real-Time Headlines

### *Synopsis*

```
Profile profile = new Profile("Test", "oil refinery");
```

### *Prerequisites*

A NewsChannel object (see “Task 1: Connecting to the News Server,” page 8)

### *Classes Used*

gari.news.Profile (page 103)

### *Notes*

Filtering real-time headlines uses the same patterns as the history search and returns only articles that match the search pattern (words, phrases, ticker symbols, and so on). For more information on the history search, see “Historical News,” page 27.

## How to Examine your Profiles

### *Synopsis*

```
ProfileGroupRequest.doReadProfileGroup(channel);
ProfileGroup group = new ProfileGroup(channel.receive());

ExtendedEnumeration e = group.getElements();
while (e.hasMoreElements()) {
       Profile profile = (Profile)e.nextElement();
      // process the profile
}
```

### *Prerequisites*

A NewsChannel object (see "Task 1: Connecting to the News Server," page 8)

### *Classes Used*

gari.news.NewsChannel (page 102)

gari.news.ProfileGroupRequest (page 109)

gari.news.ProfileGroup (page 107)

gari.util.ExtendedEnumeration (page 128)

gari.news.Profile (page 103)

### *Notes*

Once you have each profile, you can use getNumber() to find the profile's number, or getName() to find the name you assigned when you created the profile. You can use these classes to find a profile you have already created, or to look for an empty slot for a new profile.

## How to Filter Articles by Wire

See "How to Limit a Search by Wires," page 31.

# Metadata and Content

"Metadata" is information about news stories (and other stories), while "content" is the actual news content.

The primary access point for a story is the gari.news.Headline object (see "Historical News," page 27 and "Real-Time News," page 33 for information on how to get Headline objects for historical or real-time stories).

Depending on the delivery format you choose, much more information may be available through the XML metadata accompanying the story than is covered here (see the appropriate format guides for details). This guide describes metadata available only directly through API calls.

## How to Obtain a Story's Subject Codes or Stock Ticker Symbols

### *Synopsis*

```
String codeString = headline.getCodes();
String symbolString = headline.getSymbols();
```

### *Prerequisites*

A Headline object with information about a story (see "Historical News," page 27 and "Real-Time News," page 33)

### *Classes Used*

gari.news.Headline (page 85)

### *Notes*

The codeString is like the following (only typically much longer):

```
IC/comp;IC/publ;NI/Computer_Systems;NI/Info_Services
```

Each entry consists of the vocabulary type (such as "IC" for "industry code") followed by a forward slash ("/"), followed by the code (such as "comp" for computing). Entries are semi-colon-separated.

The symbolString looks similar, but in this case each entry consists of an abbreviation for the exchange, followed by a colon (":"), followed by a company ticker symbol:

```
NYSE:PDT;NYSE:MFC
```

It is easy to split either string into a string array using the standard java.lang.String split method with the argument ";":

```
String codes[] = codeString.split(";");
String symbols[] = symbolString.split(";");
```

The same method works for splitting an entry, using “/” or “:” as the argument:

```
String symbolEntry[] = symbol.split(“/”);
```

## How to Obtain the Provider of a Story

### *Synopsis*

```
ResourceID id = headline.getResourceID();
String providerCode = id.getProvider();
```

### *Prerequisites*

A Headline object with information about a story item (see “Historical News,” page 27 and “Real-Time News,” page 33)

### *Classes Used*

gari.news.Headline (page 85)

gari.news.ResourceID (page 116)

### *Notes*

The name of the provider may also be available in the XML metadata (see “How to List the Available Wires,” page 41).

## How to Obtain the Service of a Story

### *Synopsis*

```
ResourceID id = headline.getResourceID();
String service = id.getService();
```

### *Prerequisites*

A Headline object with information about a story (see "Historical News," page 27 and "Real-Time News," page 33)

### *Classes Used*

gari.news.Headline (page 85)

gari.news.ResourceID (page 116)

### *Notes*

News providers may offer multiple services and products, such as "USPR____" for "U.S. Press Releases".

The service may also be available in the XML metadata (see "How to List the Available Wires," page 41).

## How to Obtain the Publication Date/Time of a Story

### *Synopsis*

```
ResourceID id = headline.getResourceID();
String date = id.getDate();
String time = id.getTime();
```

### *Prerequisites*

A Headline object with information about a story (see "Historical News," page 27 and "Real-Time News," page 33)

### *Classes Used*

gari.news.Headline (page 85)

gari.news.ResourceID (page 116)

### *Notes*

Note that this is the official publication date of the story. It stays the same for all releases, even if subsequent releases come out on a different date.

The date is in YYYYMMDD format (for example "20070227" for February 27, 2007). The time is in HHMM format (for example, "1023" for 10:23 EST or 10:23 EDT during daylight savings time).

Depending on your news format, more date information is available through the XML metadata accompanying the news story content. See the format guides for details.

## How to Obtain the Short Identifier for a Story

### *Synopsis*

```
ResourceID id = headline.getResourceID();
String ident = id.getID();
```

### *Prerequisites*

A Headline object with information about a story (see "Historical News," page 27 and "Real-Time News," page 33)

### *Classes Used*

gari.news.Headline (page 85)

gari.news.ResourceID (page 116)

### *Notes*

Using digits, characters and underscores, the short identifier is guaranteed to be unique for the date/time/provider/service combination (for example, "_BW5880"). The length of the identifier is not fixed. This is sometimes (but not always) the same as the provider's slug for the story.

## How to Obtain the Headline Text of a Story

### Synopsis

```
String text = headline.getText();
```

### Prerequisites

A Headline object with information about a story (see "Historical News," page 27 and "Real-Time News," page 33)

### Classes Used

gari.news.Headline (page 85)

### Notes

Note that this is the display headline (for example, "Avigen to Present At Oppenheimer Pain Management Investor Conference") for use in a list of headlines, search results, and so on. The display headline may be shorter than the full headline included in the story content.

## How to List the Available Wires

### Synopsis

```
UserNewsInfo info = getInfo(channel);
WireList wires = info.getWireList();
Iterator it = wires.iterator();
while (it.hasNext()) {
       WireInfo wire = (WireInfo)it.next();
       String id = wire.getBaseWire();
       String provider = wire.getProviderService().substring(0,
       8);
       String service =
       wire.getProviderService().substring(8, 8);
    // etc.
      }
```

### *Classes Used*

gari.news.NewsChannel (page 102)

gari.news.UserNewsInfo (page 122)

gari.news.WireList (page 126)

gari.news.WireInfo (page 124)

java.util.Interator

### *Notes*

One feature of entitlements is a list of available wires. The WireInfo object (page 124) contains information about one source. Also refer to “How to Discover Your Entitlements,” page 26.

The getBaseWire returns a two-letter code for a wire, which can be used for limiting search results (see “How to Limit a Search by Wires,” page 31.

The getProviderService returns the eight-character provider id and the eight-character service id concatenated into a 16-character string.

Note that there is one WireInfo object for every provider/service combination, but the two-letter code is for provider only. For example, if Dow Jones has 10 services, there will be 10 WireInfo objects all with the two-letter code “DJ” (as returned by getBaseWire), but different services in getProviderService.

## How to Obtain the Content of a Story

### *Synopsis*

```
StoryRequest request = new StoryRequest(headline.getResourceID());
request.setStyleSheet("XMLNEWS");
request.doReadStory(channel);
StoryAnswer answer = new StoryAnswer(channel.receive());
```

```
String content = answer.getText();
```

### Prerequisites

A Headline object with information about a story (see "Historical News," page 27 and "Real-Time News," page 33)

### Information Required

(Optional) style sheet name for desired format (such as "HTML" or "XMLNEWS")

### Classes Used

gari.news.NewsChannel (page 102)

gari.news.Headline (page 85)

gari.news.ResourceID (page 116)

gari.news.StoryRequest (page 120)

gari.news.StoryAnswer (page 119)

### Notes

"Content" is the actual story. This section describes how to obtain the content of a story.

For additional information, see the code walk-through in "Task 4: Retrieving the Full Text of an Article," page 16.

Specifying the style sheet (format) for the story is optional. The default is "XMLNEWS":

```
    request.setStyleSheet("XMLNEWS");
```

The following is a list of available formats:

XMLNEWS (default) – see http://www.xmlnews.org/

HTML – see http://www.w3.org/Markup/

TEXT

NEWSML (NEWSML with embedded XHTML) – see http://www.newsml.org/

NITF (standalone NITF) – see http://www.nitf.org/

# Chapter 4: API

This chapter contains reference documentation for the NewsEdge library, concentrating on the classes, interfaces, constructors, methods and constants that client applications normally use. It does NOT document parts of the library that are deprecated or designed mainly for internal use.

The NewsEdge library includes the following packages:

gari.inap Package (higher-level networking support [transaction layer]), page 46

gari.media Package (API for the media server), page 50

gari.media.iptc Package (advanced image metadata), page 59

gari.net Package (lower-level networking support [messaging layer]), page 75

gari.news Package (main application layer), page 85

gari.util package (utility classes and methods), page 127

This chapter lists the classes in alphabetical order within each package. For simplicity, the examples in the text usually omit package names for classes and interfaces in the same package as the one being documented and for the package name in java.lang package.

For an overview of the structure of the NewsEdge library, including information on the network stack and messaging patterns, see “Architectural Overview” (page 139).

The classes use the Apache Foundation Log4J library for logging messages of different severities. You can configure logging separately for each package or even for each class. Note that the log4j.jar file must be on the client application's class path or the library methods will fail. There should also be a log4j.properties file in the root class directory, unless the application is configuring logging through some other mechanism. For more information, see “How to Configure Logging” (page 25).

## gari.inap Package

The gari.inap package contains the transaction networking layer for connecting to the NewsEdge News Server. The message/packet layer is in gari.net (page 75) and is mostly invisible to the client application.

The classes of interest to client applications are InapClient, which encapsulates the connection to the news server, and LoginData, which holds authentication information for making a connection.

## InapClient Class

#### *Synopsis*

```
LoginData login = new LoginData(username, password, "JAPI");
Java.net.InetAddress address =
                      InetAddress.getByName(hostname);
InapClient client = new InapClient(address, login);
```

#### *Notes*

This class represents a connection to the news server. You need to create an InapClient object before you can perform most other actions (see “Task 1: Connecting to the News Server,” page 8). Note that the classes in the gari.media package (page 50) do not use InapClient.

This class allows client applications to log in, manage the connection, register listeners for events, and so on. It also contains the connection state that is passed to other objects. InapClient is a sub-class of QuipClient (page 79) and most of its methods are defined there.

#### *Constructors*

InapClient (java.net.InetAddress serverAddress, LoginData login)
throws java.io.IOException, gari.net.QuipException

```
InapClient (java.net.InetAddress serverAddress, LoginData login,
            gari.net.ProxyData proxy)
    throws java.io.IOException, gari.net.QuipException

InapClient (java.net.InetAddress serverAddress, short port,
            LoginData login, gari.net.ProxyData proxy)
    throws java.io.IOException, gari.net.QuipException
```

The first two examples are convenience constructors, while the third one is the full form.

The serverAddress parameter is the IP address of the news server. The port parameter is the TCP port for connecting to the server. The default port is 6973, STARTUP_SERVICE. There are constants for the port numbers available in the gari.net.Channel class (page 77):

gari.net.Channel.ADMIN_SERVICE – 6963

gari.net.Channel.HTTP_SERVICE – 80

gari.net.Channel.HTTPS_SERVICE – 443

gari.net.Channel.NEWS_SERVICE – 6973

gari.net.Channel.QUOTES_SERVICE – 6983

gari.net.Channel.STARTUP_SERVICE – 6973

Many firewalls block most ports but allow outgoing connections through port 80 or 443. In such a case, use HTTP_SERVICE or HTTPS_SERVICE(see “How to Connect Through Port 80,” page 22).

The login parameter contains the user name and password (see “LoginData Class,” page 48).

The proxy parameter contains information for clients that connect to the Internet through a proxy server (see “ProxyData Class,” page 77 and “How to Connect Through a Proxy,” on page 21).

### *Methods*

See also the methods inherited from gari.net.QuipClient (page 79).

short changePassword (LoginData login)
throws java.io.IOException. gari.net.QuipException, gari.net.TimeoutException

Use this method to set a new password for connecting to the news server.

This method throws a java.io.IOException for a general networking error, a gari.net.TimeoutException if the server does not respond or a gari.net.QuipException for any other errors.

The LoginData object must contain your current password. The new password string is a separate field (see "LoginData Class," page 48), passed as the optional fourth parameter to the LoginData constructor.

### *Constants*

LOGIN_TIMEOUT

The number of milliseconds before a login attempt times out. This is hard-coded to 20,000 (i.e., 20 seconds).

## LoginData Class

### *Synopsis*

```
LoginData login = new LoginData(username, password, "JAPI");
```

### *Notes*

This class encapsulates authentication information for connecting to the server or changing the password. The third argument (version) in the constructor does not really matter, since it is used just for tracking on the server side. The examples use “JAPI”. You may want to use a combination of “JAPI” and your company name as a convention (for example, “JAPIAcmeFinance”).

### Constructors

LoginData ()

Default constructor to create an empty object. The client application must use the set* methods (page 49) to specify values.

LoginData (String username, String password, String version)

Convenience constructor for creating an object to authenticate with the server through an InapClient (page 46).

LoginData (String username, String password, String version, String new_password)

Convenience constructor for creating an object to change the password through the InapClient changePassword method (page 48).

The username is assigned by NewsEdge Customer Support. The password is initially assigned by NewsEdge Customer Support, but the client may choose to change it. Version is not currently used, set it to “JAPI”, or a combination of “JAPI” and your company name (e.g., “JAPIAcmeFinance”).

Use new_password only when calling the InapClient changePassword method.

### Methods

void setUsername (String userName)

String getUsername ()

Setter and accessor for the username value.

void setPassword (String password)

String getPassword ()

Setter and accessor for the password value.

void setNewPassword (String newPassword)

Setter and accessor for the new password (when using InapClient.changePassword).

void setVersionString (String version)

String getVersion ()

Setter and accessor for the version string (the third argument to the constructor). The value does not currently matter. Use “JAPI”, or a combination of “JAPI” and your company name (e.g., “JAPIAcmeFinance”).

## gari.media Package

The gari.media package contains the API for the NewsEdge News Server, which allows client applications to download photos and other multimedia resources associated with stories. This package does not use a gari.inap.InapClient or gari.news.NewsChannel, but connects directly to the media server, using its own network stack (see “Architectural Overview,” page 139).

### MediaCommConstants Interface

The interface consists only of constants, inherited by MediaConnection (page 52).

#### *Constants*

MEDIAD_THUMBNAIL_RES

The image is in very low resolution, suitable for a thumbnail (100x100 pixels). The longest dimension will be scaled to 100 pixels. The smaller side will be adjusted accordingly, while preserving the aspect ratio of the image.

MEDIAD_MIDSIZE_RES

The image is in a resolution slightly larger than a thumbnail (200x200 pixels). The longest dimension will be scaled to 200 pixels.  The smaller side will be adjusted accordingly, while preserving the aspect ratio of the image.

MEDIAD_PREVIEW_RES

The image is in a preview resolution, where details are visible (400x400 pixels). The longest dimension will be scaled to 400 pixels.  The smaller side will be adjusted accordingly while preserving the aspect ratio of the image.

MEDIAD_HALFSIZE_RES

The image is at half the original resolution.

MEDIAD_ORIGINAL_RES

The image is at its original resolution.

MEDIAD_CUSTOM_RES

The image is in a custom resolution, using dimensions supplied by the client application.

MEDIAD_ASSOCIATED

Specify to retrieve a related file.  For example, use this to retrieve the resource file (fork file) for an image.

NO_SIZE_CAP

Do not limit the size of an image returned by the media server. This constant is used by the MediaConnection  requestImage method (page 55).

ORIGINAL_DPI

Use an image's original dots-per-inch resolution, instead of specifying a custom one. This constant is used by the MediaConnection requestImage method (page 55).

PREDEFINED_GEOMETRY

Use an image's own default geometry (in place of a geometry string like “400x300”). This constant is used by the MediaConnection requestImage method (page 55).

## MediaConnection Class

### *Synopsis*

```
MediaConnection connection = new MediaConnection(hostName);
connection.requestImage(key);
MediaData image = connection.readImageResponse();
```

### *Notes*

The MediaConnection class represents a persistent connection to the media server. It inherits constants from the MediaCommConstants interface (page 50).

Normally, the client application should read media objects from the media server using a gari.news.ResourceID (page 116), a string key, which may be a string version of a resource ID, or a media object file name (see getName page 58). Note that each media item has its own resourceID.

### *Constructors*

MediaConnection (String hostName)
throws MediaException, java.io.IOException

MediaConnection (String hostName, gari.net.ProxyData)
throws MediaException, java.io.IOException

MediaConnection (java.net.InetAddress address)
throws MediaException, java.io.IOException

MediaConnection (java.net.InetAddress address, gari.net.ProxyData)
throws MediaException, java.io.IOException

The constructor opens a connection to the media server.

You can specify the media server by host name (as a string), or by IP address (using a java.net.InetAddress object). You can also include optional proxy data (see "ProxyData Class," page 77 for more information).

The constructor throws a MediaException or IOException if there is trouble connecting.

Use the isConnected method (page 57) to verify the connection.

### *Methods*

void close ()
throws java.io.IOException

Close the connection to the media server. Client applications should invoke this method when they are finished with the connection.

Throws a java.io.IOException if there is a network error closing the connection.

int getTimeout ()

void setTimeout (int timeout)

Accessor and setter for the timeout property.

If the timeout is "0" (the default), the library will block indefinitely, waiting for a response from the media server. If the timeout is greater than "0", the library will wait for timeout milliseconds for a response, then throw a MediaException () if none is received.

void requestFilename (gari.news.ResourceID resourceID)
throws java.io.IOException

void requestFilename (String key)
throws java.io.IOException

Request the media server to send the filename of an image. This is the file name under which it should be saved to disk. The XML metadata accompanying the news story will contain the filename or resourceID of the associated image (see the delivery format guides for more information).

Use the readFilenameResponse method (page 56) to get the filename.

The resourceID is a gari.news.ResourceID object used as a unique identifier for media items. Each media item has its own resourceID.

The key is either a string version of a resource id or the filename of a media object.

The filename is also returned as part of the MediaData object (see requestImage, page 55 and readImageResponse, page 57), so normally client applications do not need to call one of these methods separately.

Both methods throw an IOException if there is an error reading the filename.

void requestIPTC (String key)
throws java.io.IOException

Request the IPTC profile for an image (see “IPTCProfile Class,” page 61) for information on IPTC extended image metadata).

Use the readIPTCResponse method (page 56) to read the IPTC profile.

The key is either a string version of a resource id or the filename of a media object.

Throws an IOException if there is an error reading the profile.

```
byte [] requestImage (String key, int resolution)
    throws java.io.IOException
```

```
byte [] requestImage (String key, int resolution, String geometry,
                      int dpi, int maxsize)
    throws java.io.IOException
```

Send a request for an image to the media server. Use the readImageResponse (page 57) method to read the image as a MediaData object.

The first version is for use with anything but MEDIAD_CUSTOM_RES as the resolution argument. The second version allows the client application to supply custom resolution information.

The return value is always null. The client application should ignore it.

The key parameter is either a string version of a resource id or the filename of a media object.

The resolution parameter is one of the MEDIAD_*_RES constants or the MEDIAD_ASSOCIATED constant from the MediaCommConstants interface (page 50).

The geometry parameter is a string describing the image size, in the format widthxheight (e.g., "400x300" for 400 pixels wide and 300 pixels high). It is used only if the resolution is MEDIAD_CUSTOM_RES. (Note that you can also use the MediaCommConstants constant PREDEFINED_GEOMETRY to use the image's existing geometry. PREDEFINED_GEOMETRY, page 52, is the default.)

The dpi parameter is the image resolution in dots per inch, typically, 72 for onscreen/web display, or 300+ for print. This value is used only if the resolution is MEDIAD_CUSTOM_RES. (Note that you can also use the MediaCommConstants constant ORIGINAL_DPI to use the image's existing dpi. ORIGINAL_DPI, page 52, is the default.)

The maxsize parameter is the maximum size of the returned image, in octets (bytes). This value is used only if the resolution is MEDIAD_CUSTOM_RES. (Note that you can also use the MediaCommConstants constant NO_SIZE_CAP to avoid any restriction on the size of the returned image. NO_SIZE_CAP, page 51, is the default.)

gari.media.iptc.IPTCProfile readIPTCResponse ()
throws MediaException, gari.media.iptc.IPTCParseException, java.io.IOException

Returns the IPTC metadata profile for an image (see "IPTCProfile Class," page 61 for more information).

The application must use the requestIPTC method (page 54) first to request the profile from the media server.

The method throws an IOException if there is a networking error, a MediaException if there is an error in the message format, or an IPTCParseException if there is an error in the IPTC profile data itself.

String readFilenameResponse ()
throws MediaException, java.io.IOException

Returns the recommended file name for an image. The application client must use the requestFilename method (page 54) first to request the filename from the media server.

The filename is also contained in the MediaData object returned by readImageResponse (page 57), so the client does not have to request it separately.

The method throws an IOException if there is a networking error, or a MediaException if there is an error in the data returned by the media server.

MediaData readImageResponse ()
throws MediaException , java.io.IOException

Returns a MediaData object (page 57) containing the raw data and other information about an image.

The application must use one of the requestImage methods (page 55) first to request the image from the media server.

The method throws an IOException if there is a network error, or a MediaException if there is an error in the data returned by the media server.

boolean isConnected ()

Return true if this is an active connection to a media server.

gari.net.ProxyData getProxyData ()

Get the proxy data being used to connect to the media server, or null if none was supplied.

boolean isUsingProxyData ()

Return true if the library is connecting to the media server through a proxy (i.e., if a gari.net.ProxyData object was supplied to a constructor). See also “ProxyData Class,” page 77.

## MediaData Class

### *Synopsis*

```
MediaData image = connection.readImageResponse();
String mimeType = image.getMimeType();
byte rawData[] = image.getData();
```

### Notes

This class defines an object containing raw image byte data, together with some basic metadata (such as the size and MIME type).

Also see the MediaConnection requestImage (page 55) and readImageResponse (page 57) methods. Other metadata is available through the gari.media.iptc.IPTCProfile class (page 61).

### Constructors

Client applications should not construct MediaData objects directly. The MediaConnection readImageResponse method (page 57) generates these objects for the client application.

### Methods

String getName ()

Return the recommended file name for the image. The XMLNews story can reference the image either by filename or resourceID.

See also the MediaConnection requestFilename (page 54) and readFilenameResponse (page 56) methods.

String getMimeType ()

Get the MIME type of the image, such as “image/jpeg”.

The mime type is a code describing the file format from a list maintained by the Internet Assigned Numbers Authority (IANA), such as “image/jpeg” or “application/xhtml+xml” (see http://www.iana.org/assignments/media-types/).

long getSize ()

Get the size of the image in bytes (octets).

int getResolution ()

Get the resolution of the image.

The result will be one of the MEDIAD_*_RES constants or the MEDIAD_ASSOCIATED constant from the MediaCommConstants interface (page 50).

byte [] getData ()

Get the raw byte data for the image.

The client application can create an Image object using the byte data.

### MediaException Exception

Some of the methods and constructors in this package throw a Media Exception when there is an error in the messages passed back and forth to the media server.

#### *Constructors*

Client applications do not need to construct instances of this exception themselves. It is thrown by the NewsEdge library.

#### *Methods*

This class inherits its methods from java.lang.Exception.

## gari.media.iptc Package

The gari.media.iptc package handles advanced image metadata as defined by the International Press Telecommunications Council (IPTC).

## IPTCConstants Interface

The IPTCConstants Interface provides constants for fields in the IPTC Information Interchange Model (IIM), an image metatdata standard.

These constants appear as an argument to the IPTCProfile class's getField method (page 62).

See the IIM specification for the meanings of the fields (available through http://www.iptc.org/IIM/).

### *Constants*

IPTC_ACTIONADVISED

IPTC_AUDIOOUTCUE

IPTC_AUDIOSAMPLINGRATE

IPTC_AUDIOSAMPLINGRESOLUTION

IPTC_AUDIOTYPE

IPTC_BYLINE

IPTC_BYLINETITLE

IPTC_CAPTIONABSTRACT

IPTC_CATEGORY

IPTC_CITY

IPTC_CONTACT

IPTC_CONTENTLOCATIONCODE

IPTC_CONTENTLOCATIONNAME

IPTC_COPYRIGHTNOTICE

IPTC_COUNTRYCODE

IPTC_COUNTRYNAME

IPTC_CREDIT

IPTC_DATECREATED

IPTC_DIGITALCREATIONDATE

IPTC_KEYWORDS IPTC_AUDIODURATION

IPTC_LANGUAGEIDENTIFIER

IPTC_OBJECTATTRIBUTE

IPTC_OBJECTCYCLE

IPTC_OBJECTDATEPREVIEWDATE

IPTC_OBJECTDATEPREVIEWFILEFORMAT

IPTC_OBJECTDATEPREVIEWFILEFORMATVERSION

IPTC_OBJECTNAME

IPTC_OBJECTTYPE

IPTC_ORIGINALTRANSMISSIONREFERENCE

IPTC_ORIGINATINGPROGRAM

IPTC_PROGRAMVERSION

IPTC_PROVINCESTATE

IPTC_RASTERIZEDCAPTION

IPTC_RECORDVERSION

IPTC_REFERENCEDATE

IPTC_REFERENCENUMBER

IPTC_REFERENCESERVICE

IPTC_RELEASEDATE

IPTC_RELEASETIME

IPTC_DIGITALCREATIONTIME
IPTC_EDITORIALUPDATE
IPTC_EDITSTATUS
IPTC_EXPIRATIONDATE
IPTC_EXPIRATIONTIME
IPTC_FIXTUREIDENTIFIER
IPTC_HEADLINE
IPTC_IMAGEORIENTATION
IPTC_IMAGETYPE
IPTC_SOURCE
IPTC_SPECIALINSTRUCTIONS
IPTC_SUBJECTREFERENCE
IPTC_SUBLOCATION
IPTC_SUPPLEMENTALCATEGORY
IPTC_TIMECREATED
IPTC_URGENCY
IPTC_WRITEREDITOR

## IPTCProfile Class

This class represents the IPTC IIM metadata profile for an image and includes advanced metadata. The client application obtains a copy through the gari.media.MediaConnection's requestIPTC  (page 54) and readIPTCResponse  (page 56) methods.

This class implements the IPTCConstants (page 60), so all of the IIM constants are available through it as well.

### *Constructors*

IPTCProfile  (byte [] profile)
throws IPTCParseException

Construct a new profile object from the raw bytes of an image's IIM section. Normally, client applications do not need to use this constructor, since the library builds the IPTCProfile  object for them.

This method throws an IPTCParseException  if there is an error parsing the raw byte data.

### *Methods*

byte [] getProfile ()

Get the raw byte data for the profile.

Object getField (int id)

Get the value of an IIM field.

The id parameter is one of the constants defined in the IPTCConstants interface (page 60).

The method returns the value of the field in the IPTC profile (numbers will be wrapped as Java objects). There are also convenience methods for each field type (see below).

String getFieldName (int id)

Get the name for an IIM field.

The id parameter is one of the constants defined in the IPTCConstants interface (page 60).

The return value is the display name of the field.

short getRecordVersion()

Get the version number of the IPTC IIM in use.

The return value is a short integer.

int getUrgency()

Get the editorial urgency for the object.

The return value is an integer. 1 = most, 5 = normal, 8 = least.

String[] getObjectAttribute()

Get a news code specifying the type of object (for example, forecast, obituary, press release, and so on).

The return value is an array of zero or more strings, each containing a code from a list the IPTC maintains.

String[] getSupplementalCategory()

Get additional category information from a list maintained by the provider (also see getCategory, page 66).

The return value is an array of zero or more strings, each containing a code for one supplemental category.

String[] getKeywords()

Get keywords associated with the object (for search optimization).

The return value is an array of zero or more strings, each containing one keyword.

String[] getContentLocationCode()

Get 3-character codes for the country/geographical locations associated with the object.

The return value is an array of zero or more ISO 3166 three-letter country codes.

String[] getContentLocatonName()

Get the names of the countries and other locations associated with the object.

The return value is an array of zero or more location names.

String[] getReferenceService()

Get the service identifiers (provider and product) of any prior envelopes to which the current object refers.

The return value is an array of zero or more service identifiers.

Each service has an associated date and number (see getReferenceDate, page 64, and getReferenceNumber, page 64).

String[] getReferenceDate()

Get the dates of any prior envelopes to which the current object refers. The dates are associated with specific services in the IIM header (see getReferenceService, page 63), but the API does not currently preserve the associations.

The return value is an array of zero or more date strings.

String[] getReferenceNumber()

Get the envelope numbers of any prior envelopes to which the current object refers. The numbers are associated with specific services in the IIM header (see getReferenceService, page 63), but the API does not currently preserve the associations.

The return value is an array of zero or more strings, each containing a reference number.

String[] getByline()

Get the names of the people or organizations who created the object.

The return value is an array of zero or more strings, each containing a name.

String[] getBylineTitle()

Get the professional titles of the people or organizations who created the object. The titles are associated with specific bylines in the IIM header (see getByline, above), but the API does not currently preserve the associations.

The return value is an array of zero or more strings, each either empty or containing a person's title, such as "staff photographer", "correspondent," and so on.

String[] getContact()

Get the contact information for people who can provide more information about the object.

The return value is an array of zero or more strings, each containing a person's name and contact information.

String[] getWriterEditor()

Get the names of the people who wrote/edited the object.

The return value is an array of strings, each containing a person's name.

String getObjectType()

Get a constant specifying the general object type (news, data, or advisory).

The return value is one of 01:News, 02:Data or 03:Advisory.

String getObjectName()

Get the shorthand descriptive name, or slug, of the object.

The return value is text that identifies the object (example, "Ferry sinks").

String getEditStatus()

Get the editorial status of the object (for example, correction, and so on).

The return value is a string specifying the status of the object, as defined by the provider.

String getEditorialUpdate()

Get the new editorial status of the object, in relation to a previous instantiation of it.

The return value is a two-character numeric string containing a code specifying the update type. The only update type defined in the IIM specification is "01", indicating that the object repeats information from an associated object in a different language.

String getCategory()

Get the top-level subject code for the object.

The return value is a string containing a code from a list maintained by the provider.

String getFixtureIdentifier()

Get information about an object that recurs often and predictably.

The return value is a string such as “Weather Forecast”, as defined by the provider.

String getReleaseDate()

Get the earliest date on which the object can be released (also called the embargo date).

The return value is a string containing a date in CCYYMMDD format (following the ISO 8601 standard). For example, "19890317" indicates data for release on March 17, 1989.

String getReleaseTime()

Get the earliest time at which the object can be released.

The return value is a string containing a time in HHMMSS±HHMM format (following the ISO 8601 standard). For example, "090000-0500" indicates object for use after 0900 in New York (five hours behind UTC) when daylight saving time is not in effect.

String getExpirationDate()

Get the date on which the object is no longer valid.

The return value is a string containing a date in CCYYMMDD format (following the ISO 8601 standard). For example, "19890317" indicates that the object should not be used after March 17, 1989.

String getExpirationTime()

Get the time at which the object is no longer valid.

The return value is a string containing a time in HHMMSS±HHMM format (following the ISO 8601 standard). For example, "090000-0500" indicates an object that should not be used after 0900 in New York (five hours behind UTC).

String getSpecialInstructions()

Get other editorial information about the object such as embargoes and warnings.

The return value is text such as “second of four stories”.

String getActionAdvised()

Get the type of action that the receiver should apply to the previous version of the same object.

The return value is a two-character numeric string containing a code from the following list:

| | |
|---|---|
| 01: | Object kill (cease using this object) |
| 02: | Object replace (replace the previous version with this one) |
| 03: | Object append (add this one to the end of the previous version) |
| 04: | Object reference (cross-reference this object to a different, associated one) |

String getDateCreated()

Get the date on which the intellectual content of the object was created.

The return value is a string containing a date in the form CCYYMMDD (following the ISO 8601 standard). Where the month or day cannot be determined, the information will be represented by “00”. Where the year cannot be determined, the information for century and year will be represented by

"00". For example, "19900127" indicates the intellectual content created on January 27, 1990.

String getTimeCreated()

Get the time at which the intellectual content of the object was created.

The return value is a string containing the time in HHMMSS±HHMM format (following the ISO 8601 standard). For example, "133015+0100" indicates that the object intellectual content was created at 1:30 p.m. and 15 seconds Frankfurt time, one hour ahead of UTC.

String getDigitalCreationDate()

Get the date on which the digital copy of the object was created.

The return value is a string containing a date in CCYYMMDD format (following the ISO 8601 standard). For example, "19890317" indicates that the digital form of the object was created on March 17, 1989.

String getDigitialCreationTime()

Get the time at which the digital copy of the object was created.

The return value is a string containing the time in HHMMSS±HHMM format (following the ISO 8601 standard). For example, "133015+0100" indicates that the digital form of the object was created at 1:30 p.m. and 15 seconds Frankfurt time, one hour ahead of UTC.

String getOriginalProgram()

Get the name of the software application originally used to create the object.

The return value is the name of the originating application, such as "Word Perfect," "FrameMaker," and so on.

String getProgramVersion()

Get the version of the software application originally used to create the object.

The return value is the version number of the program returned by the getOriginalProgram method, above.

String getObjectCycle()

Get the object target news cycle(s) for the object.

The return value is one of morning (“a”), evening (“p”) or both (“b”). It is used mainly in North America.

String getCity()

Get the name of the city from which the object originated.

The return value is the name of the city (e.g., “London”).

String getSubLocation()

Get a sub-location (such as a neighborhood in a city) from which the object originated.

The return value is the name of the sub-location (e.g., “Soho”).

String getProvinceState()

Get the name of the province or state from which the object originated.

The return value is the name of the province or state (e.g., “New Jersey”).

String getCountryCode()

Get the country code for the country from which the object originated.

The return value is the country code in ISO 3166 format. Note that this method returns the three-character version of the ISO country code, not the two-character version commonly used on the Internet.

String getCountryName()

Get the name of the country from which the object originated.

The return value is the name of the country corresponding to the code returned by the getCountryCode method, above.

String getOriginalTransmissionReference()

Get information about from where the object was transmitted.

The return value is a string, from a list the provider maintains.

String getHeadline()

Get the headline for the object.

The return value is a string describing the story.

String getCredit()

Get the provider name for the object (not necessarily the owner or creator).

The return value is the name of the provider.

String getSource()

Get the original creator/owner of the object.

The return value is the name of the creator/owner of the object. This could be an agency, a member of an agency or an individual.

String getCopyrightNotice()

Get the copyright notice for the object.

The return value is the copyright notice.

String getCaptionAbstract()

Get the text description of the object.

The return value is a text description of the object. This is used particularly where the object is not text.

String getRasterizedCaption()

Get the rasterized version of the caption for the object.

The return value is the rasterized objectdata description (in other words, a picture of the caption text). It is used where characters that have not been coded are required for the caption.

String getImageType()

Get the code for a type of image.

The return value is a numeric and alphabetic character. The numeric characters 1 to 4 indicate the number of components in an image, in single or multiple envelopes. The numeric character 0 indicates no object data. The numeric character 9 specifies that the objectdata contains supplementary data to an image.

0 = no object data

'1' = Single component (e.g., black and white or one component) of a colour project.

'2', '3', '4' = Multiple components for a colour project.

'9' = Supplemental objects related to other objectdata

The alphabetic character will indicate the exact content of the current objectdata in terms of colour composition:

'W' = Monochrome

'Y' = Yellow component

'M' = Magenta component

'C' = Cyan component

'K' = Black component

'R' = Red component

'G' = Green component

'B' = Blue component

'T' = Text only

'F' = Full color composite, frame sequential

'L' = Full color composite, line sequential

'P' = Full color composite, pixel sequential

'S' = Full color composite, special interleaving

String getImageOrientation()

Get information about the layout of the image.

The return value is landscape (l), portrait (p) or square (s).

String getLanguageIdentifier()

Get the language associated with the object.

The return value is a two-letter code that identifies the language (without a country suffix) associated with the object in ISO 639:1988 format. In the future, the IIM may use three-letter language codes.

String getAudioType()

Get the type of audio associated with the object (stereo, mono).

The return value is a numeric and an alphabetic character. Values for the numeric character include the following:

0 = no objectdata

1 = mono

2 = stereo

Values for the second, alphabetic character include the following:

'A' = Actuality

'C' = Question and answer session

'M'= Music, transmitted by itself

'Q' = Response to a question

'R' = Raw sound

'S' = Scener

'T' = Text only

'V' = Voicer

'W' = Wrap

String getAudioSamplingRate()

Get the rate at which the audio was sampled, in Hertz.

The return value is a string containing the sampling rate, with leading zeroes. For example, "011025" indicates a sample rate of 11025 Hz and "022050" indicates a sample rate of 22050 Hz.

String getAudioSamplingResolution()

Get the number of bits in each audio sample.

The return value is a string specifying the sampling resolution, as in the following:

"08" for a sample size of 8 bits

"16" for a sample size of 16 bits

String getAudioDuration()

Get the length of time the audio sample runs.

The return value is a string containing a time in HHMMSS format. It indicates the running time of an audio object when played back at the speed at which it was recorded. For example, "000105" indicates a cut lasting one minute, five seconds.

String getAudioCue()

Get the out-cue information for an audio object, specifying the text or action at the end of an audio clip.

The return value is the end of the audio object, such as "... better as a team" or "fades".

String getObjectPreviewFileFormat()

Get information about the file format of the object preview.

The return value is the file format. The file format must be registered with IPTC or NAA with a unique number assigned to it.

String getObjectPreviewFileFormatVersion()

Get the version number of the file format used for the object preview.

The return value is a string representing the version of the specified by the get ObjectPreviewFileFormat method.

byte[] getObjectDataPreviewData()

Get the preview data for the object.

The return value is a byte array containing the binary preview data.

### *Constants*

This class inherits all the constants in the IPTCConstants interface (page 60).

## IPTCParseException

Some methods in the gari.media and gari.media.iptc packages throw this exception when there is an error parsing an IPTC IIM section.

### *Methods*

IPTCParseException inherits the methods from java.lang.Exception.

## gari.net Package

The gari.net package contains the lower-level networking support (messaging layer) for the NewsEdge Java API. For information on how this package fits into the overall networking architecture, see "Architectural Overview" (page 139).

## QuipEventListener Interface

### *Synopsis*

```
class MyListener implements QuipEventListener
{

    public void QuipEventNotification (QuipEvent event)
    {
        System.err.println("Network event: " +
      event.getMessage());
    }

}
```

### *Notes*

The client application creates a class implementing this interface, then registers the object through the QuipClient addQuipEventListener method (page 80), inherited by gari.inap.InapClient).

The listener object receives notification of connection events through its QuipEventNotification method. This information is encapsulated in the QuipEvent object (page 81).

One connection can have multiple listeners (for example, one for logging, one for the console, and so on).

### *Methods*

void QuipEventNotification (QuipEvent event)

The library invokes this method every time a significant event occurs in the network connection. See "QuipEvent Class," page 81 for details.

## Channel Class

This is an ancestor class of gari.news.NewsChannel (page 102) and provides some of its methods.

### *Constructors*

The client application never creates this class directly.

### *Methods*

QuipClient getQuipClient ()

Get a reference to the underlying QuipClient (superclass of the InapClient). It is useful for adding a network event listener (see "QuipEvent Class," page 81).

short getPort ()

Get the server port this channel uses. This value will be one of the *_SERVICE constants listed below. The constant can also be passed to the gari.inap.InapClient constructor (page 46).

QuipBuffer receive ()
throws java.io.IOException, TimeoutException

Receive a message from the server through this channel. The QuipBuffer object is opaque to the client application. It simply passes on to methods or constructors in other classes (see, for example, HeadlineAnswer Class on page 89).

Throws a java.io.IOException, if there is a network error, and a TimeoutException if there is a connection timeout.

### *Constants*

The following are constants for different TCP ports used on the News Server:

ADMIN_SERVICE – for internal use in the library (6963)

HTTP_SERVICE - uses the standard HTTP port 80

HTTPS_SERVICE - uses the standard TLS/SSL port 443

NEWS_SERVICE – uses port 6973

QUOTES_SERVICE - uses port 6983

STARTUP_SERVICE – uses port 6973

## ProxyData Class

### *Synopsis*

```
ProxyData proxy = new ProxyData(proxyHost, proxyPort);
InapClient client = new InapClient(newsHost, login, proxy);
```

### *Notes*

The ProxyData class provides information about a network proxy. It is supplied to the InapClient object when establishing a connection, if the client application is behind a proxy. For Java applets running on client machines, where the proxy settings (if any) are not known in advance, see the detectProxy Class (page 129).

### *Constructors*

ProxyData ()

Creates an empty object. You must use the set* methods (page 78) to configure it.

ProxyData (String host, String port)

Use if the proxy does not require authentication.

ProxyData (String host, String port, String username, String password)

Use if the proxy requires authentication.

The host parameter is the hostname/IP address of the proxy. The port is the TCP port number of the proxy, represented as a string. The username parameter is the account name for logging into the proxy, if the proxy requires authentication. The password parameter is the password for logging into the proxy, if the proxy requires authentication.

### *Methods*

void setProxyHost (String host)

String getProxyHost ()

Accessor and setter for the proxy host address.

void setProxyPort (String port)

String getProxyPort ()

Accessor and setter for the proxy TCP port.

void setProxyUsername (String username)

String getProxyUsername ()

Accessor and setter for the username to authenticate with the proxy (if required).

void setProxyPassword (String password)

String getProxyPassword ()

Accessor and setter for the password to authenticate with the proxy (if required).

## QuipBuffer Class

The QuipBuffer class represents a message passed to or from the NewsEdge News Server.

Client applications should never attempt to work with the class directly but may pass it as an argument to other methods and constructors: they should treat it as a black box. See, for example, the gari.news.ProfileGroup constructor, which takes a QuipBuffer as its argument (page 108). The client application normally obtains a QuipBuffer through the gari.news.NewsChannel receive method inherited from gari.net Channel (page 76).

## QuipClient Class

The QuipClient class is the base class for gari.inap.InapClient (page 46), which represents a higher layer in the network stack.

### *Constructors*

The client application always works through the InapClient object, so it does not need to construct QuipClient objects directly.

### *Methods*

boolean isConnected ()

Return true if there is currently a connection to the news server.

java.net.InetAddress getServerAddress ()

Return the IP address of the News Server.

short getAccountNumber ()

Return the current user's account number.

ProxyData getProxyData ()

Get the proxy data in use for connecting to the server (ProxyData Class, page 77).

boolean isUsingProxy ()

Return true if the library is currently connecting to the news server through a proxy.

void close ()

Explicitly close the connection to the server. The connection does close automatically after it is not in use for a while, or when the same user attempts to open a new connection. However, it is a good idea to use this method when the application terminates or when the connection is no longer required.

void addQuipEventListener (QuipEventListener listener)

Add a listener object to receive notification of connection events. For more information, see QuipEventListener Interface (page 75), QuipEvent Class (page 81) and "How to Monitor the Connection's Health" (page 24).

void removeQuipEventListener (QuipEventListener listener)

Remove one of the listeners attached to the connection.

QuipEventListener [] getQuipEventListeners ()

Get an array of all the listeners currently attached to the connection. If there are no listeners, this method returns an empty array, not null.

## QuipEvent Class

### *Synopsis*

```
class MyListener implements QuipEventListener
{

    public void QuipEventNotification (QuipEvent event)
    {
        System.err.println("Network event: " +
       event.getMessage());
    }

}
```

### *Notes*

The QuipEvent class defines the object passed to listeners (see the QuipEventListener Interface, page 75) to describe the status of the network connection. The object provides information that the client application can use for monitoring logging, notification, and other administrative tasks.

### *Constructors*

Client applications do not need to create QuipEvent objects because the library creates them and passes them to QuipEventListener Interface.

### *Methods*

int getType ()

Get an integer constant describing the event type. This method rreturns one of the constants described below (page 82).

int getAccountNumber ()

Get the number of the account to which the event belongs. Also see the QuipClient getAccountNumber method (page 80).

Use when the same handler is monitoring multiple connections, or to include the account number in the log.

int getMessage ()

Get a textual message describing the event.

### *Constants*

This section describes the constants returned by the getType method (above).

DISCONNECT

The library no longer has a connection to the server.

CONNECT

The library has established a connection to the server.

RECONNECT

The library has reestablished a connection to the server.

RECONNECT_FAILED

The library has failed to reestablish a connection to the server and is not currently connected.

PING_FAILED

The library was unable to detect that the news server is available and working properly.

PING_SUCCESS

The library was able to detect that the news server is available and working properly.

DUPLICATE_LOGIN

A new connection to the news server for the same account has caused the server to drop this connection.

MAX_CONNECTIONS

Cannot connect to the server because the maximum number of server-wide connections has been reached. It is not related to entitlements.

SYSTEM_ERROR

There has been some other kind of connection error. (For details, see the message accompanying the system_error constant.)

## QuipException Class

### *Synopsis*

```
try {
    // some Quip networking operation
} catch (QuipException ex) {
    System.err.println("Networking error: " + ex.getMessage());
}
```

### *Notes*

The library throws a QuipException when there is a problem with the network message layer between the client application and the server.

QuipException is a subclass of java.lang.RuntimeException, so it does not usually need to be caught in client application code (though it is still a good idea to do so).

### *Constructors*

Client applications do not need to construct QuipException objects.

### *Methods*

The class uses standard methods inherited from java.lang.Exception.

## TimeoutException Class

### *Synopsis*

```
try {
    // some Quip networking operation
} catch (TimeoutException ex) {
    System.err.println("Networking timeout: " + ex.getMessage());
}
```

### *Notes*

The library throws a TimeoutException  when a network operation times out.

TimeoutException  is a subclass of java.io.IOException. Many operations also throw an IOException,  so catching that can catch timeout exceptions as well. Catch this explicitly when you want to deal with timeouts differently from other I/O problems.

### *Constructors*

Client applications do not need to construct TimeoutException  objects.

### *Methods*

This class uses standard methods inherited from java.lang.Exception.

# gari.news Package

The gari.news package represents the main application layer in the NewsEdge Java library. Your client application will do most of its work in this package.

## Headline Class

### *Synopsis*

```
ExtendedEnumeration headlines =
 headlineAnswer.getElements();
while (headlines.hasMoreElements()) {
    Headline headline = (Headline)headlines.nextElement();
    ResourceID resourceId = headline.getResourceID();
    String headlineText = headline.getText();
    String storySummary = headline.getSummary();
    // etc.
}
```

### *Notes*

The Headline class contains metadata and access information for a single story. It is one of the most important classes for working with the NewsEdge Java API, since it provides the ResourceID, required to look up the entire text of stories.

The Headline object can result from either a history search or a set of newly-arrived real-time stories (see “Task 4: Retrieving the Full Text of an Article,” page 15, and “How to List the Available Wires,” page 41).

### *Constructors*

There are no constructors for this class. Obtain Headline objects through a HistorySearch (page 95) or Profile (page 103) object.

### *Methods*

ResourceID getResourceID ()

Get the resource identifier for the story. The resource identifier includes information about the date, time, provider, service, and so on (see “ResourceID Class,” page 116).

String getText ()

Get the text of the headline. This text is the display text (for a summary list, and so on), which may be shorter than the entire headline included in the story.

The following is an example: “CREW Requests Investigation into Rep. Doc Hastings and Top Aide”.

String getSummary ()

Get a short text summary of the story. The summary text is appropriate for displaying in a list of news stories, search results, table of contents, and so on.

The following is an example of a summary:

> WHAT: ADDICTION, the centrepiece feature documentary of HBO's groundbreaking, multi-platform campaign, brings together the nation's leading experts on drug and alcohol addiction with a collection of award-winning filmmakers to shed light on addiction, its causes and the latest and most promising developments in treatments. SOURCE HBO.

boolean isHot ()

This is a convenience method that returns true if the story includes the Dow Jones code “N/HOT”. To detect the impact rating of stories from all wires and not just Dow Jones, see getImpactRating, page 88.

This method is not a general solution for detecting "hot" or breaking stories. For information on how to detect such stories, see getCodes, below.

String getSymbols ()

Returns a semicolon-separated list of stock ticker symbols for companies associated with the story, as in the following example: "Toronto:SJR.B;NYSE:SJR"

Each symbol contains an exchange abbreviation and ticker symbol, separated by a colon, as in the following example: "NYSE:SJR". "NYSE" is the exchange, and "SJR" is the ticker symbol.

You can separate the results into an array using the java.lang.String split method:

```
String symbolString = headline.getSymbols();
String symbols[] = symbolString.split(";");
```

String getCodes ()

Returns a semicolon-separated list of subject codes associated with the story, as in the following example: "IC/comp.soft;IC/comp;NI/Computer_Systems;NI/Software". (The list is generally much longer.)

Each code contains a vocabulary identifier and a value, separated by a forward slash. In the example, "IC/comp" the vocabulary identifier is "IC" (industry code) and the value is "comp" (computers).

You can separate the results into an array using the java.lang.String split method:

```
String codeString = headline.getCodes();
String codes[] = codeString.split(";");
```

See also isHot (page 86) and getImpactRating (page 88).

int getType ()

Get a constant for the status of the story from the library's perspective. For a description of the constants HISTORY_HIT, PROFILE_MATCH, and PROFILE_MISS, see "Constants" (page 88). Note that the client application does not normally see the PROFILE_MISS constant. Users would see this constant if they created a Profile and passed true to the setShowAll method (see the setShowAll method, page 105). Any profile with the ShowAll property set to true will have all headlines sent. Headlines that 'hit' the profile will be of type PROFILE_MATCH, all others will be PROFILE_MISS.

This method has limited use for client applications.

boolean getIsUpdate ()

Returns true if this is an updated version of a previous story, or false if it is not. Always returns false for headline objects from an historical search.

int getImpactRating ()

This is a convenience method to extract the MC/HOT# impact rating from the code string (see getCodes, page 87). The impact rating indicates an estimate of the importance of a story to the audience. The result of this method is an integer from 1 (low impact) to 9 (high impact). If the story has no impact code assigned, the method returns a 0.

### *Constants*

This section describes the constants returned by the getType method (page 88).

HISTORY_HIT

Indicates that the story matched the history search pattern.

PROFILE_MATCH

Indicates that the story matched the profile search pattern.

PROFILE_MISS

Indicates that the story did not match the profile search pattern (client applications should not normally see this value). Users would see this constant if they created a Profile and passed true to the setShowAll method (page 105). Any profile with this set to true will have all headlines sent. Headlines that 'hit' the profile will be of type PROFILE_MATCH, all others will be PROFILE_MISS.

## HeadlineAnswer Class

### *Synopsis #1: After a history search*

```
search.doReadHeadlines(channel);
HeadlineAnswer answer = new
 HeadlineAnswer(channel.receive());
ExtendedEnumeration headlines = answer.getElements();
while (headlines.hasMoreElements()) {
    Headline headline = (Headline)headlines.nextElement();
    // process the Headline
}
```

### *Synopsis #2: In a real-time news observer*

```
public void update (Observable observable, Object object)
{
    ProfileHeadlines ph = (ProfileHeadlines)object;
    HeadlineAnswer answer = ph.getHeadlines();
    ExtendedEnumeration headlines = answer.getElements();
    while (headlines.hasMoreElements()) {
        Headline headline =
 (Headline)headlines.nextElement();
        // process the Headline
    }
}
```

### *Notes*

The HeadlineAnswer class represents the result of an historical query or a packet of newly-arrived stories. For an historical search, the client application must create this object itself.

The client application uses the HeadlineAnswer class to obtain an ExtendedEnumeration object (page 128) that allows the application to iterate through the headlines, each of which contains metadata and access information for a single story.

For more information, see Headline (page 85).

#### *Constructors*

HeadlineAnswer (gari.net.QuipBuffer buffer)

Create a new HeadlineAnswer object from a message buffer. Normally, create the message buffer using the NewsChannel receive method (page 76), after invoking the HistorySearch doReadHeadlines method (page 94).

Note that it is not necessary to use this approach to obtain real-time profiles. The library constructs the object for you (see the second synopsis above).

#### *Methods*

gari.util.ExtendedEnumeration getElements ();

Get an enumeration object for stepping through the Headline objects (page 85).

## HeadlineContext Class

#### *Synopsis*

```
// process 20 pages of headlines
for (int x = 0; x < 20; x++) {
    HeadlineAnswer answer = new
 HeadlineAnswer(channel.receive());
    // process the headlines
    HeadlineContext.nextHeadlines(channel);
}
```

### *Notes*

The HeadlineContext class provides static methods to page backwards or forwards through search results. The method moves the pointer to the results that will be returned to the next HeadlineAnswer object.

The context is specific to the NewsChannel (page 102). Multiple NewsChannel objects can maintain their own pointers.

### *Constructors*

Client applications should not instantiate this class, since it has only static methods.

### *Methods*

All these methods work only after the HistorySearch doReadHeadlines method has been called on the channel (see page 94). Calling doReadHeadlines affects which headlines the next HeadlineAnswer object will return.

Note that you must use the HistorySearch setMaxHits (page 97) and setGetExactHitCount (page 97) methods to determine the page sizes.

static void nextHeadlines (NewsChannel channel)
throws java.io.IOException

Page forward so that the next HeadlineAnswer object created will receive a new page of search results.

Throws a java.io.IOException if there is a networking error.

static void prevHeadlines (NewsChannel channel)
throws java.io.IOException

Page backward so that the next HeadlineAnswer object will receive a previous page of search results (one that the application has already received and moved past).

Throws a java.io.IOException if there is a networking error.

static void currentHeadlines (NewsChannel channel)
throws java.io.IOException

Move back to the initial page of top search results (the ones received by the first HeadlineAnswer object), before invoking any of the methods in this class).

Throws a java.io.IOException if there is a networking error.

static void requestHeadlinePage (NewsChannel channel, int pageNumber)
throws java.io.IOException

Move to a specific page of search results, using a zero-based index for the first results to be returned.

Note that this method works only for pages that have already been received. You cannot use it to move forward to new pages.

Throws a java.io.IOException if there is a networking error.

## HistoryCacheSearch Class

### *Synopsis*

```
HistoryCacheSearch search = new HistoryCacheSearch(client, "msft
                                                    ibm");
search.setCacheType(HistoryCacheSearch.CACHE_UPDN);
HeadlineAnswer answer = new
       HeadlineAnswer(search.doReadHeadlines());
```

### *Notes*

The HistoryCacheSearch class requests items from cache, through a ticker symbol or subject code (see the getCodes and getSymbols methods, page 87).

The class connects to a separate server that caches stories matching pre-defined queries. Searches can include only symbols or topic codes that are defined at server startup.

Note that the HistoryCacheSearch class is available only on some accounts, as an optional extra feature, by special arrangement with NewsEdge Customer Support.

### *Constructors*

HistoryCacheSearch (gari.inap.InapClient client)

Construct a new cache search.

Note that this constructor uses InapClient directly instead of working through a NewsChannel.

HistoryCacheSearch (gari.inap.InapClient client, String queryItems)

This is a convenience constructor and is the equivalent of invoking setQueryItems (page 94) after creating this object.

HistoryCacheSearch (gari.inap.InapClient client, String queryItems, int count)

This is a convenience constructor and is the equivalent of invoking setQueryItems (page 94) and setCount (page 94) after creating this object (the count defaults to 10 otherwise).

HistoryCacheSearch (gari.inap.InapClient client, String queryItems, int count, int port)

This is a convenience constructor and is the equivalent of invoking setQueryItems (page 94), setCount (page 94), and setPort (page 94) after creating this object (the port defaults to 6973 otherwise).

### *Methods*

String getCacheType ()

void setCacheType (String cacheType)

Accessor and setter for the type of cache the search will access. Uses one of the constants CACHE_NEWS, CACHE_UPDN, CACHE_WSOD, or CACHE_EDGAR10K (page 95).

If this value is not set, it defaults to CACHE_NEWS.

String getQueryItems ()

void setQueryItems (String queryItems)

Accessor and setter for the query items for which to search.

Returns a space-separated list of stock ticker symbols or subject codes (e.g., "ibm msft" to find articles about IBM and/or Microsoft).

Note that the method can search only for items that were predefined in the cache server when it was started up.

int getCount ()

void setCount (int count)

Accessor and setter for the approximate maximum number of items to return. The default value is 10.

int getPort ()

void setPort (int port)

Accessor and setter for the TCP port to use for the connection. The default is 80.

String doReadHeadlines ()

Read headlines from the cache server.

Returns a buffer that can be passed to the HeadlineAnswer constructor to create a list of Headline objects (as in the synopsis).

### Constants

This section describes the constants returned by the getCacheType and setCacheType methods (page 94).

CACHE_NEWS

Search in the cache of general stories.

CACHE_UPDN

Search in the cache of Upgrades and Downgrades.

CACHE_WSOD

Search in the cache of Wall Street On Demand financial research.

CACHE_EDGAR10K

Search in the cache of U.S. Security and Exchange Commission EDGAR 10K filings from public companies.

## HistorySearch Class

### Synopsis

```
HistorySearch search = new HistorySearch("Manhattan");
search.doReadHeadlines(channel);
HeadlineAnswer answer = new
 HeadlineAnswer(channel.receive());
ExtendedEnumeration headlines = answer.getElements();
```

### Notes

The HistorySearch class provides methods for searching against recent historical stories, including topic codes, symbols, fulltext, boolean, date range, and so on.

Both this class and the Profile class (page 103) extend the MetaHeadlineQuery class (page 99), which defines many of the methods for this class.

For additional information, see “Task 2: Searching Historical News,” page 9 and “How to Search Historical Headlines,” page 27.

### Constructors

HistorySearch ()

Construct a new search object without a query string. Note that you must invoke the setQueryText method (page 99, inherited from MetaHeadlineQuery) to set the query before executing a search.

HistorySearch (String query)

Construct a new search object and set a query string.

The following table describes the pattern matching symbols:

| Pattern | Description |
|---|---|
| * | Match anything. |
| word1 word2 | Match word1 or word2. |
| +word | The specified word is required in the results. |
| -word | The specified word is not allowed in the results. |
| /symbol | Searches for a stock ticker symbol (e.g., “/MSFT”)<br><br>NOTE: Use only the symbol. Do not include the exchange. |
| “a phrase” | Match the word as a phrase. |

### Methods

This section describes the methods defined specifically by the HistorySearch class. See also the methods in MetaQuery (page 101) and MetaHeadlineQuery (page 99), which are superclasses of HistorySearch. To limit your search to specific wires, see “How to Limit a Search by Wires,” page 31.

void setMaxHits (int maxHits)

Set the approximate maximum number of matches to return for the search. To force an exact maximum, use the setGetExactHitCount method (page 97).

For additional information on matching, see "How to Change the Number of Search Results Returned," page 29.

String getDateRange ()

void setDateRange (String dateRange)

Accessor and setter for the date range for the historical search. Note that the date range is always <u>inclusive</u> of start and end. Use "<YYYYMMDD" for articles published before or on a date, ">YYYYMMDD" for articles published on or after a date, or both combined for a range ">YYYYMMDD <YYYYMMDD".

You also may add time after the date in "HH:SS" format (with a space between the date and time) to specify a time in 24-hour format, using the U.S. eastern time zone.

For more information on specifying a date range for search results, see "How to Limit Search Results by Date," page 30.

void setGetExactHitCount (boolean flag)

If set to true, force the maximum number of hits to be exact instead of approximate (at the cost of extra computation).

See setMaxHits, above.

void setHeadlineResIDOnly (boolean flag)

If set to true, return only the ResourceID for each story, with no other metadata. Note that the setting of this method does affect other methods. If set to true, methods that return metadata (for example, getcodes, page 87) will not be available.

boolean getHeadlineResIDOnly ()

Check if the search is returning only ResourceID objects for each story.

See setHeadlineResIDOnly, above.

boolean getGetExactHitCount ()

Check if the search is enforcing a strict maximum number of results.

See setGetExactHitCount, above.

void setShowChainHeadOnly (boolean flag)

If set to true, display only the headline of the first story of the chain (different takes on the same story). This method pertains mainly to Dow Jones.

String getName ()

void setName (String name)

Accessor and setter for a name that the client application can use to identify this search. This method helps a client manage multiple search objects. Only the first story of the chain is displayed. This pertains mainly to Dow Jones (DJ).

boolean getShowChainHeadOnly ()

Check whether this search is returning only the most recent item from a chain of stories.

See setShowChainHeadOnly, above.

void doReadHeadlines (NewsChannel channel)
throws java.io.IOException

Run the search and prepare the results to be sent to a NewsChannel.

See "HeadlineAnswer Class," page 89 for how to deal with the results.

Throws a java.io.IOException if there is a networking error.

### *Constants*

See the constants for the MetaQuery superclass (page 102).

## MetaHeadlineQuery Class

This is a shared superclass of HistorySearch (page 95) and Profile (page 103). It defines general methods for configuring a news story search, and extends the MetaQuery class (page 101).

### *Constructors*

Client applications do not construct instances of this class directly. It is a superclass of HistorySearch (page 95) and Profile (page 103).

### *Methods*

In addition to the methods inherited from MetaQuery (page 101), this class defines the following methods for its HistorySearch (page 95) and Profile (page 103) subclasses:

String getQueryText ()

void setQueryText ()

Accessor and setter for the text of the search query.

The following table describes the pattern matching symbols:

| Pattern | Description |
|---|---|
| * | Match anything. |
| word1 word2 | Match word1 or word2. |
| +word | The specified word is required in the results. |
| -word | The specified word is not allowed in the |

| | |
|---|---|
| | results. |
| /symbol | Searches for a stock ticker symbol (e.g., "/MSFT")<br><br>NOTE: Use only the symbol. Do not include the exchange. |
| "a phrase" | Match the word as a phrase. |

The client application normally sets this value in the HistorySearch (page 95) and Profile (page 103) constructors.

boolean getSuppressHeadlineOnly ()

void setSuppressHeadlineOnly (boolean flag)

Accessor and setter to determine whether headline-only stories will be suppressed.

If the return value is true, the search will not include headline-only (no body) stories in the results.

Note that since this method ignores all headline-only items, news alerts that may be headline-only will be missed.

boolean getSuppressTemp ()

void setSuppressTemp (boolean flag)

Accessor and setter to determine whether the search will suppress temporary stories.

If set to true, the search will not return any temporary stories.

Note that only Dow Jones has temporary stories. This method does not affect stories from other providers.

String getWires ()

void setWires (String wires)

Accessor and setter to determine the pattern for wires (sources) to search.

A list of all wires is available through the UserNewsInfo class (page 122).

The pattern consists of "[" for include or "]" for exclude, followed by a space-separated list of two-letter wire identifiers, as in the following examples:

"[ PR BW" searches only PRNewsWire and Business Wire

"] DJ" searches everything but Dow Jones

An empty string means include all wires.

## MetaQuery Class

The MetaQuery class is the base class for MetaHeadlineQuery (page 99). It provides methods for use by the HistorySearch (page 95) and Profile (page 103) classes.

This class sets the type of returned data. However, since the default is to return everything, client applications normally do not need to use these methods. If you want to return less than the default, you could use this class to prune some of the returned information.

### *Constructors*

Client applications do not construct instances of this class directly. It is a superclass of HistorySearch (page 95) and Profile (page 103).

### *Methods*

short getMetadataType ()

void setMetadataType (int type)

Accessor and setter to determine the type of metadata returned about news stories. This method affects metadata only in the headline object, not in the content.

The type is one of the MT_* constants defined below (page 102). By default, the news server returns all available metadata. Normally, the client application does not need to change that.

### *Constants*

MT_NONE

Do not return any metadata with the Headline objects.

MT_SYMBOLS_AND_CODES

Return stock ticker symbols and subject codes.

MT_SYMBOLS_ONLY

Return only stock ticker symbols.

MT_CODES_ONLY

Return only subject codes.

MT_ALL_METADATA

Return all available metadata (default).

MT_SYMBOLS_AND_SPECIAL

Similar to MT_SYMBOLS_AND_CODES, but returns some Dow Jones-specific date codes.

## NewsChannel Class

### *Synopsis*

```
NewsChannel channel = new NewsChannel(client);
```

### *Notes*

This is the key access object for most news activities. The client application must create a news channel before searching historical news, receiving real-time news, and so on (see “Task 1: Connecting to the News Server,” page 8 and “How to Connect to the Server,” page 19).

The class represents a message conduit to the news server. You may create many news channels from one gari.inap.InapClient connection (page 46). It inherits all of its methods from gari.net.Channel (page 76).

The object is passed as an argument to other objects that need to communicate with the server.

### *Constructors*

NewsChannel (QuipClient client)

Construct a new news channel for passing messages to and from the news server.

The client value is usually an InapClient object (a subclass of QuipClient).

See “Task 1: Connecting to the News Server,” page 8 and “How to Connect to the Server,” page 19.

### *Methods*

There are no new methods for this class, but it does inherit methods from Channel (page 76).

## Profile Class

### *Synopsis*

```
Profile profile = new Profile (name, pattern);
profile.setNumber(1); // or any other slot
profile.doSetProfile(newsChannel);
ProfileManager manager = new ProfileManager(profile, newsChannel);
```

### Notes

Profile is the principal class for filtering real-time, live streaming news. A client application registers a Profile with the server for each different pattern it uses to filter incoming news. As a subclass of MetaHeadlineQuery (page 99) and MetaQuery (page 101), Profile inherits methods from both of them, just as HistorySearch (page 95) does.

See also ProfileManager (page 111).

### Constructors

Profile ()

Construct a new real-time news filter.

Profile (String name, String query)

This is a convenience constructor, equivalent to calling the setName method (page 105) with the name parameter and the setQueryText (page 99) with the query parameter.

The name is any string meaningful to the application – you can use it for organizing profiles. If a profile name is not specified, a default one is created. The default name is "untitled #", where "#" is the profile number.

### Methods

In addition to the methods listed here, this class also inherits all of the methods from its superclass MetaHeadlineQuery (page 99) and MetaQuery (page 101).

short getProfileStatus ()

void setProfileStatus (short status)

Accessor and setter for the profile's active status. The status is one of the constants NO_PROFILE, ACTIVE_PROFILE, or INACTIVE_PROFILE, defined below (page 107).

When a profile is not active, it stays on the server, but no Headline objects are sent to the client application.

boolean getShowAll ()

void setShowAll (boolean flag)

Accessor and setter for the showAll property.

If true, return all stories, even if they do not match the query. If false (default), return only stories that match the query.

Setting showAll to true gives the same behaviour as using the * pattern (see HistorySearch, page 96).

boolean getHeadlineResIDOnly()

void setHeadlineResIDOnly (boolean flag)

Accessor and setter for the headlineResIDOnly property.

This method controls whether the Headline objects returned will contain all of the information about the story (headline text, metadata, and so on) or just the ResourceID.

The default is false (i.e., return all of the information). Use true to return only the ResourceID.

String getName ()

void setName (String name)

Accessor and setter for the profile's name.

This method is provided for the benefit of client applications, which can use it to find and organize profiles. The server does not require it. If a profile name is not specified, a default one is created. The default name is “untitled #,” where “#” is the profile number.

int getNumber ()

void setNumber (int number)

Accessor and setter for the profile's slot number.

The news server has a series of slots for different profiles, numbered from 1 to 1023, depending on the account's entitlements.

See the UserNewsInfo class (page 122) to find out your entitlements (including maximum number of profiles). Assigning a profile to a slot deletes any profile currently occupying that slot.

boolean getBackfill ()

void setBackfill (boolean flag)

Accessor and setter for the backfill property.

This method determines whether the news server will fill the profile with recently-arrived stories as well as including new items as they arrive. If true, the server will backfill the profile at startup.

boolean getReturnOK ()

void setReturnOK (boolean flag)

Accessor and setter for the returnOK property.

This method determines whether the server will acknowledge the profile submission. An acknowledgement message ("ADD_OK") is returned on the channel on which the profile was submitted.

boolean getIsNew ()

Returns true if the profile has recently been created, false if otherwise.

void doSetProfile (NewsChannel channel)
throws java.io.IOException

Submit this profile to the server through a news channel. The application must use this method before receiving stories. It will replace any profile currently in the slot (see getNumber and setNumber, page 105).

Throws a java.io.IOException if there is a networking error.

void doDeleteProfile (NewsChannel  channel)
throws java.io.IOException

Remove this profile from the server.

Normally, client applications should not call this method directly. Instead, use the deleteProfile  method (page 112) in the ProfileManager  class.

Throws a java.io.IOException if there is a networking error.

### *Constants*

These constants describe the status of the profile. See getProfileStatus and setProfileStatus (page 104).

NO_PROFILE

There is no profile in this slot.

ACTIVE_PROFILE

The profile in this slot is active, and stories are being delivered.

INACTIVE_PROFILE

There is a profile in this slot, but it is not active, and no stories are being delivered.

## ProfileGroup Class

### *Synopsis*

```
ProfileGroupRequest.doReadProfileGroup(channel);
ProfileGroup group = new ProfileGroup(channel.receive());

ExtendedEnumeration e = group.getElements();
while (e.hasMoreElements()) {
```

```
            Profile profile = (Profile)e.nextElement();
            // process the profile
        }
```

### Notes

The ProfileGroup class represents all the profiles currently registered with the server for the current account. The ProfileGroup object comes through a ProfileGroupRequest (page 109).

The client application can use this object to display profiles in a list for a user, to modify them programmatically, and so on (see "How to Examine your Profiles," page 35).

### Constructors

ProfileGroup (QuipBuffer buffer)
throws gari.news.ParsingException

Create a new profile group from a network message.

You must use the doReadProfileGroup method (page 109) in ProfileGroupRequest first, so that the server will send the information to the channel.

Throws a gari.news.ParsingException if there is an internal problem reading the response.

### Methods

int getMaxProfiles ()

Return the maximum number of profile slots available (not necessarily the number in use). This method is also available through UserNewsInfo (page 122).

gari.util.ExtendedEnumeration getElements ()

Return an ExtendedEnumeration (page 128) of all the profiles currently registered with the server.

This method allows the application to iterate through the profiles.

Use the Profile getNumber method (page 105) to find each profile's slot.

## ProfileGroupRequest Class

### *Synopsis*

```
ProfileGroupRequest.doReadProfileGroup(channel);
ProfileGroup group = new ProfileGroup(channel.receive());
```

### *Notes*

The ProfileGroupRequest class defines a static method for reading a ProfileGroup from the news server.

### *Constructors*

Client applications should not instantiate this class, since it has only static methods.

### *Methods*

static void doReadProfileGroup (NewsChannel channel)
throws java.io.IOException

Instruct the server to send the profile group information. The server returns this information in a message that the client application can obtain using the NewsChannel receive() method (page 76).

The ProfileGroup constructor (page 108) parses the message returned and creates a profile group.

Throws a java.io.IOException if there is a networking error.

## ProfileHeadlines Class

### *Synopsis*

```
public void update(Observable src, Object arg) {

   ProfileHeadlines ph = (ProfileHeadlines)arg;
   HeadlineAnswer answer = ph.getHeadlines();
   // ...
}
```

### *Notes*

The ProfileHeadlines class is a wrapper class for HeadlineAnswer. It holds the HeadlineAnswer object (a packet of Headline objects being delivered to the client application) together with a flag indicating whether these stories are backfilled (see the getBackfill method page 106).

When the client application sets up a profile using the ProfileManager (page 111), the client application's observer update method will receive instances of this object.

Normally, you simply get the HeadlineAnswer using the getHeadlines method (page 110).

### *Constructors*

Client applications should not construct ProfileHeadlines objects directly.

### *Methods*

HeadlineAnswer getHeadlines ()

Return the packet of newly-arrived Headline objects (see "HeadlineAnswer Class," page 89 for more details).

boolean isBackfill ()

Return true if these are recently-arrived stories rather than real-time stories (see the getBackfill method on page 106).

## ProfileManager Class

### *Synopsis*

```
ProfileManager manager = new ProfileManager(profile, newsChannel);
manager.addObserver(new MyProfileFilter());
```

### *Notes*

This class implements the java.util.Observable interface. It allows the client application to receive news through a Profile (page 103) and to manage its lifecycle.

The client application must create a class implementing the standard Java java.util.Observer class. The library passes news through the observer's update method in the form of ProfileHeadlines objects. See “Task 3: Obtaining and Filtering Real-Time News,” page 12, “How to Filter Real-Time Headlines,” page 35 and “ProfileHeadlines Class,” page 110.

The flow of stories (called a “drain”) can be stopped and started. The news server will save stories, space permitting, and transmit them once the client application restarts the drain.

### *Constructors*

ProfileManager (Profile profile, NewsChannel channel)

Create a new profile manager for the profile and channel.

### *Methods*

Profile getProfile ()

Get the profile being managed (the same profile passed through the constructor of the changeProfile method, below).

void restartDrain ()

Stop and restart the transmission of stories to the observers.

void startDrain ()

Start the transmission of stories to the observers.

void stopDrain ()

Stop the transmission of stories to the observers. The news server will save new stories (as space allows) for transmission once the client application invokes startDrain or restartDrain (see above).

void deleteProfile ()

Delete the profile being managed. This is the preferred method for removing a profile from the news server.

void changeProfile (Profile profile)

Remove the current profile and add a new profile in its place.

void addObserver (java.util.Observer observer)

Add an observer to receive packets of real-time stories as they arrive. There may be multiple observers receiving packets at the same time.

void deleteObserver (java.util.Observer observer)

Delete the specified observer, so that it will no longer receive packets of stories.

void deleteObservers ()

Delete all observers currently receiving packets of stories.

## QuantifiedHeadline Class

### *Synopsis*

```
QuantifiedHeadlineRequest.doReadHeadlines(channel);
QuantifiedHeadlineAnswer answer = new
       QuantifiedHeadlineAnswer(channel.receive());
ExtendedEnumeration results = answer.getElements();
while (results.hasMoreElements()) {
    QuantifiedHeadline headline =
       (QuantifiedHeadline)results.nextElement();
     System.out.println(headline.getCount() + ": " +
       headline.getText());
}
```

### *Notes*

This class extends the Headline class (see "Headline Class," page 85). It is identical to the Headline class, except for the addition of a property for the number of times the story has been viewed.

The news server does not support quantified headlines for all accounts by default – you must make special arrangements with NewsEdge Customer Support to use them.

Note that this feature is not available on all servers.

### *Constructors*

Client applications should not construct these objects directly. The objects come from a QuantifiedHeadlineAnswer (page 114).

### *Methods*

In addition to the following, this class inherits the methods from the Headline class (page 85).

int getCount ()

Returns the number of times the story has been read.

## QuantifiedHeadlineAnswer Class

### *Synopsis*

```
QuantifiedHeadlineRequest.doReadHeadlines(channel);
QuantifiedHeadlineAnswer answer = new
        QuantifiedHeadlineAnswer(channel.receive());
ExtendedEnumeration results = answer.getElements();
```

### *Notes*

The QuanitfiedHeadlineAnswer class is similar to HeadlineAnswer (see "HeadlineAnswer Class," page 89), except that it returns an enumeration of QuantifiedHeadline objects instead of Headline objects.

### *Constructors*

There is only one constructor for use by client applications:

QuantifiedHeadlineAnswer (gari.net.QuipBuffer buffer)

Construct a new object based on a message received from the news server. The buffer is the result of calling the receive method (page 76) on a NewsChannel.

Note that it is necessary to invoke the QuantifiedHeadlineRequest doReadHeadlines method on the news channel first (page 115).

The client application should treat the QuipBuffer parameter as a black box.

### *Methods*

ExtendedEnumeration getElements ()

Return a gari.util.ExtendedEnumeration of QuantifiedHeadline objects.

Note that each item in the enumeration must be cast from java.lang.Object to QuantifiedHeadline (page 113).

## QuantifiedHeadlineRequest Class

### *Synopsis*

```
QuantifiedHeadlineRequest.doReadHeadlines(channel);
QuantifiedHeadlineAnswer answer = new
       QuantifiedHeadlineAnswer(channel.receive());
ExtendedEnumeration results = answer.getElements();
```

### *Notes*

This class defines a static method that causes the news server to send QuantifiedHeadline objects through the news channel.

The client application must invoke the static doReadHeadlines method (page 115) on the news channel before creating a QuantifiedHeadlineAnswer object (page 114), so that the server will send the information to the channel.

### *Constructors*

Client applications should not instantiate this class, since it has only static methods.

### *Methods*

static void doReadHeadlines (NewsChannel channel)
throws java.io.IOException

Instruct the news server to deliver QuantifiedHeadline objects through the news channel.

Note that quantified headline support is available only by special arrangement with NewsEdge Customer Support. It is not available for most accounts.

Throws a java.io.IOException if there is a networking error.

## ResourceID Class

### *Synopsis*

```
ResourceID id = headline.getResourceID();
String date = id.getDate();
String time = id.getTime(); // etc.
StoryRequest request = new StoryRequest(id);
```

### *Notes*

This class provides a multi-part identifier for the story. The identifier includes some metadata, such as the date, time, provider, service, and a short identifier (similar to the slug).

The identifier can be converted to a string and parsed back into a ResourceID object.

This is the key object for obtaining the full content of a story (see StoryRequest, page 120).

### *Constructors*

ResourceID (String id)

Parse a resource ID string into a ResourceID object. This is the reverse of the toString method (page 118).

Normally, client applications do not need to use the constructor explicitly, since they obtain preparsed ResourceID objects from the Headline object getResourceID method (page 122) for both real-time and historical news.

See toString (page 118) for the string format.

### Methods

String getDate ()

Return an 8-character string describing the story's publication date (not necessarily the date of transmission).

The format is YYYYMMDD (e.g., "20070322" for March 22, 2007).

The date is the same for all takes of a story: it is the original publication time, not the transmission time.

String getTime ()

Return a 4-character string describing the story's publication time (not necessarily the time of transmission).

The format is HHMM on a 24-hour clock, (e.g., "1500" for 3:00 pm).

The time is always in U.S. Eastern time (EDT or EST, as appropriate).

String getProvider ()

Return an 8-character string identifying the story's upstream provider (i.e., the provider before NewsEdge).

Identifiers are right-padded with underscore characters to bring them up to 8 characters (e.g., "PR_NEWS_" for PR Newswire).

See "How to List the Available Wires" (page 41) for how to look up the full name of a provider, and other related information.

String getService ()

Return an 8-character string identifying the story's upstream provider's service. Providers typically offer multiple services for different topics, audiences, and so on.

Identifiers are right-padded with underscore characters to bring them up to 8 characters (e.g., "USPR____" for US press releases).

Service identifiers are unique to the provider, but not necessarily globally unique.

See "How to List the Available Wires" (page 41) for how to look up the full name of a provider, and other related information.

String getID ()

Return the short identifier for the news story. The identifier is like a slug but is guaranteed to be unique for the date/time/provider/service combination. It may or may not use the original provider's slug, depending on uniqueness.

This ID stays the same for all takes of a story. The length is not restricted (e.g., "NYTH008").

String toString ()

Convert this ResourceID to its string representation.

The constructor can parse the string to make a new ResourceID object (for example, if the client application needed to store or transmit it as text, then retrieve it again).

In the example
"200703221100PR_NEWS_USPR____NYTH008"

The first 8 characters are date (see getDate, page 117)

The next 4 characters are time (see getTime, page 117)

The next 8 characters are provider id (see getProvider, page 117)

The next 8 characters are service id (see getService, page 118)

The remainder is the identifier/slug (see getID, page 118)

## StoryAnswer Class

### *Synopsis*

```
StoryRequest request = new StoryRequest(resourceId);
request.doReadStory(channel);
StoryAnswer answer = new StoryAnswer(channel.receive());
String content = answer.getText();
```

### *Notes*

Use this class to receive the full content of a story from the news server. The library delivers the story as text after the client application invokes the StoryRequest doReadStory (page 121) method on the news channel. The format depends on the properties of the StoryRequest object.

For more information on how to get the full content of a news story, see “How to Obtain the Content of a Story,” page 42.

### *Constructors*

StoryAnswer (QuipBuffer buffer)

Construct a story answer from a message received from the news server, after requesting it using a StoryRequest object (page 120).

Use the NewsChannel receive method (page 76) to obtain the message buffer, as in the synopsis (above).

### *Methods*

String getText ()

Get the full content of the news story in text form.

The format may be ASCII, HTML, XML, and so on, depending on the properties of the StoryRequest object; however, the library will always return it as a Java String object.

This is the only way to get the full content and metadata for a news story through this library.

## StoryRequest Class

### *Synopsis*

```
StoryRequest request = new StoryRequest(resourceId);
request.doReadStory(channel);
StoryAnswer answer = new StoryAnswer(channel.receive());
```

### *Notes*

Use this class to request the full content (and full metadata) of a story from the news server. After issuing the request, you can use the NewsChannel receive method (page 76) to read a reply from the server and construct a StoryAnswer object (page 119) to parse the message.

Different delivery formats are available (see "How to Obtain the Content of a Story," page 42).

### *Constructors*

StoryRequest (Headline headline)

StoryRequest (Headline headline, String stylesheet)

StoryRequest (Headline headline, String stylesheet, boolean showChain)

These are convenience constructors that invoke the Headline object's getResourceID (page 122) method to obtain the resource id of the story.

StoryRequest (ResourceID resourceID)

StoryRequest (ResourceID resourceID, String stylesheet)

StoryRequest (ResourceID resourceID, String stylesheet, Boolean showChain)

These three are the same as the first three constructors above but take a resource id object directly.

The first argument is the key for looking up the story (either the resource id, or the Headline object containing it).

The second argument is the style sheet, which controls the delivery format (see getStylesheet, below).

The third argument controls whether the result shows chained stories (multiple takes building on each other) or not. Defaults to false.

### *Methods*

String getStylesheet ()

void setStylesheet (String stylesheet)

Accessor and setter for the style sheet property.

The options are “TEXT” for plain text, “HTML” for hypertext markup format (used on the Web), “NEWSML” for the IPTC NewsML markup language with embedded XHTML, “NITF” for the IPTC News Industry Text Format, and “XMLNEWS” for Moody’s Analytics’ XMLNews format.

Defaults to “XMLNEWS”.

void setShowChain (boolean flag)

Setter for the showChain property. If true, show multiple chained takes of the story (see “Constants,” page 122).

void doReadStory (NewsChannel channel)
throws java.io.IOException

Read the story into the news channel.

You must invoke this method before creating a StoryAnswer object (page 119).

Throws a java.io.IOException if there is an error reading the story.

Headline getHeadline ()

Get the Headline object (if one was supplied to a constructor).

The return value is undefined if a ResourceID was passed instead.

ResourceID getResourceID ()

Get the resource id of the news story, as supplied to the constructor.

### *Constants*

The following constants are returned by the setShowChain method (page 121):

RETRIEVE_CHAIN

Alias for the boolean true value, for retrieving a chain of takes rather than just the most recent one.

RETRIEVE_TAKE_ONLY

Alias for the boolean false value, for retrieving only the most recent take of a story.

## UserNewsInfo Class

### *Synopsis*

```
UserNewsInfoRequest.doUserNewsInfoRequest(channel);
UserNewsInfo info = new UserNewsInfo(channel.receive());
String maximumProfiles = info.getNProfiles();
WireList wires = info.getWireList();
```

### Notes

The UserNewsInfo class provides information on entitlements, such as limits and so on, for the current user. The client application constructs a UserNewsInfoRequest (page 124) object by reading the buffer from NewsChannel.

Note that the methods return numeric values as string representations. Use java.lang.Integer or java.lang.Float valueOf methods (or similar) to convert to a number.

### Constructors

Client applications do not need to construct instances of this class.

### Methods

Most of the methods in this class are obsolete. Only the following are still relevant for client applications:

java.lang.String getNProfiles ()

Returns a string representation of the number of profiles (real-time filter) slots available.

See “Task 3: Obtaining and Filtering Real-Time News,” page 12 and “How to Examine your Profiles,” page 35.

java.lang.String getNWires ()

Returns a string representation of the number of wires (upstream providers) available.

WireList getWireList ()

Returns a list of records describing each of the wires (upstream providers) available.

See “WireList Class,” page 126 for more information.

## UserNewsInfoRequest Class

### *Synopsis*

```
UserNewsInfoRequest.doUserNewsInfoRequest(channel);
UserNewsInfo info = new UserNewsInfo(channel.receive());
```

### *Notes*

The UserNewsInfoRequest class holds a single static method for reading entitlement information from the server into a NewsChannel.

See "How to Discover Your Entitlements," page 26.

### *Methods*

static void doUserNewsInfoRequest (NewsChannel channel)

Read the entitlements for the account into a buffer in the NewsChannel. See UserNewsInfo (page 122), for information on parsing.

Throws a java.io.IOException if there is a networking error.

## WireInfo Class

### *Synopsis*

```
WireInfo wire = wireList.getWireInfo(i);
$providerService = wire.getProviderService();
$name = wire.getDescription();
```

### *Notes*

The WireInfo class provides two-letter codes, full names, and other information about one news source. It returns up to two levels: base wire (provider) and subwire (product/service). See "How to List the Available Wires," page 41.

The library uses two-letter codes for searches. Client applications can display full names to human readers.

### *Constructors*

Client applications should not call the constructor directly – the library creates all instances of the class.

### *Methods*

This class contains several internal or obsolete methods. The following are the methods that client applications may use:

String getBaseWire ()

Return the two-character code for the base wire service.

String getDisplayWire ()

Return the two-character code for the wire service. If this is a subwire, it may be different than the base wire.

boolean isSubwire ()

True if this is a sub-service of a different base wire.

String getProviderService ()

Returns a 16-character string. The first 8 characters of the string are the provider ID right-padded with underscore characters (e.g., "BIZWIRE_" for Business Wire). This is the same as the value returned by ResourceId.getProvider() (page 117).

The last 8 characters of the string are the service ID right-padded with underscore characters (such as "USPR____" for US press releases). This is the same as the value returned by ResourceId.getService() (page 118).

String getDescription ()

Returns the name of the provider in a format suitable for display (for example, “Business Wire”).

## WireList Class

### *Synopsis*

```
WireList wires = userNewsInfo.getWireList();
for (int i = 0; i < wires.size(); i++) {
    WireInfo wire = wires.getWireInfo(i);
    // process the wire
}
```

### *Notes*

The WireList class provides a list of WireInfo objects (page 124), representing information sources (wires). The WireList getWireList () method (page 123) in the UserNewsInfo class creates a WireList for the client application.

This class implements the standard Java collections interface java.util.List. It adds a getWireInfo method to avoid casting.

### *Constructors*

Client applications should not call the constructors directly, since the library creates instances for the client application.

### *Methods*

In addition to standard methods from java.util.List, this class includes the following method:

WireInfo getWireInfo (int index)

Return information about the wire at index (zero-based).

The following two methods are equivalent, but getWireInfo avoids the need to cast:

```
        info = (WireInfo)wires.get(index);
        info = wires.getWireInfo(index);
```

## ParsingException

### *Synopsis*

```
UserNewsInfo info = null;
try {
    info = new UserNewsInfo(channel.receive());
} catch (ParsingException ex) {
    System.err.println("Error parsing information from news
       server: " +
                       ex.getMessage());
}
```

The constructors for ProfileGroup (page 108) and UserNewsInfo (page 122) throw this exception when there is an error parsing a message from the server. It usually indicates an internal error, not a client error.

### *Constructors*

Client applications do not need to create new instances of this exception.

### *Methods*

This class inherits its methods from java.lang.Exception.

## gari.util package

The gari.util package represents utility classes and methods.

The most important method is ExtendedEnumeration, which the library uses for lists of Headline objects, and other similar information.

## ExtendedEnumeration Interface

### *Synopsis*

```
ExtendedEnumeration headlines = headlineAnswer.getElements();
int headlineCount = headlines.count();
while (headlines.hasMoreElements()) {
    Headline headline = (Headline)headlines.nextElement();
    // process the headline
}
```

### *Notes*

This class represents an extension of the standard Java java.util.Enumeration interface. It adds a method to count the remaining items in the enumeration, and a method to reset the counter. It is used for lists of gari.news.Headline (page 85) and gari.news.Profile (page 103) objects.

### *Methods*

int count ()

Return the number of items remaining unread in the enumeration.

void resetIndex ()

Reset the enumeration so that it returns to the beginning.

## Base64 Class

### *Synopsis*

```
String encoded = Base64.encode(byteData);
byte decoded[] = Base64.decode(encoded);
```

### *Notes*

This class represents static methods for Base64 encoding and decoding.

Base64 encoding allows binary objects like photos to be sent as text. It is used mainly internally, but it might be useful for client applications. There are multiple variants of Base64, and this class gives access to the flavor that NewsEdge uses.

### *Constructors*

Client applications should not instantiate this class, since it has only static methods.

### *Methods*

static byte[] decode (String encoded)

Decode a well-formed complete Base64 string back into an array of bytes. It must have an even multiple of 4 data characters (not counting \n), padded out with "=" as needed.

static String encode (byte data[])

Encode an arbitrary array of bytes as Base64 printable ASCII. It will be broken into lines of 72 characters each. The last line is not terminated with a line separator. The output will always have an even multiple of data characters, exclusive of \n. It is padded out with "=".

## detectProxy Class

### *Synopsis*

```
ProxyData proxy = null;
detectProxy detector = new detectProxy(new
       URL("http://www.yahoo.com/"));
if (detector.isProxySet()) {
  proxy = new ProxyData(detector.getProxyHost(),
        Integer.toString(detector.getProxyPort()));
}
```

### Notes

The detectProxy class is a utility class for auto-detecting a network proxy when the proxy settings cannot be known in advance. It is mainly for use by a Java applet that might be running on a customer's own machine. For a regular client application, you can hard-code the proxy settings.

If you pass a well-known URL like http://www.yahoo.com/ or http://www.google.com/, the class tries to auto-detect the proxy settings.

### Constructors

detectProxy (java.net.URL url)

Create a new proxy detector. For the URL argument, it is best to use a well-known URL like http://www.yahoo.com/ or http://www.google.com/.

### Methods

boolean isProxySet ()

Value is true if a proxy was detected.

String getProxyHost ()

If a proxy was detected, return its host address.

int getProxyPort ()

If a proxy was detected, return its TCP port number.

# Appendix A: Quick Start Listings

This appendix provides complete, short Java command-line application samples for the following tasks from the Quick Start chapter:

“Task 1: Connecting to the News Server,” page 8

“Task 2: Searching Historical News,” page 9

“Task 3: Obtaining and Filtering Real-Time News, page 12

“Task 4: Retrieving the Full Text of an Article,” page 15

The applications are designed for training rather than production use, and do not include robust error handling or logging.

## Listing: Connecting to the News Server

(From “Task 1: Connecting to the News Server,” page 8.)

The following is a complete, short Java command-line application to ping a NewsEdge News Server and display an error message (on failure) or some information from the server (on success). You must replace the values of the USERNAME, PASSWORD, and HOSTNAME constants with your actual connection information:

```
package gari.examples;

import gari.inap.InapClient;
import gari.inap.LoginData;
import gari.net.Channel;
import gari.net.QuipException;

import java.io.IOException;
import java.net.InetAddress;

/**
 * Ping an NewsEdge News Server.
 */
public class NewsServerPing {

    // change these
    private final static String USERNAME = "my_username";
private final static String PASSWORD = "my_password";
    private final static String HOSTNAME = "my_hostname";

    public static void
    main(String[] args)
    throws IOException, QuipException {

        InapClient client = amcConnect(USERNAME, PASSWORD, HOSTNAME);

        // Test if the connection is active
        System.out.println("Connected to NewsEdge News Server.");
        System.out.println("  Is Connected: " + client.isConnected());
    }

    /**
     * Establish a connection to the NewsEdge News Server.
     */
    private static InapClient
    amcConnect(String username, String password,
            String hostname)
    throws IOException, QuipException

    {
        LoginData login = new LoginData(username, password, "JAPI");
        InetAddress address = InetAddress.getByName(hostname);
        return new InapClient(address, Channel.STARTUP_SERVICE, login);
    }
}
```

## Listing: Searching Historical News

(From “Task 2: Searching Historical News,” page 9.)

The following is a complete demo application sample for searching historical news articles. You must replace the values of the USERNAME, PASSWORD, and HOSTNAME constants with your actual connection information:

```
package gari.examples;

import gari.inap.InapClient;
import gari.inap.LoginData;
import gari.net.Channel;
import gari.net.QuipException;
import gari.news.Headline;
import gari.news.HeadlineAnswer;
import gari.news.HistorySearch;
import gari.news.NewsChannel;
import gari.util.ExtendedEnumeration;

import java.io.IOException;
import java.net.InetAddress;

public class NewsHistorySearch {

    // change these
    private final static String USERNAME = "my_username";
    private final static String PASSWORD = "my_password";
    private final static String HOSTNAME = "my_hostname";

    public static
    void main(String[] args)
    throws IOException, QuipException {
        InapClient client = amcConnect(USERNAME, PASSWORD, HOSTNAME);
        NewsChannel channel = new NewsChannel(client);
        ExtendedEnumeration searchResult = doHistorySearch(channel, "xml");

        System.out.println("There are " + searchResult.count()
                + " matching articles.\n");
        while (searchResult.hasMoreElements()) {
            Headline metadata = (Headline) searchResult.nextElement();
            System.out.println("Headline: " + metadata.getText());
            System.out.println("Resource id: " + metadata.getResourceID()
                    + "\n");
        }
    }

    /**
     * Search past articles.
     */
    private static ExtendedEnumeration
    doHistorySearch(NewsChannel channel,
            String pattern)
```

```
      throws IOException {
        HistorySearch search = new HistorySearch(pattern);
        search.doReadHeadlines(channel);
        HeadlineAnswer answer = new HeadlineAnswer(channel.receive());
        return answer.getElements();
    }

    /**
     * Establish a connection to the NewsEdge News Server.
     */
    private static InapClient
    amcConnect(String username, String password,
            String hostname)
    throws IOException, QuipException

    {
        LoginData login = new LoginData(username, password, "JAPI");
        InetAddress address = InetAddress.getByName(hostname);
        return new InapClient(address, Channel.STARTUP_SERVICE, login);
    }
}
```

## Listing: Obtaining and Filtering Real Time News

(From “Task 3: Obtaining and Filtering Real-Time News,” page 12.)

In this sample application, the sample class itself implements the java.util.Observer interface. It displays news headlines and resource IDs until you stop the application.

As with the other examples, enter your actual username, password, and hostname in the constants at the beginning of the code.

Note again that this may not work with proxy servers. For more information on working with proxy servers, see “How to Connect Through a Proxy,” page 21.

```
package gari.examples;

import gari.inap.InapClient;
import gari.inap.LoginData;
import gari.net.Channel;
import gari.net.QuipException;
import gari.news.Headline;
import gari.news.HeadlineAnswer;
import gari.news.NewsChannel;
import gari.news.Profile;
import gari.news.ProfileHeadlines;
import gari.news.ProfileManager;
import gari.util.ExtendedEnumeration;

import java.io.IOException;
import java.net.InetAddress;
import java.util.Observable;
import java.util.Observer;

public class ProfileFilter implements Observer {

    // change these
    private final static String USERNAME = "my_username";
    private final static String PASSWORD = "my_password";
    private final static String HOSTNAME = "my_hostname";

    public static void
    main(String[] args)
    throws IOException, QuipException {
        InapClient client = amcConnect(USERNAME, PASSWORD, HOSTNAME);
        NewsChannel channel = new NewsChannel(client);
        startFilter(channel, "*", new ProfileFilter());
    }

    public static void
    startFilter(NewsChannel channel, String pattern,
            Observer observer)
    throws IOException {
        Profile profile = new Profile("Sample profile", "*");
        profile.setNumber(1);
        profile.doSetProfile(channel);

        ProfileManager manager = new ProfileManager(profile, channel);
        manager.addObserver(new ProfileFilter());
    }
    public void
    update(Observable src, Object arg) {
        HeadlineAnswer answer = ((ProfileHeadlines) arg).getHeadlines();
        ExtendedEnumeration headlines = answer.getElements();
        while (headlines.hasMoreElements()) {
            Headline metadata = (Headline) headlines.nextElement();
            System.out.println("Headline: " + metadata.getText());
            System.out.println("Resource id: " + metadata.getResourceID()
                    + "\n");
        }
    }

    /**
     * Establish a connection to the NewsEdge News Server.
     */
    private static InapClient
    amcConnect(String username, String password,
            String hostname)
    throws IOException, QuipException

    {
        LoginData login = new LoginData(username, password, "JAPI");
        InetAddress address = InetAddress.getByName(hostname);
        return new InapClient(address, Channel.STARTUP_SERVICE, login);
    }
}
```

## Listing: Retrieving the Full Text of an Article

(From “Task 4: Retrieving the Full Text of an Article,” page 15.)

The following is a complete demo application, reusing the amcConnect() and doHistorySearch() methods developed in earlier examples. You must replace the values of the USERNAME, PASSWORD, and HOSTNAME constants with your actual connection information:

```
package gari.examples;

import gari.inap.InapClient;
import gari.inap.LoginData;
import gari.net.Channel;
import gari.net.QuipException;
import gari.news.Headline;
import gari.news.HeadlineAnswer;
import gari.news.HistorySearch;
import gari.news.NewsChannel;
import gari.news.ResourceID;
import gari.news.StoryAnswer;
import gari.news.StoryRequest;
import gari.util.ExtendedEnumeration;

import java.io.IOException;
import java.net.InetAddress;

public class ArticleFetch {

    // change these
    private final static String USERNAME = "my_username";
    private final static String PASSWORD = "my_password";
    private final static String HOSTNAME = "my_hostname";

    public static void
    main(String[] args)
    throws IOException, QuipException {
        InapClient client = amcConnect(USERNAME, PASSWORD, HOSTNAME);
        NewsChannel channel = new NewsChannel(client);
        ExtendedEnumeration searchResult = doHistorySearch(channel, "xml");

        Headline metadata = (Headline) searchResult.nextElement();
        String article = getArticle(channel, metadata.getResourceID());
        System.out.println("Story text:\n" + article);
    }

    /**
     * Get the text of an article.
     */
    private static String
    getArticle(NewsChannel channel, ResourceID resourceId)
    throws IOException {
        StoryRequest request = new StoryRequest(resourceId);
        request.doReadStory(channel);
        StoryAnswer answer = new StoryAnswer(channel.receive());
        return answer.getText();
    }

    /**
     * Search past articles.
     */
    private static ExtendedEnumeration
    doHistorySearch(NewsChannel channel,
            String pattern)
    throws IOException {
        HistorySearch search = new HistorySearch(pattern);
        search.doReadHeadlines(channel);
        HeadlineAnswer answer = new HeadlineAnswer(channel.receive());
        return answer.getElements();
    }

    /**
     * Establish a connection to the NewsEdge News Server.
     */
    private static InapClient amcConnect(String username, String password,
            String hostname) throws IOException, QuipException

    {
        LoginData login = new LoginData(username, password, "JAPI");
        InetAddress address = InetAddress.getByName(hostname);
        return new InapClient(address, Channel.STARTUP_SERVICE, login);
    }
}
```

# Appendix B: Architectural Overview

The NewsEdge Java API allows you to develop an interface between your application and the NewsEdge News Server, enabling you to connect to the news server (via proxy, port 80 and so on), to search/retrieve metadata and content from news stories, to search news archives, and to launch and filter real-time news feeds.

This appendix describes the architecture of the NewsEdge API library. It is not necessary to read this appendix to be able to program the client application. The appendix is useful to those who are interested in the underlying architecture of the library.

The library includes the following packages:

gari.inap Package – contains the transaction networking layer. Includes the InapClient and LoginData classes.

gari.media Package – contains the API for the media server. Includes the MediaConnection and MediaData classes.

gari.media.iptc Package – provides access to advanced image metadata. Includes the IPTCProfile class and IPTCConstants interface.

gari.net Package – contains the lower-level networking support messaging layer for the library. Includes the QuipEventListener interface, Channel class and various Quip classes.

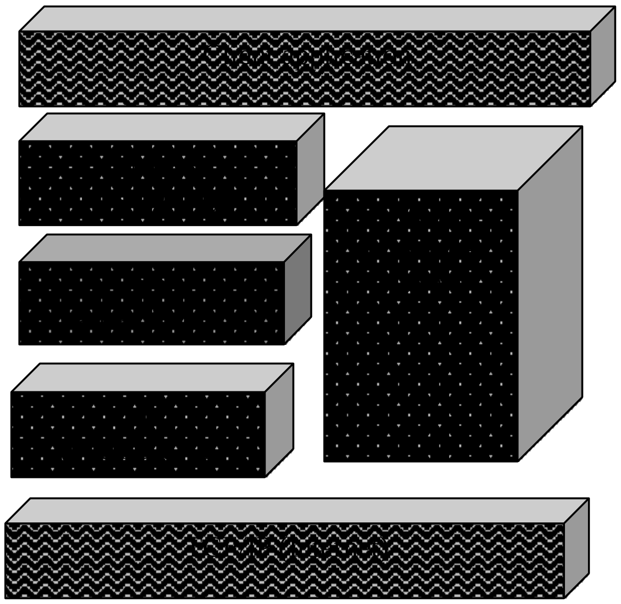
gari.news Package – contains the main application layer. Includes the Headline, HistorySearch, MetaHeadlineQuery, and Profile class (among others).

gari.util package – contains utility classes and methods such as ExtendedEnumeration interface and Base64 class.

For more information on the packages and classes, see “API” (page 45).

## Network Stack

The NewsEdge Java API is a custom networking stack built on top of TCP/IP, as in the following diagram (the lighter parts in the center are NewsEdge):



The main part of the NewsEdge library consists of three layers:

Quip layer, which provides basic packet networking and error handling on top of TCP/IP (see gari.net Package, page 75)

Inap layer, which provides transactional support on top of the Quip layer (see gari.inap Package, page 46)

News layer, which provides full messaging support on top of the Inap layer (see gari.news Package, page 85)

The client application works mainly through the news layer. Note that there is an alternative, single-layer stack for media downloads such as photos, and so on (see gari.media Package, page 50).

The NewsEdge library does not use HTTP, but you can configure it to use port 80 to work with existing firewall rules (see “How to Connect Through Port 80” page 22).

## Messaging Patterns

The NewsEdge library uses two messaging patterns:

Request/response (pull): the client application requests information from the news server and receives a reply (for example: historical news searches, wire information)

Syndication (push): the client application receives information continuously as it becomes available (for example: real-time streaming news)

### Request/Response Operations

Request/response operations typically involve four steps:

1. Construct a request object.
2. Issue the request to the news server.
3. Read the response message from the news server.
4. Construct a response object from the message.

An historical news search (see “How to Search Historical Headlines,” page 27) is an example of a request/response operation, and involves the following steps:

1. Construct a gari.news.HistorySearch object (page 95).
2. Use the HistorySearch doReadHeadlines method (page 98) to issue the request to the news server through a gari.news.NewsChannel.
3. Read the response with the NewsChannel receive method (page 76).
4. Construct a gari.news.HeadlineAnswer object (page 89) from the response message.

## Syndication Operations

Syndication operations are asynchronous, and typically involve three steps:

1. Create a listener object.
2. Register the listener object with the library.
3. Respond to messages the library delivers.

Listener objects are invoked in a separate Java thread, so they do not interfere with the main program flow.

A real-time news syndication (see "How to Start a Real-Time News Feed," page 33) is an example of a syndication operation, and involves the following steps:

1. Construct an object implementing the java.util.Observer interface.
2. Register the observer with the library through a gari.news.ProfileManager object (page 111).

The library invokes the observer's update method (in a separate thread) whenever new news arrives.

# Glossary

| | |
|---|---|
| Component: | One logical part of a package, such as the photo, main story, or sidebar. A component usually has one or more files. |
| Content: | Content is the actual news story, unlike metadata which is information about the news story (such as when it was written, what it is about, when it can be released, and so on). End-users generally see content in print or online, but do not see much, if any, of the metadata. |
| Element: | A pair of tags and the text or nested elements between them in an XML document, such as "`<distributor>AP Online</distributor>`". |
| File: | A physical representation of a component, such as a JPEG file for an image, or an XHTML file for a text story. Note that a component may have more than one alternative file (such as a photo in high and low resolutions, or a text story in XHTML and NITF. Each file represents a different version of the same information. |
| HTML: | The Hypertext Markup Language, a W3C standard for marking up the content of web pages. http://www.w3.org/MarkUp/. |
| IANA: | Internet Assigned Numbers Authority is an organization that oversees global IP address allocation, DNS root zone management, and other Internet protocol assignments. It is operated by ICANN (Internet Corporation for Assigned Names and Numbers (see http://www.iana.org). |
| ISO 639: | A list of standard, two-letter abbreviations for natural languages, such as "en" for English or "ja" for Japanese. http://www.loc.gov/standards/iso639-2/englangn.html |
| ISO 8601: | An international standard for representing dates and times, such as "2006-08-21T04:00:00-04:00" for August 21, 2006 at 4:00 am EDT. http://www.w3.org/TR/NOTE-datetime |

| | |
|---|---|
| IPTC: | The International Press Telecommunications Council, the standards body that maintains NITF and NewsML. http://www.iptc.org/. |
| Log4J: | A highly-configurable free Java logging library available from the Apache Foundation. The Log4J library is required for client applications, since the NewsEdge Java API depends on it. The library is available from http://logging.apache.org/log4j/ |
| Metadata: | Information about the news story, such as when it was written, what it is about, when it can be released, and so on. Content is the news story itself. End-users generally see content in print or online, but do not see much, if any, of the metadata. |
| Mixed content: | XML text, some of which may be tagged using inline elements. http://www.w3.org/TR/REC-xml/#sec-mixed-content |
| Namespace: | A scheme for disambiguating XML names using URLs mapped to prefixes, such as "`xn:`" for "`http://www.xmlnews.org/namespaces/meta#`". The namespace functions similarly to a package in the Java or C++ programming languages. http://www.w3.org/TR/REC-xml-names/ |
| NewsML: | An XML-based format maintained by the IPTC for encoding metadata and packaging information about news stories. http://www.newsml.org/ |
| NITF: | The News Industry Text Format, an XML-based format maintained by the IPTC for encoding the content of news stories. http://www.nitf.org/. |
| Package: | A complete distribution of related components, such as a news story with its accompanying photo and sidebar. |
| W3C: | The World Wide Web Consortium, the main standards body for the web, and maintainer of XML. http://www.w3.org/ |

| | |
|---|---|
| XML: | The Extensible Markup Language, a standard for representing structured, hierarchical information using plain text files. http://www.w3.org/XML/ |

# Index