SIEBEL 7
eBusiness

# SIEBEL EMAIL MARKETING STAND-ALONE API GUIDE

*VERSION 7.5.3, REVISION A*
*MARCH 2004*

Siebel Systems, Inc., 2207 Bridgepointe Parkway, San Mateo, CA 94404

# Contents

## Chapter 3. Heartbeat API



## Chapter 4. Mailing Data API



## Chapter 5. Mailing API



## Chapter 6. Mail Mode API

# Introduction

This manual is designed to help incorporate Siebel Email Marketing Stand-Alone component technology into the user's environment.

**NOTE:** This document may contain references to BoldFish. Products formerly labeled as BoldFish are now known as Siebel Email Marketing Stand-Alone.

Although job titles and duties at your company may differ from those listed in the following table, the audience for this guide consists primarily of employees in these categories:

| | |
|---|---|
| **Siebel Application Administrators** | Administrators who plan, set up, and maintain Siebel applications. |
| **Siebel Application Developers** | Developers who plan, implement, and configure Siebel applications, possibly adding new functionality. |
| **Siebel System Administrators** | Administrators responsible for the whole system, including installing, maintaining, and upgrading Siebel applications. |

# Additional Documentation

This guide does not contain information about upgrading Siebel Email Marketing Stand-Alone. For that information, see *Siebel Email Marketing Stand-Alone Installation Guide* on SupportWeb.

You may also want to consult the other documentation for Siebel Email Marketing Stand-Alone, for example *Siebel Email Marketing Stand-Alone Administration Guide*, and *Siebel Email Marketing Stand-Alone User Guide* on SupportWeb.

Siebel EMSA API uses the log4j logging API. You many want to consult the Jakarta Web site at http://jakarta.apache.org/log4j/docs/ for more information.

# How This Guide Is Organized

- This guide is organized by APIs, error messages, and an example application.
- This guide provides information necessary to monitor Siebel Email Marketing Stand-Alone.

# Documentation Conventions

Throughout this guide, you will see key information identified by the following headings:

**CAUTION:** Indicates information to help you prevent possible harm to your data or network.

**NOTE:** Indicates hints and tips that will help you use Siebel Email Marketing Stand-Alone functionality.

## Typefaces

Table 1 describes the following typefaces are used to highlight specific elements within the manual.

**Table 1. Typefaces**

| Typeface | Usage | Example |
|---|---|---|
| Courier | Text displayed onscreen. | C:\bes > |
| Courier Italics | Text displayed onscreen that varies. | C:*filename.txt* |
| Courier Bold | Text you input. | C:**install** |

| Typeface | Usage | Example |
|---|---|---|
| Courier Bold Italics | Text you input that varies. | C:***directoryname*** |
| *Italics* | Emphasis of new terms. References to titles of books and other publications. | …click *Save* in the *Users* screen. |

# Revision History

*Siebel Email Marketing Stand-Alone API Guide*

## Version 7.5.3, Rev. A

**Table 2. Changes Made in Version 7.5.3, Rev. A**

| Topic | Revision |
|---|---|
| "Log4j Logging API" on page 13 | New chapter. |
| "disconnect" on page 21 | Added new description. |
| "setDebug" on page 21 | Added new description and deleted synopsis and parameter sections. |
| Chapter 3, "Heartbeat API" | Deleted descriptions for Methods for the entire chapter. |
| "newMailingToList - short version" on page 30 | New title, description, and parameters section. |
| "newMailingToList - extended version" on page 32 | New title, description, and parameters section. |
| "newMailingToQuery - short version" on page 34 | New title, description, and parameters section. |
| "newMailingToQuery - extended version" on page 35 | New title, description, and parameters section. |
| List Configuration API, Mailing Control Result, Mailing Status, Message API, Statistics API, Mode Status API, and List Parameters. | Deleted chapters. |

## Additional Changes

Changed the template and format for the entire book.

# 1 Log4j Logging API

Siebel EMSA API uses the log4j logging API. By default there are five levels of logging. Logging requests are enabled for a certain level or higher. A properties file controls the levels enabled. For more information, see the Jakarta Web site at http://jakarta.apache.org/log4j/docs/.

There are five levels of logging preconfigured for use by the log4j API. These are presented in the following list in correct order, where each logging level presented includes all of the levels listed below it, too:

- `DEBUG`—Informational events that are most useful to debug an application.
- `INFO`—Informational messages that highlight the progress of the application.
- `WARN`—Potentially harmful situations.
- `ERROR`—Events that might still allow the application to continue running.
- `FATAL`—Very severe error events that will presumably lead the application to abort.

## Logger Configuration in Siebel EMSA

In the example that follows, `logDir` and `format` are used as variables to hold a string which will be referred to later on in the example.

The root logger is configured to log the `INFO (2)` level and those it includes, for example `WARN`, `ERROR`, and `FATAL`.

`bfapi` is the one appender. `bfapi` is an alias which will be used to refer to properties for a specific appender. For our purposes here, simply consider appenders unique files to which logs are written, each with their own style and log4j level threshold.

The example that follows illustrates how the `bfapi` appender is configured. Notice how the variables `logDir` and `format` are used. Note the setting of the `bfapi` appender threshold to `WARN`, which is less inclusive than the root logger's threshold (of `INFO`). (Setting it to more inclusive, for example `DEBUG`, does not work.)

For more information, see the Jakarta Web site at http://jakarta.apache.org/log4j/docs/.

### Logger Configuration Example

```
# --------------------

# LOGGER CONFIGURATION

# --------------------


# Where the log files should be placed.  By default

# they are put in the application execution directory

#

logDir=.

# Format used for SiebelEMSAAPI.log

format=%d{ISO8601} [%t] %-5p %c{1} %m%n

# The rootLogger

log4j.rootLogger=INFO, bfapi

# Configuration for SiebelEMSAAPI.log

#

log4j.appender.bfapi =org.apache.log4j.RollingFileAppender

log4j.appender.bfapi.File=${logDir}/SiebelEMSAAPI.log

log4j.appender.bfapi.threshold=WARN

log4j.appender.bfapi.MaxFileSize=10MB

log4j.appender.bfapi.MaxBackupIndex=10

log4j.appender.bfapi.layout =org.apache.log4j.PatternLayout
```

```
log4j.appender.bfapi.layout.ConversionPattern=${format}
```

## Using the Logger in Siebel EMSA

In the example that follows, note the use of the Siebel EMSA Logger class. This class wraps a `log4j` Logger object and implements the same logging methods found in that class by calling the wrapped object's methods (for example, `com.boldfish.util.log.Logger.error(Object)` calls the `org.apache.log4j.Logger.error(Object)` method). It also adds some functionality not important for this class. You should use it (not the `log4j` class) in order to make sure that messages logged deep within the Siebel EMSA API libraries are written to your log file.

Note, in the example that follows, the use of the `Logger.info(Object)` method. This writes out an `INFO` level message to all configured appenders that meet the log4j rules for appenders. Here is a list of the log4j methods that would be of interest to you:

- `debug(Object)`—Logs a message object at the `DEBUG` level.
- `debug(Object, Throwable t)`—Logs a message object at the `DEBUG` level with a stack trace of `t`.
- `info(Object)`—Logs a message object at the `INFO` level.
- `info(Object, Throwable t)`—Logs a message object at the `INFO` level with a stack trace of `t`.
- `warn(Object)`—Logs a message object at the `WARN` level.
- `warn(Object, Throwable t)`—Logs a message object at the `WARN` level with a stack trace of `t`.
- `error(Object)`—Logs a message object at the `ERROR` level.
- `error(Object, Throwable t)`—Logs a message object at the `ERROR` level with a stack trace of `t`.
- `fatal(Object)`—Logs a message object at the `FATAL` level.
- `fatal(Object, Throwable t)`—Logs a message object at the `FATAL` level with a stack trace of `t`.

### Using the Logger Example

```
import org.apache.log4j.helpers.FileWatchdog;
import org.apache.log4j.PropertyConfigurator;
import org.apache.log4j.LogManager;
import com.boldfish.util.log.Logger;
public class LoggerExample {
  private static Logger logger
      = Logger.getLogger(LoggerExample.class);
public static void main(String[] args) {
    loggerInit("log4j-config.properties");
  }
private static void loggerInit(String loggerConfigFile) {
    long watchdogDelay = 5 * 1000;
    FileWatchdog watcher
        = new FileWatchdog(loggerConfigFile) {
                public void doOnChange() {
                  new PropertyConfigurator().doConfigure(
                          filename,
                          LogManager.getLoggerRepository());
          logger.info("Re-read configuration file “
                              + filename);
                }
              };
    watcher.setDelay(watchdogDelay);
    watcher.start();
```

```
    }
}
```

# Connection API 2

The `bfConnection` class represents a connection to Siebel Email Marketing Stand-Alone (EMSA). All transactions require that you first connect to the server. After you are finished, you should disconnect from the server.

## Fields

### defaultEncoding

#### Description

The default character encoding for the JVM.

#### Synopsis

```
public static final java.lang.String defaultEncoding
```

## Constructors

### bfConnection

#### Description

Constructor for a connection. You should pass in a valid Siebel Email Marketing Stand-Alone user and password for the transactions you need to perform. For example, if you are trying to create a list, you should pass in a *Site Manager's credentials*. If you are changing list characteristics, you should pass in at least a *List Owner's credentials*.

For example:

```
bfConnection("localhost", 2000,"user@yourcompany.com", "blah");
```

### Synopsis

```
public bfConnection(java.lang.String host,
      int port,
      java.lang.String email,
      java.lang.String password)
   throws  java.lang.Exception
```

### Parameters

`host`—The hostname or IP of the machine that Siebel Email Marketing Stand-Alone is running on.

`port`—The API port number for Siebel Email Marketing Stand-Alone.

`email`—The email address for the user.

`password`—The user's password.

`verbose`—Causes the parameter values passed to this method to be displayed to standard output.

## bfConnection

### Synopsis

```
public bfConnection(java.lang.String host,
      int port,
      java.lang.String email,
      java.lang.String password,
      boolean verbose)
   throws  java.lang.Exception
```

# Methods

## disconnect

### Description

This is necessary to properly close the connection to the Siebel EMSA Server. Failure to do so may result in unexpected behavior on the Siebel EMSA Server. It is proper practice to wait three seconds after disconnecting before terminating the client application. `disconnect` is a good candidate for a final block.

### Synopsis

```
public void disconnect()
      throws java.io.IOException
```

## finalize

### Synopsis

```
public void finalize()
```

### Overrides

`finalize` in class `java.lang.Object`

## setDebug

### Description

Used by Siebel Professional and Global Services for troubleshooting.

# Heartbeat API 3

The bfHeartbeat class provides a way to talk to the Siebel Email Marketing Stand-Alone (EMSA) to confirm that it is properly servicing Siebel EMSA API connections.

## Constructor

### bfHeartbeat

#### Description

Constructor for a bfHeartbeat. You should pass in a valid Siebel EMSA bfConnection.

For example:

```
bfConnection("localhost", 2000,"user@yourcompany.com", "blah");
```

#### Synopsis

```
public bfHeartbeat(bfConnection c)
```

#### Parameters

c—a bfConnection

# Methods

## ping

### Description

Ping is a call to see if the server is accepting Siebel EMSA API calls. This method allows a user to *ping* the Siebel EMSA Server to validate that the server is running.

### Synopsis

```
public boolean ping()
   throws java.lang.Exception
```

# Mailing Data API 4

The bfMailingData class is used to represent a mailing. It allows you to specify different individuals for different mailings. After you create a bfMailingData object, pass it to an instantiated bfMailing object.

## Constructor

### bfMailingData

#### Synopsis

```
public bfMailingData()
```

## Methods

### addSubMailing

#### Description

Adds a new addSubMailing to the Mailing. This addSubMailing method is used with mailings using queries. Call this once for each Mail Mode used in mailing to associate a content file with a particular Mail Mode.

#### Synopsis

```
public void addSubMailing(int mode,
      java.lang.Object content,
      java.lang.String contentType,
```

```
        java.lang.String columnNames)
    throws com.boldfish.netapi.BadParameterException
```

### Parameters

`mode`—The Mail Mode for this `addSubMailing`. See `bfMailMode`.

`content`—Points to the content. Valid types are: `String`, `File`, `byte[]`, `Enumeration`, `InputStream` or `DataSource`.

`contentType`—The MIME content type for this content.

`columnNames`—The Mail Merge Variable names.

## addSubMailing

### Description

Adds a new `addSubMailing` to the `Mailing`. Each `addSubMailing` has a content-type and associated subscribers.

### Synopsis

```
public void addSubMailing(int mode,
        java.lang.Object contentObj,
        java.lang.String contentType,
        java.lang.String columnNames,
        java.lang.Object subscribers)
    throws com.boldfish.netapi.BadParameterException
```

### Parameters

`mode`—The Mail Mode for this `addSubMailing`. See `bfMailMode`.

`content`—Points to the content. Valid types are: `String`, `File`, `byte[]`, `Enumeration`, or `InputStream`.

`contentType`—The MIME content type for this content.

columnNames—The ordering of the content.

subscribers—The subscriber email addresses and possibly the Mail Merge Variable Values. Valid types are: String, File, byte[], Enumeration, or InputStream.

## addSubMailing

### Description

Adds a new addSubMailing to the Mailing. Each addSubMailing has a content-type and associated subscribers. In this call, the subscribers are specified and the system-stored subscribers will not be used.

### Synopsis

```
public void addSubMailing(int mode,
      java.lang.Object contentObj,
      java.lang.String contentType,
      java.lang.String columnNames,
      java.lang.Object subscribersObj,
      java.util.Date deliverBy)
   throws com.boldfish.netapi.BadParameterException
```

### Parameters

mode—The Mail Mode for this addSubMailing. See Mail Mode API.

content—Points to the content. Valid types are: String, File, byte[], Enumeration, or InputStream.

contentType—The MIME content type for this content.

columnNames—The ordering of the content.

subscribers—Points to the subscribers. Valid types are: String, File, byte[], Enumeration, or InputStream.

`deliverBy`—Expiration date given to the mailing. If the sending date is greater than `deliverBy`, then the mailing is canceled.

## addAttachment

### Description

Adds a new attachment to the `Mailing`.

### Synopsis

```
public void addAttachment(
      java.lang.String contentType,
      java.lang.Object contentObj,
      java.lang.String name)
   throws com.boldfish.netapi.BadParameterException
```

### Parameters

`contentType`—The MIME content type for this content.

`content`—Points to the content. Valid types are: `String`, `File`, `byte[]`, `Enumeration`, or `InputStream`.

`name`—The name of the attachment.

## getHasSubscribers

### Synopsis

```
public boolean getHasSubscribers()
```

# Mailing API 5

The `bfMailing` class allows you to create a mailing to a segment of email addresses uploaded from a flat file or retrieved from the database by an SQL query. Additionally, this class allows you to supply various email headers to be used in the mailing.

In order to call any of this class's methods, you will need to create a `bfMailingData` object. The `bfMailingData` object contains the mailing content, attachments, and an optional, uploaded email address segment.

## Constructor

### bfMailing

#### Description

Constructor for a mailing. You should pass in a valid Siebel Email Marketing Stand-Alone `bfConnection` object.

#### Synopsis

```
public bfMailing(bfConnection c)
```

#### Parameters

c—A `bfConnection` object

# Methods

## newMailingToList - short version

### Description

Sends out a new mailing. This method initiates a mailing to a segment of email addresses uploaded in a `bfMailingData` object. Use the same `bfMailingData` object to specify the necessary Mail Modes and their corresponding content. This call generates a `messageID` for you. The appearance of the email is specified in Siebel Email Marketing Stand-Alone configuration settings.

### Synopsis

```
public bfMailingControl newMailingToList(
        java.lang.String listName,
        long date,
        java.lang.String campaignID,
        java.lang.String subject,
        bfMailingData content,
        java.lang.String charSet
        boolean useHUB)
     throws com.boldfish.netapi.BadParameterException,
        java.io.IOException,
        lwd.BadVersionException
```

### Parameters

`listName`—The name of a list to use for its predefined settings. The email recipients are passed to the server through the `content` parameter using a `bfMailingData` object.

`date`—Specifies the date and time on which the email campaign is to be started.

---

**NOTE:** The server's system clock is compared to this object's value. If the server's system clock is running later than the client's system clock and if the client's current system time is used in constructing this object, the mailing will be delayed by the difference in time between the server's and client's clocks.

---

`campaignID`—Used to uniquely specify a campaign for a list. Helpful when viewing reports.

`subject`—The Subject header for the email.

---

**NOTE:** The Subject header will be altered if the list setting "Subject: Header Line" is set to something other than "Preserve as is" or if a custom Subject header is passed in the body of the email.

---

`content`—A `bfMailingData` object containing the content, attachments, and recipients to upload to the Siebel EMSA Server.

`charSet`—The character encoding (character set) applied to the outgoing email message body.

---

**NOTE:** The US-ASCII character encoding is applied to the email message headers.

---

**CAUTION:** When characters are converted from a specific character set to a different one within Siebel EMSA, characters which cannot be converted are replaced with a "?". This is a feature of Java.

---

`useHUB`—A deprecated feature. Set to false.

# newMailingToList - extended version

## Description

Sends out a new mailing. This method initiates a mailing to a segment of email addresses uploaded in a `bfMailingData` object. Use the same `bfMailingData` object to specify the necessary Mail Modes and their corresponding content. The appearance of the email is specified by the parameters passed in this method call, as well as the Siebel Email Marketing Stand-Alone configuration settings. In order to set the ReplyTo, From, and To headers using this method, the list must be configured to preserve these header values.

## Synopsis

```
public bfMailingControl newMailingToList(
        java.lang.String listName,
        long date,
        java.lang.String campaignID,
        java.lang.String to,
        boolean toIsPersonalized,
        java.lang.String from,
        java.lang.String sender,
        java.lang.String replyTo,
        java.lang.String subject,
        java.lang.String messageID,
        bfMailingData content,
        java.lang.String charSet,
        boolean useHUB)
    throws com.boldfish.netapi.BadParameterException,
        java.io.IOException,
```

```
lwd.BadVersionException
```

## Parameters

`listName`—The name of the list whose appearance is to be used.

`date`—Specifies the date and time on which the email campaign is to be started.

`campaignID`—Used to uniquely specify a campaign for a list. Helpful when viewing reports.

`to`—The email message To header. Specify a "" if you want to use the appearance settings of the list.

`toIsPersonalized`—Set to `true` if you want To header to be personalized.

`from`—The email message From header. Specify a "" if you want to use the appearance settings of the list.

`sender`—The email message Sender header. Specify a "" if you want to use the appearance settings of the list.

---

**NOTE:** Does not alter the SMTP envelope Sender (which bounces are sent to).

---

`messageID`—Controls the email message `MessageID` header in the following way:

- When not specified (or when using the short version of the `newMailingToList` method), the value generated will be unique per email sent out and in the format, `<AES459196.XXX@ServerHostName>`, where XXX is a counter reset each time the Siebel EMSA is restarted.
- When this parameter is specified, the system generates a unique `MessageID` header of the format, `<AES459196.XXX.YYY>`, where YYY is the value passed to this parameter.

`content`—The `bfmailingData` object containing the content, attachments, and recipients to upload to the Siebel EMSA Server.

charSet—The character encoding (character set) applied to the outgoing email message body.

**NOTE:** The US-ASCII character encoding is applied to the email message headers.

**CAUTION:** When characters are converted from a specific character set to a different one within Siebel EMSA, characters which cannot be converted are replaced with a "?". This is a feature of Java.

useHUB—A deprecated feature. Set to false.

## newMailingToQuery - short version

### Description

This method initiates a mailing to a segment of email addresses retrieved from a database using a SQL select statement (referred to by Siebel EMSA as a *query*). A bfMailingData object is used to specify the necessary Mail Modes and corresponding content. The appearance of the email is specified in Siebel Email Marketing Stand-Alone configuration settings.

### Synopsis

```
public bfMailingControl newMailingToQuery(
        java.lang.String queryName,
        long date,
        java.lang.String campaignID,
        java.lang.String subject,
        bfMailingData content,
        java.lang.String charSet,
        boolean useHUB)
    throws com.boldfish.netapi.BadParameterException,
```

```
java.io.IOException,
lwd.BadVersionException
```

### Parameters

`queryName`—Name of the predefined query used to access the database and retrieve the email Subscribers and Mail Merge Variable Values. The Mail Merge Variables are the SQL statement column names. The query is associated with a list in the Siebel EMSA GUI, and it is from this list that the appearance settings are retrieved.

`date`—Specifies the date when to start sending the email.

`campaignID`—Used to specify a campaign.

`subject`—The Subject header of the email.

`content`—The `bfMailingData` object containing the content and attachments to upload to the Siebel EMSA Server.

`charSet`—The character encoding (character set) applied to the outgoing email message body.

**NOTE:** The US-ASCII character encoding is applied to the email message headers.

**CAUTION:** When characters are converted from a specific character set to a different one within Siebel EMSA, characters which cannot be converted are replaced with a "?". This is a feature of Java.

`useHUB`—A deprecated feature. Set to false.

## newMailingToQuery - extended version

### Description

This method initiates a mailing to a segment of email addresses retrieved from a database using a SQL select statement (referred to by Siebel EMSA as a *query*). A `bfMailingData` object is used to specify the necessary Mail Modes and corresponding content. The appearance of the email is specified by the parameters passed in this method call as well as the Siebel Email Marketing Stand-Alone configuration settings.

## Synopsis

```
public bfMailingControl newMailingToQuery(
        java.lang.String queryName,
        long date,
        java.lang.String campaignID,
        java.lang.String to,
        boolean toIsPersonalized,
        java.lang.String from,
        java.lang.String sender,
        java.lang.String replyTo,
        java.lang.String subject,
        java.lang.String messageID,
        bfMailingData content,
        java.lang.String charSet,
        boolean useHUB)
    throws com.boldfish.netapi.BadParameterException,
        java.io.IOException,
        lwd.BadVersionException
```

## Parameters

`queryName`—Name of the predefined query used to access the database and retrieve the email Subscribers and Mail Merge Variable Values. The Mail Merge Variables are the SQL statement column names. The query is associated with a list in the Siebel EMSA GUI, and it is from this list that unspecified appearance settings are retrieved.

`date`—Specifies the date and time on which the email campaign is to be started.

`campaignID`—Used to specify a campaign for a list. Helpful when viewing reports.

`to`—The email message To header. Use `""` if you wish to use the appearance settings of the list.

`toIsPersonalized`—Set to true if you want the To header to be personalized. Use `""` if you wish to use the appearance settings of the list.

`from`—The email message From header. Use `""` if you wish to use the appearance settings of the list.

`sender`—The email message Sender header. Use `""` if you wish to use the appearance settings of the list.

---

**NOTE:** Does not alter the SMTP envelope Sender (which bounces are sent to).

---

`replyTo`—The email message ReplyTo header. Use `""` if you wish to use the appearance settings of the list.

`subject`—The Subject header for the email.

`messageID`—Controls the email message `MessageID` header in the following ways:

- When not specified (or when using the short version of the `newMailingToList` method), the value generated will be unique per email sent out and in the format, `<AES459196.XXX@ServerHostName>`, where `XXX` is a counter reset each time the Siebel EMSA is restarted.
- When this parameter is specified, the system generates a unique `MessageID` header of the format, `<AES459196.XXX.YYY>`, where `YYY` is the value passed to this parameter.

`content`—The `bfMailingData` object containing the content and attachments to upload to the EMSA Server.

`charSet`—The character encoding (character set) applied to the outgoing email message body.

**NOTE:** The US-ASCII character encoding is applied to the email message headers.

**CAUTION:** When characters are converted from a specific character set to a different one within Siebel EMSA, characters which cannot be converted are replaced with a "?". This is a feature of Java.

`useHUB`—A deprecated feature. Set to false.

# Mail Mode API 6

This chapter includes predefined constants for specifying Mail Modes used with the Siebel Email Marketing Stand-Alone (EMSA) API.

## Constructor

### bfMailMode

#### Synopsis

```
public bfMailMode()
```

## Fields

### DONT_CARE

#### Synopsis

```
public static final int DONT_CARE
```

### MAILMODE_ALL

#### Synopsis

```
public static final int MAILMODE_ALL
```

## MAILMODE_AOL

### Synopsis

```
public static final int MAILMODE_AOL
```

## MAILMODE_HTML

### Synopsis

```
public static final int MAILMODE_HTML
```

## MAILMODE_TEXT

### Synopsis

```
public static final int MAILMODE_TEXT
```

## MAILMODE_TEXTANDHTML

### Synopsis

```
public static final int MAILMODE_TEXTANDHTML
```

## MAILMODE_WIRELESS

### Synopsis

```
public static final int MAILMODE_WIRELESS
```

# Data Source API 7

`DataSource` is an abstract class for holding any source of bytes and the encoding (as in UTF8, EUC_JP, and so on) that those bytes are encoded with.

## Constructor

### DataSource

#### Synopsis

```
public DataSource()
```

## Methods

### asInputStream

#### Description

Returns an `InputStream` for reading the bytes in this Source.

#### Synopsis

```
public abstract java.io.InputStream asInputStream()
   throws java.io.IOException
```

### getEncoding

#### Description
Returns the name of the encoding used to produce the bytes in Source.

#### Synopsis
```
public abstract java.lang.String getEncoding()
```

## Sub Classes

### DataSource.FromByteArray

`DataSource.FromByteArray` is a subclass of `DataSource` that gets the bytes from a byte array.

#### Constructor
**DataSource.FromByteArray**

#### Synopsis
```
public DataSource.FromByteArray(byte[] bytes,
        java.lang.String encoding)
```

#### Methods
**asInputStream**

#### Description
Returns an `InputStream` for reading the bytes in this Source.

#### Synopsis
```
public java.io.InputStream asInputStream()
```

#### getEncoding

### Description
Returns the name of the encoding used to produce the bytes in Source.

### Synopsis
```
public java.lang.String getEncoding()
```

## DataSource.FromEnumeration

`DataSource.FromEnumeration` is a subclass of `DataSource` that gets the bytes by applying `.toString` to each element of an Enumeration.

### Constructor
#### DataSource.FromEnumeration

### Synopsis
```
public DataSource.FromEnumeration(java.util.Enumeration enum)
```

### Methods
#### asInputStream

### Description
Returns an `InputStream` for reading the bytes in this Source.

### Synopsis
```
public java.io.InputStream asInputStream()
```

#### getEncoding

### Description
Returns the name of the encoding used to produce the bytes in Source.

### Synopsis

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

## DataSource.FromFile

`DataSource.FromFile` is a subclass of `DataSource` that gets the bytes from a file.

### Constructor

#### DataSource.FromFile

### Synopsis

```
public DataSource.FromFile(java.io.File file,
      java.lang.String encoding)
```

### Methods

#### asInputStream

### Description

Returns an `InputStream` for reading the bytes in this Source.

### Synopsis

```
public java.io.InputStream asInputStream()
      throws java.io.IOException
```

#### getEncoding

### Description

Returns the name of the encoding used to produce the bytes in Source.

### Synopsis

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

## DataSource.FromInputStream

`DataSource.FromInputStream` is a subclass of `DataSource` that gets the bytes from an arbitrary `InputStream`.

### Constructor

**DataSource.FromInputStream**

### Synopsis

```
public DataSource.FromInputStream(java.io.InputStream in,
      java.lang.String encoding)
```

### Methods

**asInputStream**

### Description

Returns an `InputStream` for reading the bytes in this Source.

### Synopsis

```
public java.io.InputStream asInputStream()
```

**getEncoding**

### Description

Returns the name of the encoding used to produce the bytes in Source.

### Synopsis

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

## DataSource.FromString

`DataSource.FromString` is a subclass of `DataSource` that gets the bytes from a String.

## Constructor

### DataSource.FromString

## Synopsis

```
public DataSource.FromString(java.lang.String str)
```

## Methods

### asInputStream

## Description

Returns an `InputStream` for reading the bytes in this Source.

## Synopsis

```
public java.io.InputStream asInputStream()
      throws java.io.IOException
```

### getEncoding

## Description

Returns the name of the encoding used to produce the bytes in Source.

## Synopsis

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

### toString

## Synopsis

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

# Error Messages 8

The `BadParameterException` is meant to contain information about any errors that might be encountered on the server side of the communication. If you see this exception, print out its value by means of `getMessage()`.

## Constructor

### BadParameterException

#### Synopsis

```
public BadParameterException(java.lang.String s)
```

*Constructor*

# An Example Application 9

The APISample1 class implements a class that will send an email. This sample application allows you to use Siebel Email Marketing Stand-Alone (EMSA) network API protocol to send an email to users who are subscribed to an internal list and have specified TEXT as their default Mail Mode. The sample application will also send an attachment.

To compile and run, add common.jar, log4.jar, and netapi.jar to your classpath. They can be found in the Siebel EMSA classes directory.

```
USAGE: java APISample1 [-H host] [-P port] [-l login] [-p password]
```

There are four basic steps to sending email.

***To send email***

1. Create a new connection to the server:

   ```
   c = new bfConnection(p.getProperty("-H", "localhost"),
      Integer.parseInt(p.getProperty("-P", "2000")),
      p.getProperty("-l", "user@yourcompany.com"),
      p.getProperty("-P", "trivial"),
      true);
   ```

   This call requires a valid login and password to the server.

2. Get a bfMailing and bfMailingData object:

   ```
   bfMailing flight = new bfMailing(c);
   bfMailingData fd = new bfMailingData();
   ```

   These classes will allow you to construct your content.

3. Associate the content you want to send with the Mail Mode:

```
fd.addSubMailing(bfMailMode.MAILMODE_TEXT, test, fd.TEXT,
“EmailAddr”);
```

In this case we are adding a `String` and sending to the `TEXT` users. The content type of the `String` is specified as `text/plain`.

4 Send the mailing:

```
flight.newMailingToList(“test”, new Date().getTime(),

   “,

   “Phil’s birthday”,

   fd,

   “us-ascii”,

   false);
```

## APISample1

```
   public class APISample1 {

      public static Properties parseArgs(String arg[], String
      parseFor[]) {

      Properties p=new Properties();

for (int i=0; i < arg.length; i++)

   for (int j=0; j < parseFor.length; j++)

   if (arg[i].equals(parseFor[j]))

   if (arg.length > i+1) {

      p.put(parseFor[j], arg[i+1]);

      continue;

      }

   return p;

   }

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

```
String content = "Dear Bill, [$EmailAddr]\r\n\r\nIt has been many
years since we have spoken and I want to invite you to a friend's
house. We are celebrating the 26th birthday of their dog named Phil.
\r\n\r\n Phil is an Alaskan Husky and likes the outdoors. We feel
especially privileged to attend since Phil does not like strangers.
So make sure that you wear some thick clothing. Phil is bound to take
a bite or two. \r\n\r\n Lately Phil has a new girlfriend and so we're
bringing her a new shower and tub. Well actually, it is a gift
certificate for a contractor to build an addition to the house. It'll
really be awesome when it's in.\r\n\r\n Well, enough about the
rambling. Make sure you don't get anything dinky or Phil will be sure
to take a bite out of you. See you next year at the party.\r\n";

   String opts[] = {"-H", "-P", "-l", "-p", "-d"};

   Properties p = parseArgs(arg, opts);

   bfConnection c;

   // Connect to the server

   c = new bfConnection(p.getProperty("-H", "localhost"),

        Integer.parseInt(p.getProperty("-P", "2000")),

        p.getProperty("-l", "n@yourcompany.com"),

        p.getProperty("-p", "trivial"),

        true);

   c.setDebug( Integer.parseInt(p.getProperty("-d", "0")));

   // Get a mailing object

   bfMailing flight = new bfMailing(c);

   bfMailingData fd = new bfMailingData();

   // Associate this content with TEXT subscribers

   fd.addSubMailing(bfMailMode.MAILMODE_TEXT, // Subscriber type

      content, // Body of email

      fd.TEXT, // content-type: text/plain

      "EmailAddr"); // mail-merge variables
```

```
// Add an attachment for good measure
fd.addAttachment("image/gif", new File("phil.gif"), "phil.gif");
// send off the mailing
flight.newMailingToList("test", new Date().getTime(),
     "",
     "Phil's birthday",
     fd,
     "us-ascii",
     false);
c.disconnect();
}
}
```

# Index

## N



## P



## Q



## R



## S



## T



## U