



# TIBCO Rendezvous®

## Java Reference

Version 8.7.0 | October 2023

# Contents

<b>Concepts</b>	<b>12</b>
Implementations	13
Strings and Character Encodings	15
Interrupting Event Dispatch Threads	17
Network Overview—rxd Transports	18
Intranet Web Site Considerations	20
Rendezvous Files	20
JNI Shared Libraries	20
rxd	20
<b>Programmer's Checklist</b>	<b>21</b>
Checklist	22
Install	22
Code	22
Compile	22
Run	23
Archive Files	24
Shared Library Files	26
IPM Library	26
<b>Rendezvous Environment</b>	<b>28</b>
Tibrv	29
Tibrv.close()	31
Tibrv.defaultQueue()	33
Tibrv.getErrorCallback()	34
Tibrv.getVersion()	35
Tibrv.isIPM()	36
Tibrv.isNativeImpl()	37
Tibrv.isValid()	38
Tibrv.open()	39

Tibrv.processTransport()	42
Tibrv.setErrorCallback()	43
Tibrv.setRVParameters()	44
TibrvSdContext	46
TibrvSdContext.setDaemonCert()	48
TibrvSdContext.setUserCertWithKey()	50
TibrvSdContext.setUserCertWithKeyBin()	52
TibrvSdContext.setUserNameWithPassword()	54
<b>Data</b>	<b>55</b>
Field Names and Field Identifiers	56
Finding a Field Instance	57
TibrvMsg	58
TibrvMsg()	64
TibrvMsg.add()	66
Add Scalar	70
TibrvMsg.addField()	73
TibrvMsg.dispose()	74
TibrvMsg.get()	75
Get Scalar	79
TibrvMsg.getAsBytes()	81
TibrvMsg.getByteSize()	82
TibrvMsg.getEvent()	83
TibrvMsg.getField()	84
TibrvMsg.getFieldByIndex()	86
TibrvMsg.getFieldInstance()	87
TibrvMsg.getNumFields()	89
TibrvMsg.getReplySubject()	90
TibrvMsg.getSendSubject()	91
TibrvMsg.getStringEncoding()	92
TibrvMsg.getTypeName()	93

TibrvMsg.removeField()	94
TibrvMsg.removeFieldInstance()	96
TibrvMsg.reset()	97
TibrvMsg.setReplySubject()	98
TibrvMsg.setSendSubject()	99
TibrvMsg.setStringEncoding()	100
TibrvMsg.toString()	102
TibrvMsg.update()	103
Update Scalar	106
TibrvMsg.updateField()	108
TibrvMsgField	109
TibrvMsgField()	112
TibrvMsgField.toString()	114
TibrvMsgView	115
TibrvMsgView.extract()	118
TibrvMsgView.get()	120
Get Scalar from TibrvMsgView	122
TibrvMsgView.getField()	124
TibrvMsgView.getFieldByIndex()	126
TibrvMsgView.getFieldInstance()	127
TibrvMsgView.getNumFields()	129
TibrvDate	130
TibrvDate()	134
TibrvDate.getTimeNanoseconds()	136
TibrvDate.getTimeSeconds()	137
TibrvDate.setTime()	138
TibrvIPAddr	140
TibrvIPAddr()	142
TibrvIPAddr.getAddr()	144
TibrvIPAddr.getAsBytes()	145
TibrvIPAddr.getAsString()	146

TibrvIPPort .....	147
TibrvIPPort() .....	149
TibrvIPPort.getPort() .....	150
TibrvIPPort.getAsBytes() .....	151
TibrvXml .....	152
TibrvXml() .....	154
TibrvXml.getBytes() .....	155
<b>Events and Queues .....</b>	<b>156</b>
TibrvEvent .....	157
TibrvEvent.destroy() .....	159
TibrvEvent.getClosure() .....	160
TibrvEvent.getQueue() .....	161
TibrvEvent.isValid() .....	162
TibrvEvent.isVectorListener() .....	163
TibrvListener .....	164
TibrvListener() .....	167
TibrvListener.getSubject() .....	169
TibrvListener.getTransport() .....	170
TibrvMsgCallback .....	171
TibrvMsgCallback.onMsg() .....	172
TibrvVectorListener .....	174
TibrvVectorListener() .....	176
TibrvVectorListener.getSubject() .....	181
TibrvVectorListener.getTransport() .....	182
TibrvVectorCallback .....	183
TibrvVectorCallback.onMsgs() .....	184
TibrvTimer .....	186
TibrvTimer() .....	189
TibrvTimer.getInterval() .....	191
TibrvTimer.resetInterval() .....	192

TibrvTimerCallback .....	194
TibrvTimerCallback.onTimer() .....	195
TibrvDispatchable .....	196
TibrvDispatchable.dispatch() .....	197
TibrvDispatchable.poll() .....	198
TibrvDispatchable.timedDispatch() .....	199
TibrvQueue .....	201
TibrvQueue() .....	204
TibrvQueue.destroy() .....	205
TibrvQueue.dispatch() .....	206
TibrvQueue.getCount() .....	208
TibrvQueue.getDiscardAmount() .....	209
TibrvQueue.getLimitPolicy() .....	210
TibrvQueue.getMaxEvents() .....	211
TibrvQueue.getName() .....	212
TibrvQueue.getPriority() .....	213
TibrvQueue.isDefault() .....	214
TibrvQueue.isValid() .....	215
TibrvQueue.poll() .....	216
TibrvQueue.setName() .....	217
TibrvQueue.setLimitPolicy() .....	218
TibrvQueue.setPriority() .....	220
TibrvQueue.timedDispatch() .....	221
TibrvQueueGroup .....	223
TibrvQueueGroup() .....	225
TibrvQueueGroup.add() .....	226
TibrvQueueGroup.contains() .....	227
TibrvQueueGroup.destroy() .....	228
TibrvQueueGroup.dispatch() .....	229
TibrvQueueGroup.elements() .....	230
TibrvQueueGroup.getCount() .....	231

TibrvQueueGroup.isValid()	232
TibrvQueueGroup.poll()	233
TibrvQueueGroup.remove()	234
TibrvQueueGroup.timedDispatch()	235
TibrvDispatcher	237
TibrvDispatcher()	240
<b>Transports</b>	<b>242</b>
TibrvTransport	243
TibrvTransport.createInbox()	246
TibrvTransport.destroy()	248
TibrvTransport.isValid()	249
TibrvTransport.getDescription()	250
TibrvTransport.requestReliability()	251
TibrvTransport.send()	253
TibrvTransport.sendReply()	254
TibrvTransport.sendRequest()	256
TibrvTransport.setBatchSize()	258
TibrvTransport.setDescription()	260
TibrvProcessTransport	261
TibrvNetTransport	263
TibrvRvdTransport	265
TibrvRvdTransport()	268
TibrvRvdTransport.getDaemon()	271
TibrvRvdTransport.getNetwork()	272
TibrvRvdTransport.getService()	273
TibrvRvdTransport.setBatchMode()	274
<b>Virtual Circuits</b>	<b>276</b>
TibrvVcTransport	277
TibrvVcTransport.createAcceptVc()	280

TibrvVcTransport.createConnectVc()	282
TibrvVcTransport.getConnectSubject()	284
TibrvVcTransport.waitForVcConnection()	285
<b>Fault Tolerance</b>	<b>287</b>
Fault Tolerance Road Map	288
TibrvFtMember	289
TibrvFtMember()	293
TibrvFtMember.destroy()	297
TibrvFtMember.getActivationInterval()	298
TibrvFtMember.getActiveGoal()	299
TibrvFtMember.getGroupName()	300
TibrvFtMember.getHeartbeatInterval()	301
TibrvFtMember.getPreparationInterval()	302
TibrvFtMember.getQueue()	303
TibrvFtMember.getTransport()	304
TibrvFtMember.getWeight()	305
TibrvFtMember.isValid()	306
TibrvFtMember.setWeight()	307
TibrvFtMemberCallback	309
TibrvFtMemberCallback.onFtAction()	310
TibrvFtMonitor	312
TibrvFtMonitor()	314
TibrvFtMonitor.destroy()	316
TibrvFtMonitor.getGroupName()	317
TibrvFtMonitor.getTransport()	318
TibrvFtMonitorCallback	319
TibrvFtMonitorCallback.onFtMonitor()	320
<b>Certified Message Delivery</b>	<b>321</b>
TibrvCmListener	322



TibrvCmListener()	324
TibrvCmListener.destroy()	326
TibrvCmListener.confirmMsg()	327
TibrvCmListener.isExplicitConfirm()	329
TibrvCmListener.setExplicitConfirm()	330
TibrvCmTransport	331
TibrvCmTransport()	335
TibrvCmTransport.addListener()	339
TibrvCmTransport.allowListener()	341
TibrvCmTransport.destroy()	342
TibrvCmTransport.disallowListener()	343
TibrvCmTransport.expireMessages()	345
TibrvCmTransport.getDefaultTimeLimit()	347
TibrvCmTransport.getLedgerName()	348
TibrvCmTransport.getName()	349
TibrvCmTransport.getRequestOld()	350
TibrvCmTransport.getSyncLedger()	351
TibrvCmTransport.getTransport()	352
TibrvCmTransport.removeListener()	353
TibrvCmTransport.removeSendState()	355
TibrvCmTransport.reviewLedger()	357
TibrvCmTransport.send()	359
TibrvCmTransport.sendReply()	361
TibrvCmTransport.sendRequest()	363
TibrvCmTransport.setDefaultTimeLimit()	365
TibrvCmTransport.setPublisherInactivityDiscardInterval()	366
TibrvCmTransport.syncLedger()	368
TibrvCmReviewCallback	369
TibrvCmReviewCallback.onLedgerMsg()	370
TibrvCmMsg	372
TibrvCmMsg.getSender()	374

TibrvCmMsg.getSequence() .....	375
TibrvCmMsg.getTimeLimit() .....	377
TibrvCmMsg.setTimeLimit() .....	378
<b>Distributed Queue .....</b>	<b>380</b>
TibrvCmQueueTransport .....	381
TibrvCmQueueTransport() .....	386
TibrvCmQueueTransport.destroy() .....	390
TibrvCmQueueTransport.getCompleteTime() .....	391
TibrvCmQueueTransport.getUnassignedMessageCount() .....	392
TibrvCmQueueTransport.getWorkerWeight() .....	393
TibrvCmQueueTransport.getWorkerTasks() .....	394
TibrvCmQueueTransport.setCompleteTime() .....	395
TibrvCmQueueTransport.setTaskBacklogLimit...() .....	396
TibrvCmQueueTransport.setWorkerWeight() .....	398
TibrvCmQueueTransport.setWorkerTasks() .....	399
<b>Exceptions and Errors .....</b>	<b>401</b>
TibrvException .....	402
TibrvException.printStackTrace() .....	404
TibrvStatus .....	405
TibrvErrorCallback .....	411
TibrvErrorCallback.onError() .....	412
<b>Custom Datatypes .....</b>	<b>414</b>
TibrvMsg.getDecoder() .....	415
TibrvMsg.getEncoder() .....	416
TibrvMsg.setHandlers() .....	417
TibrvMsgDecoder .....	419
TibrvMsgDecoder.decode() .....	420
TibrvMsgEncoder .....	421
TibrvMsgEncoder.canEncode() .....	422

<a href="#">TibrvMsgEncoder.encode()</a> .....	423
<a href="#">TIBCO Documentation and Support Services</a> .....	424
<a href="#">Legal and Third-Party Notices</a> .....	427

# Concepts

---

This section presents concepts specific to the TIBCO Rendezvous® Java language interface. For concepts that pertain to Rendezvous software in general, see the book *TIBCO Rendezvous Concepts*.

# Implementations

Rendezvous Java language interface offers several implementations. [Implementations](#) summarizes and compares them. To specify an implementation, see [Archive Files](#), and [Tibrv.open\(\)](#).

## Implementations

Aspect	JNI (Native) Preferred	JNI (Native) Backward Compatibility
Suggested Usage	Use in most situations.	Use only for backward compatibility, when it is impractical to migrate to the JNI preferred implementation.
Coding	A thin Java layer serves as a programming interface to an underlying C implementation.	
Primary Runtime Environment	Independent application	Independent application or browser applet
Scope of Functionality	Maximal coverage of C functionality. Exception: I/O events	
Speed	Faster for all receiving applications—whether they access all fields or only a small subset of fields from inbound messages.	Slower for all receiving applications.  Retained for backward compatibility only.
Transport Support	rvd and intra-process transports	rvd and intra-process transports
Message Storage	Messages exist both in Java (garbage-collected)	

Aspect	JNI (Native) Preferred	JNI (Native) Backward Compatibility
	storage and in native C (JNI) storage.	
Timer Granularity	Supports fine-resolution timers (within operating system limitations)	
Forward Feature Support	In release 8.0 and later, only this implementation supports new features; for example, sending message arrays, vector listeners, dispose.	None
Library Requirements	Must load the Rendezvous JNI library	

# Strings and Character Encodings

Rendezvous software uses strings in several roles:

- String data inside message fields
- Field names
- Subject names (and other *associated* strings that are not strictly *inside* the message)
- Certified delivery (CM) correspondent names
- Group names (fault tolerance)

## Encodings and Translation

Java programs represent all these strings in the Unicode 2-byte character set.

- Before sending an outbound message, Rendezvous software translates these strings into the character encoding appropriate to the ISO locale.
- Conversely, when extracting a string from an inbound message, Rendezvous software translates it to Unicode.

Rendezvous translates its strings *as if* the message used the default encoding (see [Default Encoding](#), below). This assumption is not always correct (see [Inbound Translation](#), below).

## Default Encoding

The default encoding depends on the locale where Java is running. That is, the locale determines the value of the Java system property `file.encoding`, which in turn determines the translation scheme.

For example, the United States is locale `en_US`, and uses the Latin-1 character encoding (also called ISO 8859-1); Japan is locale `ja_JP`, and uses the Shift-JIS character encoding.

When the system property `file.encoding` is inaccessible, the default encoding is 8859-1 (Latin-1). Programs can override this system property; for details, see [TibrvMsg.setStringEncoding\(\)](#).

**Note**

Some browsers (for example, Microsoft Internet Explorer) do not permit programs to access the system property `file.encoding`. When programs attempt to access it, the browser throws a `SecurityException`. Although this is normal, and the program continues to run, the browser may nonetheless print a stack trace, indicating that the program cannot access that system property.

## Outbound Translation

Outbound translation from Unicode to a specified encoding occurs when adding a string to a message.

### Exotic Characters

A wire-format string can contain only characters that are valid in the encoding of the surrounding message. The translation procedure detects exotic characters, and throws an exception with `TibrvStatus.INVALID_ENCODING`.

## Inbound Translation

Inbound translation occurs before the program receives the data.

Automatic inbound translation is correct when two programs exchange messages within the same locale.

**Warning**

In contrast, the automatic translation might be incorrect when the sender and receiver use different character encodings.

In this situation, the receiver must *explicitly* retranslate to the local encoding.

## See Also

[TibrvMsg.getStringEncoding\(\)](#)

[TibrvMsg.setStringEncoding\(\)](#)



# Interrupting Event Dispatch Threads

The Java method `Thread.interrupt()` is *ineffective* when all of these conditions are simultaneously true:

- [Tibrv](#) is open using the native (JNI) implementation.
- The thread dispatches an event queue or queue group.
- The queue or queue group is empty of events.

To effectively interrupt such a thread, a program can destroy the queue or queue group. In this situation, the dispatch method throws an exception; the thread can use that exception as an occasion to exit.

# Network Overview—rxd Transports

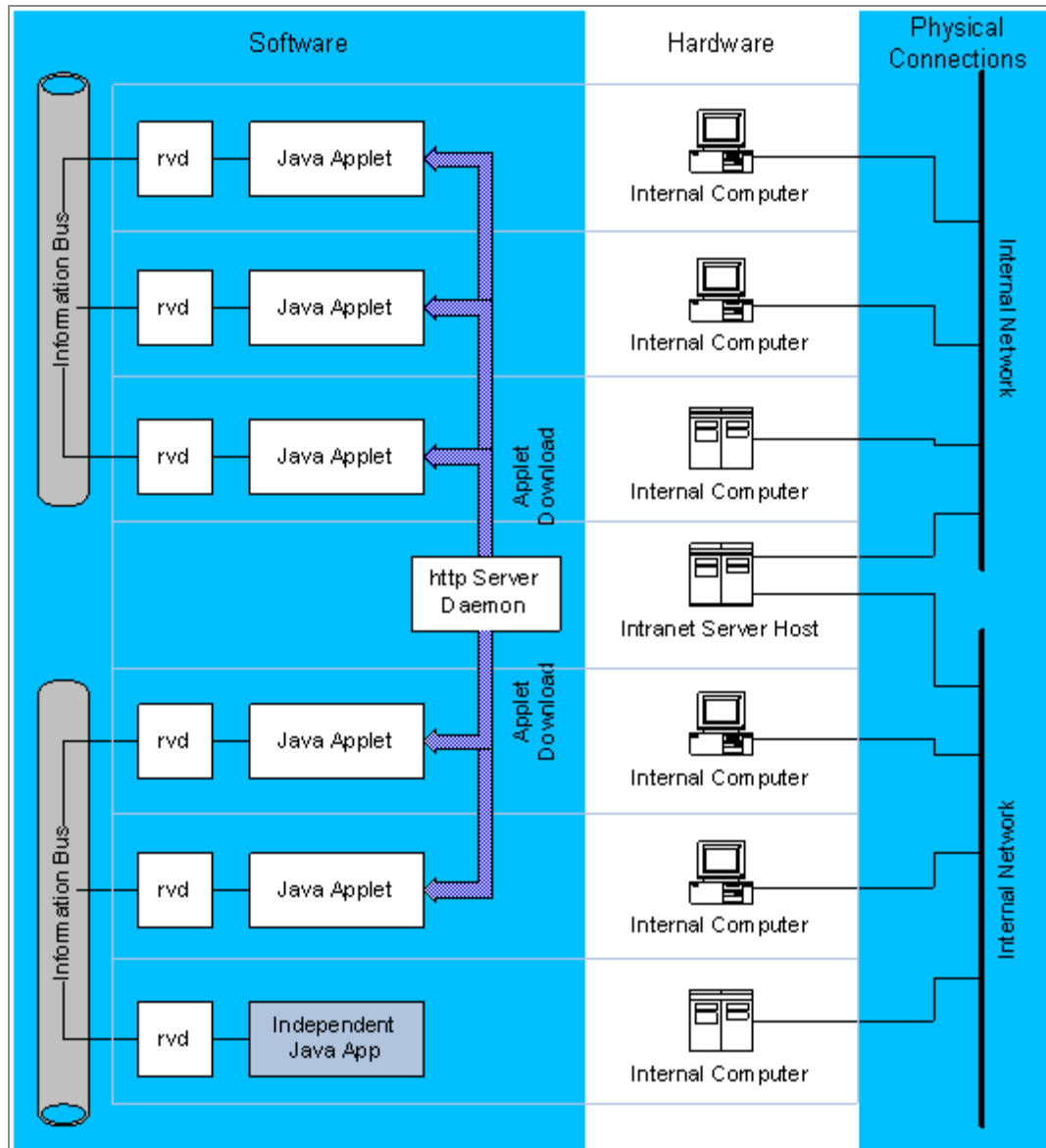
[Java Programs with rxd Transports](#) shows programs that communicate using rxd transports.

In the top and bottom of [Java Programs with rxd Transports](#), Java applets run in two separate local networks. Browsers download the applets from an intranet web server (center), but the applets then communicate with other Rendezvous programs on their *local* networks (that is, they do not communicate with web server host).

This configuration is well-suited to intranets within an enterprise. The applets may run on a single network, or on several networks connected by Rendezvous routing daemons.

The independent Java application (bottom) also uses an rxd transport, and communicates with applets on its network.

Figure 1: Java Programs with rvd Transports



# Intranet Web Site Considerations

When you design Java programs that run on an intranet—using rvd transports to connect to the local network—pay special attention to these issues.

## Rendezvous Files

Correct operation of programs requires the Rendezvous classes. The program searches for the archive file that contains the Rendezvous classes; the CLASSPATH environment variable guides the search.

## JNI Shared Libraries

Correct operation of [TibrvRvdTransport](#) requires prior installation of the JNI shared libraries.

## rvd

[TibrvRvdTransport](#) objects must be able to connect to an rvd process. We recommend that you install the rvd executable on the client host computer.

# Programmer's Checklist

---

Consult this section for details of the application development cycle.

# Checklist

Developers of Rendezvous programs can use this checklist during the four phases of the development cycle: installing Rendezvous software, coding your Java program, compiling your Java program, and running your program as either an independent application or as a browser applet.

## Install

- Install the Rendezvous software release, which automatically includes the Java archive files in the `lib` subdirectory.
- The `CLASSPATH` variable must include an archive file that contains the Rendezvous classes.

On UNIX platforms, include the appropriate archive file from [Archive Files](#).

On other supported platforms, this step is automatic.

## Code

- Import the correct Rendezvous package.

```
Import com.tibco.tibrv.*
```

## Compile

- An up-to-date version of Java (JDK) is required.
- The `CLASSPATH` variable must include an archive file that contains the Rendezvous classes. Include the appropriate archive file from [Archive Files](#).
- To use TIBCO RendezvousÆ Server In-Process Module (IPM), the `CLASSPATH` variable must include the archive file `tibrvnative.jar`; it is the only archive file that supports IPM.

# Run

- An up-to-date version of Java (JRE) is required.

## Run as Independent Application

- Rendezvous software must be properly installed, so that the application can access shared library files. For details, see [Shared Library Files](#).
- The CLASSPATH variable must include the location of the Rendezvous classes. Include the appropriate archive file from [Archive Files](#).
- The application must be able to connect to a Rendezvous daemon process (rvd).

## Run as Applet Using rvd Transports

- Rendezvous software must be properly installed on the computer on which the applet is running, so that the application can access shared library files.
- Applets running in a browser gain speed when the Rendezvous package is installed locally. To take advantage of this efficiency, the CLASSPATH variable (as read when the browser starts) must include an appropriate archive file from [Archive Files](#).
- The applet must be able to connect to a Rendezvous daemon process (rvd). If the browser does not permit the applet to connect to rvd, adjust the browser's security settings to permit TCP connections to the rvd client connection socket (the daemon parameter specifies this socket).

## Run as Independent Application with IPM

- Ensure that the IPM C library is in the path variable; see [IPM Library](#).

# Archive Files

[Archive Files](#) details the jar files for various implementations of Rendezvous for Java. Place the appropriate jar file in the CLASSPATH environment variable.

## Archive Files

Edition	Archive File	Contents
---------	--------------	----------

### JNI Preferred

We recommend the following two native (JNI) implementations for most situations.

These implementations were formerly known as *thin-message* technology. As of release 8.2, this name no longer aptly describes the actual implementation differences.

Native Local	tibrvnative.jar	Standard classes Certified message delivery Fault tolerance <a href="#">TibrvRvdTransport</a> Vector sending Vector listening Dispose IPM
Native Secure Daemon	tibrvnativesd.jar	Standard classes Certified message delivery Fault tolerance Secure daemons ( <a href="#">TibrvSdContext</a> ) <a href="#">TibrvRvdTransport</a> Vector sending Vector listening Dispose

### JNI Backward Compatibility

The following two native (JNI) implementations are retained only for backward compatibility, when it is impractical to migrate to the native preferred implementation.

These implementations were formerly known as *full-message* technology. As of release 8.2, this name no longer aptly describes the actual implementation differences.



Edition	Archive File	Contents
Backward Compatibility Local	tibrvj.jar	Standard classes Certified message delivery Fault tolerance <a href="#">TibrvRvdTransport</a>
Backward Compatibility Secure Daemon	tibrvsd.jar	Standard classes Certified message delivery Fault tolerance Secure daemons ( <a href="#">TibrvSdContext</a> ) <a href="#">TibrvRvdTransport</a>

# Shared Library Files

Independent Java applications that use rvd transports must be able to access Rendezvous shared library files (C libraries). [Environment Variables for Shared Library Files](#) details the environment variables that direct Java applications to the Rendezvous installation directory. The installation directory must contain the required shared library files.

## Environment Variables for Shared Library Files

Platform	Environment Variable
Windows	PATH must include <i>install_dir\bin</i>  The installation procedure sets this variable automatically.
UNIX (Except as below)	LD_LIBRARY_PATH must include <i>install_dir/lib</i>  For 64-bit JVM, LD_LIBRARY_PATH must also include <i>install_dir/lib/64</i>
HP/UX	SHLIB_PATH must include <i>install_dir/lib</i>

# IPM Library

A Java program can use standard Rendezvous communication library or the IPM library.

To select between the standard Rendezvous communication library or the IPM library, modify the path variable according to [Selecting the Communications Library](#).

## Selecting the Communications Library

Library	Instructions
Standard Rendezvous Communications Library	<p><b>UNIX</b> Ensure that the subdirectory <code>TIBCO_HOME/lib</code> appears <i>before</i> <code>TIBCO_HOME/lib/ipm</code> in your <code>LD_LIBRARY_PATH</code> (or <code>SH_LIBRARY_PATH</code>) environment variable.</p> <p><b>Windows</b> Ensure that the subdirectory <code>TIBCO_HOME/bin</code> appears</p>

Library	Instructions
IPM Communications Library	<i>before</i> TIBCO_HOME/bin/ipm in your PATH environment variable.
	<b>UNIX</b> Ensure that the subdirectory TIBCO_HOME/lib/ipm appears <i>before</i> TIBCO_HOME/lib in your LD_LIBRARY_PATH (or SH_LIBRARY_PATH) environment variable.  <b>Windows</b> Ensure that the subdirectory TIBCO_HOME/bin/ipm appears <i>before</i> TIBCO_HOME/bin in your PATH environment variable.

Existing Rendezvous applications that use the standard shared library do not require modifications in order to use the IPM library instead.

# Rendezvous Environment

---

This brief section describes the methods that open and close the internal machinery upon which Rendezvous software depends.

# Tibrv

*Class*

## Declaration

```
class com.tibco.tibrv.Tibrv.Tibrv  
    extends java.lang.Object
```

## Purpose

The Rendezvous environment.

## Remarks

Programs do not create instances of [Tibrv](#). Instead, programs use its static methods to open and close the Rendezvous environment. Private constants and variables contain references to Rendezvous resources.

For a list of constants that this class defines, see [Implementations](#).

Method	Description
<b>Environment Life Cycle and Properties</b>	
<a href="#">Tibrv.open()</a>	Start Rendezvous internal machinery.
<a href="#">Tibrv.close()</a>	Stop and destroy Rendezvous internal machinery.
<a href="#">Tibrv.getVersion()</a>	Return the version string or one of its components.
<a href="#">Tibrv.isValid()</a>	Determine whether the Rendezvous machinery is open.
<a href="#">Tibrv.isNativeImpl()</a>	Discriminate between native (JNI) implementation and Java implementation.

Method	Description
--------	-------------

## Utility Objects

<a href="#">Tibrv.defaultQueue()</a>	Extract the default queue object.
<a href="#">Tibrv.processTransport()</a>	Extract the intra-process transport object.

## Asynchronous Errors

<a href="#">Tibrv.getErrorCallback()</a>	Extract the callback object that processes asynchronous errors.
<a href="#">Tibrv.setErrorCallback()</a>	Set the callback for processing asynchronous errors.

## IPM

<a href="#">Tibrv.isIPM()</a>	Test whether the IPM library is loaded.
<a href="#">Tibrv.setRVParameters()</a>	Set TIBCO Rendezvous daemon command line parameters for IPM.

## Inherited Methods

[java.lang.Object.equals](#)  
[java.lang.Object.getClass](#)  
[java.lang.Object.hashCode](#)  
[java.lang.Object.notify](#)  
[java.lang.Object.notifyAll](#)  
[java.lang.Object.toString](#)  
[java.lang.Object.wait](#)

# Tibrv.close()

*Method*

## Declaration

```
static void close() throws TibrvException
```

## Purpose

Stop and destroy Rendezvous internal machinery.

## Remarks

After [Tibrv.close\(\)](#) destroys the internal machinery, Rendezvous software becomes inoperative:

- Events no longer arrive in queues.
- All events, queues and queue groups are unusable, so programs can no longer dispatch events.
- All transports are unusable, so programs can no longer send outbound messages.

After closing a [Tibrv](#) object, all events, transports, queues and queue groups associated with that environment are invalid; it is illegal to call any methods of these objects.

After closing a [Tibrv](#) environment, you can reopen it either with the same implementation, or with a different implementation.

## Reference Count

A reference count protects against interactions between programs and third-party packages that call [Tibrv.open\(\)](#) and [Tibrv.close\(\)](#). Each call to [Tibrv.open\(\)](#) increments an internal counter; each call to [Tibrv.close\(\)](#) decrements that counter. A call to [Tibrv.open\(\)](#) actually creates internal machinery only when the reference counter is zero; subsequent calls merely increment the counter, but do not duplicate the machinery. A call to [Tibrv.close\(\)](#) actually destroys the internal machinery only when the call decrements the counter to zero; other calls merely decrement the counter. In each program, the number of calls to [Tibrv.open\(\)](#) and [Tibrv.close\(\)](#) must match.

To ascertain the actual state of the internal machinery, use [Tibrv.isValid\(\)](#).

## See Also

[Tibrv.isValid\(\)](#)

[Tibrv.open\(\)](#)



# Tibrv.defaultQueue()

*Method*

## Declaration

```
static TibrvQueue defaultQueue()
```

## Purpose

Extract the default queue object.

## Remarks

If Rendezvous is not open, this method returns null.

Each process has exactly one default queue; the call [Tibrv.open\(\)](#) automatically creates it. Programs must not destroy the default queue.

## See Also

[TibrvQueue](#).

# Tibrv.getErrorCallback()

*Method*

## Declaration

```
static TibrvErrorCallback getErrorCallback()
```

## Purpose

Extract the callback object that processes asynchronous errors.

## Remarks

If the program has not set an error callback, this method returns null.

## See Also

[Tibrv.setErrorCallback\(\)](#)

[TibrvErrorCallback](#)

# Tibrv.getVersion()

*Method*

## Declaration

```
static String getVersion()  
static String getCmVersion()  
static String getFtVersion()  
static int   getMajorVersion()  
static int   getMinorVersion()  
static int   getUpdateVersion()  
static String getVersionString() // This method is deprecated //
```

## Purpose

Return the version string or one of its components.

## Remarks

getVersion() returns the version number of the Java package (whether using machinery from [Tibrv.IMPL\\_JAVA](#) or [Tibrv.IMPL\\_NATIVE](#)).

getCmVersion() returns the version number of the underlying C implementation of the certified delivery library ([Tibrv.IMPL\\_NATIVE](#) only).

getFtVersion() returns the version number of the underlying C implementation of the fault tolerance library ([Tibrv.IMPL\\_NATIVE](#) only).

getMajorVersion(), getMinorVersion() and getUpdateVersion() return the three segments of the complete version number that getVersion() returns (it formats these three segments into a string).

The method getVersionString() is deprecated (though not yet obsolete), starting with release 6.2. We encourage programmers to migrate to getVersion() instead.

# Tibrv.isIPM()

*Method*

## Declaration

```
public static boolean isIPM();
```

## Purpose

Test whether the IPM library is loaded.

## Remarks

You can use this call to determine whether an application program process has linked the IPM. You can test that your program dynamically links the correct library. You can program different behavior depending on which library is linked.

true indicates that the program links the IPM library (from the lib/ipm/ subdirectory).

false indicates that the program links the standard Rendezvous library (from the lib/ directory).

# Tibrv.isNativeImpl()

*Method*

## Declaration

```
static boolean isNativeImpl()
```

## Purpose

Discriminate between native (JNI) implementation and Java implementation.

## Remarks

When a program opens the Rendezvous environment with [Tibrv.IMPL\\_SELECT](#), it can later use this method to determine the effective implementation.

This method returns true when Rendezvous software uses the JNI library. Otherwise it returns false.

# Tibrv.isValid()

*Method*

## Declaration

```
static boolean isValid()
```

## Purpose

Determine whether the Rendezvous machinery is open.

## Remarks

This method returns true after the method [Tibrv.open\(\)](#) has returned successfully.

Otherwise this method returns false. The value false usually indicates one of these situations:

- The program has not opened the Rendezvous environment.
- Attempts to open the environment failed.
- The environment was open, but the program closed it with [Tibrv.close\(\)](#).

## See Also

[Tibrv.open\(\)](#)

[Tibrv.close\(\)](#)

# Tibrv.open()

*Method*

## Declaration

```
static void open() throws TibrvException  
static void open(int implementation) throws TibrvException  
static void open(String pathname) throws TibrvException
```

## Purpose

Start Rendezvous internal machinery.

## Remarks

This call creates the internal machinery that Rendezvous software requires for its operation:

- Internal data structures.
- Default event queue.
- Intra-process transport.
- Event driver.

Until the first call to [Tibrv.open\(\)](#) creates the internal machinery, all events, transports, queues and queue groups are unusable. Messages and their methods do not depend on the internal machinery.

Parameter	Description
implementation	Open Rendezvous using this implementation.  Choose a value from among the constants <a href="#">Tibrv.IMPL_NATIVE</a> , <a href="#">Tibrv.IMPL_JAVA</a> , and <a href="#">Tibrv.IMPL_SELECT</a> . See <a href="#">Implementations</a> .

Parameter	Description
pathname	<p>Programs that use IPM can supply a filepath name, which explicitly specifies a configuration file. IPM reads parameter values from that file.</p> <p>For details, see <a href="#">Configuring IPM in TIBCO Rendezvous Concepts</a>.</p> <p>When IPM is not available, this version of the method throws an exception with error status.</p>

## Method Forms

With no argument, open using [Tibrv.IMPL\\_SELECT](#). To determine the actual implementation, use [Tibrv.isNativeImpl\(\)](#).

With an implementation argument, open using the specified implementation.

## Implementations

These constants denote the implementation choices.

Constant	Description
Tibrv.IMPL_NATIVE	<p>Open TIBCO Rendezvous machinery using the implementation in the JNI library, if possible; otherwise, throw a <a href="#">TibrvException</a> with one of these status codes: <a href="#">TibrvStatus.VERSION_MISMATCH</a>, <a href="#">TibrvStatus.WRONG_JAVA_ARCHIVE</a>, <a href="#">TibrvStatus.LIBRARY_NOT_FOUND</a>, <a href="#">TibrvStatus.LIBRARY_NOT_LOADED</a>.</p> <p>An exception could indicate an incorrect value of the CLASSPATH environment variable.</p>
Tibrv.IMPL_JAVA	Open TIBCO Rendezvous machinery using the pure Java implementation.
Tibrv.IMPL_SELECT	Open TIBCO Rendezvous machinery using JNI library if possible; otherwise, use the Java implementation.



## Reference Count

A reference count protects against interactions between programs and third-party packages that call [Tibrv.open\(\)](#) and [Tibrv.close\(\)](#). Each call to [Tibrv.open\(\)](#) increments an internal counter; each call to [Tibrv.close\(\)](#) decrements that counter. A call to [Tibrv.open\(\)](#) actually creates internal machinery only when the reference counter is zero; subsequent calls merely increment the counter, but do not duplicate the machinery. A call to [Tibrv.close\(\)](#) actually destroys the internal machinery only when the call decrements the counter to zero; other calls merely decrement the counter. In each program, the number of calls to [Tibrv.open\(\)](#) and [Tibrv.close\(\)](#) must match.

## See Also

[Implementations](#)

[Tibrv.close\(\)](#)

[Tibrv.isNativeImpl\(\)](#)

# Tibrv.processTransport()

*Method*

## Declaration

```
static TibrvProcessTransport processTransport()
```

## Purpose

Extract the intra-process transport object.

## Remarks

If Rendezvous is not open, this method returns null.

Each process has exactly one intra-process transport; the call [Tibrv.open\(\)](#) automatically creates it. Programs must not destroy the intra-process transport.

## See Also

[TibrvProcessTransport](#)

# Tibrv.setErrorCallback()

*Method*

## Declaration

```
static void setErrorCallback(TibrvErrorCallback errorCallback)
```

## Purpose

Set the callback for processing asynchronous errors.

Parameter	Description
errorCallback	Use this <a href="#">TibrvErrorCallback</a> object to process all asynchronous errors.

## See Also

[Tibrv.getErrorCallback\(\)](#)

[TibrvErrorCallback](#)

[TibrvErrorCallback.onError\(\)](#)

# Tibrv.setRVParameters()

*Method*

## Declaration

```
static void setRVParameters(String[] parameters)  
    throws TibrvException
```

## Purpose

Set TIBCO Rendezvous daemon command line parameters for IPM.

## Remarks

The TIBCO Rendezvous daemon process (rvd) accepts several command line parameters. When IPM serves the role of the daemon, this call lets you supply those parameters from within the application program.

This call is optional. When this call is present, it has no effect unless it precedes the call to [Tibrv.open\(\)](#). For interaction semantics, see Parameter Configuration—Precedence and Interaction in TIBCO Rendezvous Concepts.

This call is available only with IPM. When IPM is not available, this call throws an exception with error status.

Parameter	Description
parameters	Supply an array of strings. Each string is either a command line parameter name (for example, -logfile) or its value.  For details about parameters, see rvd in TIBCO Rendezvous Administration

## IPM: Configuring Parameters In Program Code

```
String rvParams[] = {"-reliability", "3",  
    "-reuse-port", "30000"};
```

```
Tibrv.setRVParameters(rvParams);  
Tibrv.open();
```

## See Also

Configuring IPM in TIBCO Rendezvous Concepts

# TibrvSdContext

*Class*

## Declaration

```
final class com.tibco.tibrv.TibrvSdContext
static final String TIBRV_SECURE_DAEMON_ANY_NAME = null;
static final String TIBRV_SECURE_DAEMON_ANY_CERT = null;
```

## Purpose

This class defines static methods for interacting with secure Rendezvous daemons.

## Remarks

Programs do not create instances of [TibrvSdContext](#). Instead, programs use its static methods to configure user names, passwords and certificates, and to register trust in daemon certificates.

To use the methods of this class, Java programs must satisfy these two criteria:

- Programs must compile and run with the archive file `tibrvjsd.jar` (which defines this class); see [Archive Files](#).
- Programs must open the Rendezvous environment using the native implementation; see [Tibrv.open\(\)](#)—in particular, the constant [Tibrv.IMPL\\_NATIVE](#), which specifies the Rendezvous JNI library implementation.

Method	Description
<a href="#">TibrvSdContext.setDaemonCert()</a>	Register trust in a secure daemon.
<a href="#">TibrvSdContext.setUserCertWithKey()</a>	Register a (PEM) certificate with private key for identification to secure daemons.
<a href="#">TibrvSdContext.setUserCertWithKeyBin()</a>	Register a (PKCS #12) certificate

Method	Description
	with private key for identification to secure daemons.
<code>TibrvSdContext.setUserNameWithPassword()</code>	Register a user name with password for identification to secure daemons.

# TibrvSdContext.setDaemonCert()

*Method*

## Declaration

```
static void setDaemonCert(  
    java.lang.String  daemonName,  
    java.lang.String  daemonCert)  
throws TibrvException
```

## Purpose

Register trust in a secure daemon.

## Remarks

When any program transport connects to a secure daemon, it verifies the daemon's identity using TLS protocols. Certificates registered using this method identify trustworthy daemons. Programs divulge user names and passwords to daemons that present registered certificates.

Parameter	Description
daemonName	Register a certificate for a secure daemon with this name. For the syntax and semantics of this parameter, see <a href="#">Daemon Name</a> , below.
daemonCert	Register this public certificate. The text of this certificate must be in PEM encoding. See also <a href="#">Certificate</a> .

## Daemon Name

The daemon name is a three-part string of the form:

```
ssl:host:port_number
```



This string must be identical to the string you supply as the daemon argument to the transport creation call; see [TibrvRvdTransport\(\)](#).

Colon characters (:) separate the three parts.

*ssl* indicates the protocol to use when attempting to connect to the daemon.

*host* indicates the host computer of the secure daemon. You can specify this host either as a network IP address, or a hostname. Omitting this part specifies the local host.

*port\_number* specifies the port number where the secure daemon listens for TLS connections.

(This syntax is similar to the syntax connecting to remote daemons, with the addition of the prefix *ssl*.)

In place of this three-part string, you can also supply the constant [TibrvSdContext.TIBRV\\_SECURE\\_DAEMON\\_ANY\\_NAME](#). This form lets you register a catch-all certificate that applies to any secure daemon for which you have not explicitly registered another certificate. For example, you might use this form when several secure daemons share the same certificate.

## Certificate

For important details, see [CA-Signed Certificates](#) in TIBCO Rendezvous Administration.

In place of an actual certificate, you can also supply the constant [TibrvSdContext.TIBRV\\_SECURE\\_DAEMON\\_ANY\\_CERT](#). The program accepts any certificate from the named secure daemon. For example, you might use this form when testing a secure daemon configuration, before generating any actual certificates.

## Any Name and Any Certificate

Notice that the constants [TibrvSdContext.TIBRV\\_SECURE\\_DAEMON\\_ANY\\_NAME](#) and [TibrvSdContext.TIBRV\\_SECURE\\_DAEMON\\_ANY\\_CERT](#) each eliminate one of the two security checks before transmitting sensitive identification data to a secure daemon. We strongly discourage using both of these constants simultaneously, because that would eliminate all security checks, leaving the program vulnerable to unauthorized daemons.

# TibrvSdContext.setUserCertWithKey()

*Method*

## Declaration

```
static void setUserCertWithKey(  
    java.lang.String userCertWithKey,  
    java.lang.String password)  
throws TibrvException
```

## Purpose

Register a (PEM) certificate with private key for identification to secure daemons.

## Remarks

When any program transport connects to a secure daemon, the daemon verifies the program's identity using TLS protocols.

The Rendezvous API includes two methods that achieve similar effects:

- This call accepts a certificate in PEM text format.
- [TibrvSdContext.setUserCertWithKeyBin\(\)](#) accepts a certificate in PKCS #12 binary format.

Parameter	Description
userCertWithKey	Register this user certificate with private key. The text of this certificate must be in PEM encoding.
password	Use this password to decrypt the private key.



### Important

For important information about password security, see Security Factors in TIBCO Rendezvous Administration.

## CA-Signed Certificate

You can also supply a certificate signed by a certificate authority (CA). To use a CA-signed certificate, you must supply not only the certificate and private key, but also the CA's public certificate (or a chain of such certificates). Concatenate these items in one string. For important details, see CA-Signed Certificates in TIBCO Rendezvous Administration.

## Exceptions

An exception that reports status [TibrvStatus.INVALID\\_FILE](#) can indicate either disk I/O failure, or invalid certificate data, or an incorrect password.

## See Also

[TibrvSdContext.setUserCertWithKeyBin\(\)](#)

# TibrvSdContext.setUserCertWithKeyBin()

*Method*

## Declaration

```
void setUserCertWithKeyBin(  
    byte[]      userCertWithKey,  
    java.lang.String password);
```

## Purpose

Register a (PKCS #12) certificate with private key for identification to secure daemons.

## Remarks

When any program transport connects to a secure daemon, the daemon verifies the program's identity using TLS protocols.

The Rendezvous API includes two methods that achieve similar effects:

- This call accepts a certificate in PKCS #12 binary format.
- [TibrvSdContext.setUserCertWithKey\(\)](#) accepts a certificate in PEM text format.

Parameter	Description
userCertWithKey	Register this user certificate with private key. The binary data of this certificate must be in PKCS #12 format.
password	Use this password to decrypt the private key.



### Important

For important information about password security, see [Security Factors](#) in TIBCO Rendezvous Administration.

## CA-Signed Certificate

You can also supply a certificate signed by a certificate authority (CA). To use a CA-signed certificate, you must supply not only the certificate and private key, but also the CA's public certificate (or a chain of such certificates). For important details, see [CA-Signed Certificates](#) in TIBCO Rendezvous Administration.

## Exceptions

An exception that reports status [TibrvStatus.INVALID\\_FILE](#) can indicate either disk I/O failure, or invalid certificate data, or an incorrect password.

## See Also

[TibrvSdContext.setUserCertWithKey\(\)](#)

[www.rsasecurity.com/rsalabs/pkcs](http://www.rsasecurity.com/rsalabs/pkcs)

# TibrvSdContext.setUserNameWithPassword()

*Method*

## Declaration

```
static TibrvStatus setUserNameWithPassword(  
    java.lang.String  userName,  
    java.lang.String  password)  
throws TibrvException
```

## Purpose

Register a user name with password for identification to secure daemons.

## Remarks

When any program transport connects to a secure daemon, the daemon verifies the program's identity using TLS protocols.

Parameter	Description
userName	Register this user name for communicating with secure daemons.
password	Register this password for communicating with secure daemons.



### Important

For important information about password security, see [Security Factors](#) in TIBCO Rendezvous Administration.

# Data

---

This section describes messages and the data they contain.

## See Also

[Strings and Character Encodings](#)

[Custom Datatypes](#)

# Field Names and Field Identifiers

In Rendezvous 5 and earlier releases, programs would specify fields within a message using a field name. In Rendezvous 6 and later releases, programs can specify fields in two ways:

- A *field name* is a character string. Each field can have at most one name. Several fields can have the same name.
- A *field identifier* is a 16-bit unsigned integer, which must be unique within the message. That is, two fields in the same message cannot have the same identifier. However, a nested submessage is considered a separate identifier space from its enclosing parent message and any sibling submessages.

Java presents these identifiers as 32-bit integers, padding the high bytes with zero.

Message methods specify fields using a combination of a field name and a unique field identifier. When absent, the default field identifier is zero.

To compare the speed and space characteristics of these two options, see [Search Characteristics](#).

## Rules and Restrictions

NULL is a legal field name *only* when the identifier is zero. It is *illegal* for a field to have *both* a non-zero identifier *and* a NULL field name.

Note that in Java, NULL is *not* the same as "" (the empty string). It is legal for a field to have a non-zero identifier and the empty string as its field name. However, we generally recommend *against* using the empty string as a field name.

## Adding a New Field

When a program adds a new field to a message, it can attach a field name, a field identifier, or both. If the program supplies an identifier, Rendezvous software checks that it is unique within the message; if the identifier is already in use, the operation fails with the status code `TibrvStatus.ID_IN_USE`.

## Search Characteristics

In general, an identifier search completes in constant time. In contrast, a name search completes in linear time proportional to the number of fields in the message. Name search



is quite fast for messages with 16 fields or fewer; for messages with more than 16 fields, identifier search is faster.

## Space Characteristics

The smallest field name is a one-character string, which occupies three bytes in Rendezvous wire format. That one ASCII character yields a name space of 127 possible field names; a larger range requires additional characters.

Field identifiers are 16 bits, which also occupy three bytes in Rendezvous wire format. However, those 16 bits yield a space of 65535 possible field identifiers; that range is fixed, and cannot be extended.

## Finding a Field Instance

When a message contains several field instances with the same field name, these methods find a specific instance by name and number (they do not use field identifiers):

- [TibrvMsg.removeFieldInstance\(\)](#).
- [TibrvMsg.getFieldInstance\(\)](#).

# TibrvMsg

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvMsg  
    extends java.lang.Object
```

## Purpose

Represent Rendezvous messages.

## Remarks

This class has no `destroy()` method. Instead, the Java garbage collector reclaims storage automatically. Nonetheless it is possible to explicitly manage native message storage; see [TibrvMsg.dispose\(\)](#).

Method	Description
<b>Message Life Cycle and Properties</b>	
<a href="#">TibrvMsg()</a>	Create a message object.
<a href="#">TibrvMsg.dispose()</a>	Release native storage associated with the message.
<a href="#">TibrvMsg.getNumFields()</a>	Extract the number of fields in a message.
<b>Fields</b>	
<a href="#">TibrvMsg.add()</a>	Add a field to a message.
<a href="#">Add Scalar</a> (convenience methods)	Add a field containing a scalar value.

Method	Description
<a href="#">TibrvMsg.addField()</a>	Add a field object to a message.
<a href="#">TibrvMsg.get()</a>	Get the value of a specified field from a message.
<a href="#">Get Scalar</a> (convenience methods)	Get the value of a field as a scalar value.
<a href="#">TibrvMsg.getAsBytes()</a>	Extract the data from a message as a byte sequence.
<a href="#">TibrvMsg.getField()</a>	Get a specified field from a message.
<a href="#">TibrvMsg.getFieldByIndex()</a>	Get a field from a message by an index.
<a href="#">TibrvMsg.getFieldInstance()</a>	Get a specific instance of a field from a message.
<a href="#">TibrvMsg.removeField()</a>	Remove a field from a message.
<a href="#">TibrvMsg.removeFieldInstance()</a>	Remove a specified instance of a field from a message.
<a href="#">TibrvMsg.update()</a>	Update a field within a message.
<a href="#">Update Scalar</a> (convenience methods)	Update a field containing a scalar value.
<a href="#">TibrvMsg.updateField()</a>	Update a field within a message.
<b>Address Information</b>	
<a href="#">TibrvMsg.getReplySubject()</a>	Extract the reply subject from a message.
<a href="#">TibrvMsg.getSendSubject()</a>	Extract the subject from a message.
<a href="#">TibrvMsg.setReplySubject()</a>	Set the reply subject for a message.

Method	Description
<a href="#">TibrvMsg.setSendSubject()</a>	Set the subject for a message.
<b>Event Dispatched</b>	
<a href="#">TibrvMsg.getEvent()</a>	Extract the event associated with a (dispatched) message object.
<b>String and Character Conversion</b>	
<a href="#">TibrvMsg.getTypeName()</a>	Convert a type designator to its corresponding string name.
<a href="#">TibrvMsg.toString()</a>	Format a message as a string.
<a href="#">TibrvMsg.getStringEncoding()</a>	Return the character encoding for converting between Java Strings and wire format strings.
<a href="#">TibrvMsg.setStringEncoding()</a>	Set the character encoding for converting between Java Unicode strings and wire format strings.
<b>Custom Datatypes</b>	
See <a href="#">Custom Datatypes</a> .	
<a href="#">TibrvMsg.getDecoder()</a>	Extract the decoder interface for a custom datatype.
<a href="#">TibrvMsg.getEncoder()</a>	Extract the encoder interface for a custom datatype.
<a href="#">TibrvMsg.setHandlers()</a>	Define a custom datatype by registering its encoder and decoder interfaces.

## Inherited Methods

```
java.lang.Object.equals
java.lang.Object.getClass
java.lang.Object.hashCode
java.lang.Object.notify
java.lang.Object.notifyAll
java.lang.Object.wait
```

## Datatype Constants

These constants are all defined with short values. To extract a human-readable name from a short value, see [TibrvMsg.getTypeName\(\)](#)

### Datatype Constants

Constant	Comment
TibrvMsg.DEFAULT	Used only as a type argument to <a href="#">TibrvMsg.add()</a>
TibrvMsg.MSG	
TibrvMsg.DATETIME	
TibrvMsg.OPAQUE	
TibrvMsg.STRING	
TibrvMsg.XML	Byte-array, compressed for network transmission.
TibrvMsg.BOOL	
TibrvMsg.I8	
TibrvMsg.U8	

Constant	Comment
TibrvMsg.I16	
TibrvMsg.U16	
TibrvMsg.I32	
TibrvMsg.U32	
TibrvMsg.I64	
TibrvMsg.U64	
TibrvMsg.F32	
TibrvMsg.F64	
TibrvMsg.IPPORT16	
TibrvMsg.IPADDR32	
TibrvMsg.U8ARRAY	
TibrvMsg.I16ARRAY	
TibrvMsg.U16ARRAY	
TibrvMsg.I32ARRAY	
TibrvMsg.U32ARRAY	
TibrvMsg.I64ARRAY	
TibrvMsg.U64ARRAY	
TibrvMsg.F32ARRAY	
TibrvMsg.F64ARRAY	

Constant	Comment
TibrvMsg.USER_FIRST	Custom datatypes begin with this number.
TibrvMsg.USER_LAST	

## See Also

[Strings and Character Encodings](#)

[TibrvMsgField](#)

[Custom Datatypes](#)

# TibrvMsg()

## Constructor

## Declaration

```
TibrvMsg()  
TibrvMsg(TibrvMsg msg) throws TibrvException  
TibrvMsg(byte[] bytes) throws TibrvException
```

## Purpose

Create a message object.

## Remarks

None of these constructors place address information on the new message object.

This class has no `destroy()` method. Instead, the Java garbage collector reclaims storage automatically. To actively manage message storage, see [TibrvMsg.dispose\(\)](#).

To use an inbound message outside of its callback, you can use the copy constructor to detach (that is, copy) a message—for example, to process a message in a different thread. (If your program detaches a message, then you must also ensure that no references to the copy remain; such references could interfere with garbage collection. Furthermore, when using the TIBCO Rendezvous JNI preferred library, we recommend that programs call [TibrvMsg.dispose\(\)](#) to explicitly release the copy's storage within the native C environment.)

Parameter	Description
msg	Create an independent copy of this message.
bytes	Create a message with fields populated from this byte array.  For example, programs can create such byte arrays from messages using the method <a href="#">TibrvMsg.getAsBytes()</a> , and store them in files; after reading them from such files, programs can reconstruct a message from its byte array.



## Method Forms

With no argument, create an empty message (with no fields).

With a [TibrvMsg](#) argument, create an independent copy of that message, copying all its fields (field values are also independent copies).

With a byte array argument, create a message with fields populated from the byte array.

## See Also

[TibrvMsg.dispose\(\)](#)

[TibrvMsg.getAsBytes\(\)](#)

[TibrvMsgCallback.onMsg\(\)](#)

[TibrvVectorCallback.onMsgs\(\)](#)

# TibrvMsg.add()

*Method*

## Declaration

```
void add(  
    java.lang.String fieldName,  
    java.lang.Object data)  
    throws TibrvException  
void add(  
    java.lang.String fieldName,  
    java.lang.Object data,  
    short type)  
    throws TibrvException  
void add(  
    java.lang.String fieldName,  
    java.lang.Object data,  
    short type,  
    int fieldId)  
    throws TibrvException
```

## Purpose

Add a field to a message.

## Remarks

This method copies the information into the new message field. All related convenience methods behave similarly.

Parameter	Description
fieldName	Add a field with this name.  null is a legal name. However, if fieldId is non-zero, then fieldName must be non-null.
data	Add a field with this data value.

Parameter	Description
	null is illegal.
type	<p>Add a field with this explicit type. For a list of types, see <a href="#">Datatype Constants</a>.</p> <p>When absent or <a href="#">TibrvMsg.DEFAULT</a>, determine the field's type from the type of the data. In <a href="#">Java to Wire Format Datatype Conversion Matrix</a>, filled dots indicate default encodings between homologous types.</p>
fieldId	<p>Add a field with this identifier. All field identifiers must be unique within each message.</p> <p>When absent, add a field without an identifier.</p> <p>Integers in the range [0, 65535] are valid identifiers.</p> <p>Zero is a special value, indicating no identifier. It is illegal to add a field that has both a null field name <i>and</i> a non-zero field identifier.</p>

## Encoding and Type Conversion

This method automatically tags Java data with a corresponding Rendezvous wire format type; sending the message actually triggers the encoding into wire format. [Java to Wire Format Datatype Conversion Matrix](#) specifies default encodings with filled circles; for some types you can override the default decoding by using convenience calls that force specific types (see [Add Scalar](#)).

## Encoding XML

We recommend converting the XML document string to a byte sequence, using the encoding that corresponds to your locale (file encoding system property). Then create a [TibrvXml](#) object containing the bytes, and add that object to the message. This practice ensures that all receivers can easily parse the resulting XML document. For more information, see [Decoding XML](#).

## Nested Message

When the data argument is a message object, this method adds only the data portion of the nested message; it does not include any address information or certified delivery information.

## Empty Array

The behavior of `TibrvMsg.add` differs among the various implementations when the [data](#) argument is an empty array. The reason for the discrepancy is that Java supports empty arrays, while C does not.

The native (JNI) preferred implementation uses underlying C calls to manipulate messages stored outside of the Java environment, so it cannot support empty arrays; consequently, this method throws an exception when it receives an empty array.

The native (JNI) backward compatibility implementation and the pure Java implementation both manipulate messages stored within the Java environment, so they do support empty arrays; consequently, this method correctly adds an empty array to a message field.



# Add Scalar

## Convenience Methods

### Declaration

```
void add(  
    java.lang.String fieldName,  
    scalar_type data)  
    throws TibrvException  
void add(  
    java.lang.String fieldName,  
    scalar_type data,  
    int fieldId)  
    throws TibrvException  
void addUbits(  
    java.lang.String fieldName,  
    scalar_type data)  
    throws TibrvException  
void addUbits(  
    java.lang.String fieldName,  
    scalar_type data,  
    int fieldId)  
    throws TibrvException
```

### Purpose

Add a field containing a scalar value.

### Method Forms

The convenience methods named **add()** determine the field type from the numeric type of the data.

The convenience methods named **addUnn()** add *unsigned* integer fields, *discarding* the sign bit of a Java integer data value.

Method Name	Java Data Type	Field Type	Type Description
add	boolean	TibrvMsg.BOOL	boolean
add	float	TibrvMsg.F32	32-bit floating point
add	double	TibrvMsg.F64	64-bit floating point
add	byte	TibrvMsg.I8	8-bit integer
add	short	TibrvMsg.I16	16-bit integer
add	int	TibrvMsg.I32	32-bit integer
add	long	TibrvMsg.I64	64-bit integer
addU8	byte	TibrvMsg.U8	8-bit unsigned integer
addU16	short	TibrvMsg.U16	16-bit unsigned integer
addU32	int	TibrvMsg.U32	32-bit unsigned integer
addU64	long	TibrvMsg.U64	64-bit unsigned integer

Parameter	Description
fieldName	<p>Add a field with this name.</p> <p>null is a legal name. However, if fieldId is non-zero, then fieldName must be non-null.</p>
data	<p>Add a field with this data value.</p> <p>null is illegal.</p>
fieldId	<p>Add a field with this identifier. All field identifiers must be unique within each message.</p> <p>When absent, add a field without an identifier.</p>

Parameter	Description
	Zero is a special value, indicating no identifier. It is illegal to add a field that has both a null field name, and a non-zero field identifier.



# TibrvMsg.addField()

*Method*

## Declaration

```
void addField(TibrvMsgField field)  
throws TibrvException
```

## Purpose

Add a field object to a message.

## Remarks

This method copies the information into the new message field. All related methods behave similarly.

It is illegal to add a field that has *both* a null field name, and a non-zero field identifier.

## Field Name Length

The the longest possible field name is 127 bytes.

Parameter	Description
field	Add this field to the message.

## See Also

[TibrvMsg.add\(\)](#)

[Add Scalar](#)

# TibrvMsg.dispose()

*Method*

## Declaration

```
void dispose()
```

## Purpose

Release native storage associated with the message.

## Remarks

In the native (JNI) preferred implementations, messages occupy storage outside of the Java environment (that is, in the native C environment) and also within the Java environment. When the Java garbage collector recycles the Java message object, this action triggers release of the corresponding native storage as well.

However, the timing of garbage collection is unpredictable, delaying the release of native storage as well. In applications where efficient management of native storage is a critical performance factor, you can use this method to explicitly free the native storage.

Call this dispose method at the end of a message callback method to immediately free the native storage associated with the message. The Java message object is independent of the native storage (and independent of this method), and it remains intact until the Java garbage collector recycles it in the usual way.

Attempting to access the message after calling this method results in an exception. (However, [TibrvMsg.getSendSubject\(\)](#) and [TibrvMsg.setReplySubject\(\)](#) retrieve null rather than throwing an exception

## Restrictions

dispose() is available only in the JNI preferred implementation. It is not available in the JNI backward compatibility implementation, nor in the pure Java implementation.

# TibrvMsg.get()

*Method*

## Declaration

```
java.lang.Object get(  
    java.lang.String fieldName)  
    throws TibrvException  
java.lang.Object get(  
    int fieldId)  
    throws TibrvException  
java.lang.Object get(  
    java.lang.String fieldName,  
    int fieldId)  
    throws TibrvException
```

## Purpose

Get the value of a specified field from a message.

## Remarks

Programs specify the field to retrieve using the `fieldName` and `fieldId` parameters.

The method returns a snapshot of the field value.

When a program gets fields with the datatypes listed here, this method (and convenience methods) return a reference to the actual value in the message—*not* a copy. Use caution when modifying values of these types: [TibrvMsg.MSG](#), [TibrvMsg.DATETIME](#), [TibrvMsg.IPPORT16](#), [TibrvMsg.IPADDR32](#), [TibrvMsg.OPAQUE](#) (extracted as a byte array), [TibrvMsg.XML](#) (extracted as a byte array), and *all* array types.

Programs can use a related method to loop through all the fields of a message; to retrieve each field by its integer index number, see [TibrvMsg.getFieldByIndex\(\)](#).

Parameter	Description
fieldName	Get a field with this name.
fieldId	Get the field with this identifier.

## Field Search Algorithm

This method, and related methods that *get* message fields, all use this algorithm to find a field within a message, as specified by a field identifier and a field name.

### Procedure

1. If the program supplied zero as the identifier, or omitted any identifier, then begin at step 3.  
If the program supplied a *non-zero* field identifier, then search for the field with that identifier.  
If the search succeeds, return the field.  
On failure, continue to step 2.
2. If the identifier search (in step 1) fails, and the program supplied a non-null field name, then search for a field with that name.  
If the name search succeeds, and the identifier in the field is null, return the field.  
If the name search succeeds, but the actual identifier in the field is non-null (so it does not match the identifier supplied) then throw an exception with the status code [TibrvStatus.ID\\_CONFLICT](#).  
On failure, or if the program supplied null as the field name, return null.
3. When the program supplied zero as the identifier, or omitted any identifier, then begin here.  
Search for a field with the specified name—even if that name is null.  
If the search succeeds, return the field.  
On failure, return null.

If a message contains several fields with the same name, searching by name finds the first instance of the field with that name.

## Extracting Fields from a Nested Message

Earlier releases of Rendezvous software allowed programs to get fields from a nested submessage by concatenating field names. Starting with release 6, Rendezvous software no longer supports this special case convenience. Instead, programs must separately extract the nested submessage using [TibrvMsg.get\(\)](#) (or a related method), and then get the desired fields from the submessage.

## Method Forms

With only a field name, find the field by name. If the field name is not present in the message, return null. If several fields with that name are present in the message, this method returns the first one that it finds.

With only a field identifier, find the field with that identifier (since identifiers are unique, the message can contain at most one such field). If the identifier is not present in the message, return null.

With both a field name and a field identifier, search first by identifier, and then by field name. If neither are present in the message, return null. If identifier search succeeds, return the field value. If the name search succeeds, but the actual identifier in the field is non-zero (so it does not match the identifier supplied) then throw a [TibrvException](#) with status code [TibrvStatus.ID\\_CONFLICT](#).

## Decoding and Type Conversion

This method automatically decodes the extracted field data from its Rendezvous wire format type to a corresponding Java type. In [Wire Format to Java Datatype Conversion Matrix](#), filled circles (as well as + and - symbols) specify default decodings between homologous types; for some types you can override the default decoding by using convenience calls that force specific types (see [Get Scalar](#)).

Java does not admit unsigned integers. When extracting an unsigned integer from an inbound message field, this method automatically promotes the value to the corresponding Java type that is large enough to contain it ([Wire Format to Java Datatype Conversion Matrix](#) indicates this with +). When extracting an unsigned 64-bit integer, Java does not have a type that can contain a number that uses all 64 bits; this method decodes it to a Java long, interpreting the high bit as a sign bit ([Wire Format to Java Datatype Conversion Matrix](#) indicates this with -).

## Decoding XML

After extracting the XML document into a byte array, explicitly convert it to a string using the encoding that corresponds to your locale (file encoding system property). For more information, see [Encoding XML](#).

Figure 3: Wire Format to Java Datatype Conversion Matrix

		Java Destination Type															
		Bool	Float	Double	Byte	Short	Integer	Long	float[]	double[]	byte[]	short[]	int[]	long[]	TibrvIPAddr	TibrvIPPort	TibrvDate
TibrvMsg Source Type	TibrvMsg.BOOL	●	N	N	N	N	N	N									
	TibrvMsg.F32	N	●	S	N	N	N	N									
	TibrvMsg.F64	N	N	●	N	N	N	N									
	TibrvMsg.I8	N	N	N	●	S	S	S									
	TibrvMsg.I16	N	N	N	N	●	S	S									
	TibrvMsg.I32	N	N	N	N	N	●	S									
	TibrvMsg.I64	N	N	N	N	N	N	●									
	TibrvMsg.U8	N	N	N	N	+	S	S									
	TibrvMsg.U16	N	N	N	N	N	+	S									
	TibrvMsg.U32	N	N	N	N	N	N	+									
	TibrvMsg.U64	N	N	N	N	N	N	-									
	TibrvMsg.IPADDR32													●			
	TibrvMsg.IPPORT16														●		
	TibrvMsg.DATETIME															●	
	TibrvMsg.F32ARRAY								●								
	TibrvMsg.F64ARRAY									●							
	TibrvMsg.I8ARRAY										●						
	TibrvMsg.I16ARRAY											●					
	TibrvMsg.I32ARRAY												●				
	TibrvMsg.I64ARRAY													●			
	TibrvMsg.U8ARRAY																
	TibrvMsg.U16ARRAY																
	TibrvMsg.U32ARRAY																
	TibrvMsg.U64ARRAY																
	TibrvMsg.MSG																●
	TibrvMsg.OPAQUE										●						
	TibrvMsg.STRING																●
	TibrvMsg.XML																●
	TibrvMsg.MSGARRAY																●
	TibrvMsg.STRINGARRAY																●

Key	
●	Homologous types; conversion always supported; no loss of information
S	Supported conversion; always supported
N	Numeric conversion; loss of information is possible (without warning)
+	Unsigned converted to signed integer with double bit precision; no loss of information
-	Unsigned 64-bit integer interpreted as signed 64 bits; high bit interpreted as sign bit
	Unsupported conversion

# Get Scalar

## Convenience Methods

### Declaration

```
scalar_type getScalar_type(  
    java.lang.String fieldName,  
    int fieldId)  
    throws TibrvException  
scalar_type getAsScalar_type(  
    java.lang.String fieldName,  
    int fieldId)  
    throws TibrvException
```

### Purpose

Get the value of a field as a scalar value.

### Remarks

Each convenience method in this family retrieves a field and extracts its data. If the field's type (as it exists) does not match the type of the convenience method, then the **getType** method throws an exception; in contrast, the **getAsType** method attempts to convert the data (see [Decoding and Type Conversion](#), and [Wire Format to Java Datatype Conversion Matrix](#)). If conversion is not possible, the method throws an exception with status code [TibrvStatus.CONVERSION\\_FAILED](#).

Parameter	Description
fieldName	Get a field with this name.
fieldId	Get the field with this identifier.

Method Name	Field Type	Java Type	Type Description
<code>getBoolean</code> <code>getAsBoolean</code>	<code>TibrvMsg.BOOL</code>	<code>boolean</code>	<code>boolean</code>
<code>getFloat</code> <code>getAsFloat</code>	<code>TibrvMsg.F32</code>	<code>float</code>	32-bit floating point
<code>getDouble</code> <code>getAsDouble</code>	<code>TibrvMsg.F64</code>	<code>double</code>	64-bit floating point
<code>getByte</code> <code>getAsByte</code>	<code>TibrvMsg.I8</code>	<code>byte</code>	8-bit integer
<code>getShort</code> <code>getAsShort</code>	<code>TibrvMsg.I16</code>	<code>short</code>	16-bit integer
<code>getInt</code> <code>getAsInt</code>	<code>TibrvMsg.I32</code>	<code>int</code>	32-bit integer
<code>getLong</code> <code>getAsLong</code>	<code>TibrvMsg.I64</code>	<code>long</code>	64-bit integer



# TibrvMsg.getAsBytes()

*Method*

## Declaration

```
byte[] getAsBytes() throws TibrvException
```

## Purpose

Extract the data from a message as a byte sequence.

## Remarks

Return a copy of the message data as a byte sequence, suitable for archiving in a file. To reconstruct the message from bytes, see [TibrvMsg\(\)](#).

The byte data includes the message header and all message fields in Rendezvous wire format. It does not include address information, such as the subject and reply subject, nor certified delivery information.

The byte sequence can contain interior null bytes.

## See Also

[TibrvMsg\(\)](#)

# TibrvMsg.getByteSize()

*Method*

## Declaration

```
int getByteSize()  
    throws TibrvException
```

## Purpose

Return the size of a message (in bytes).

## Remarks

This measurement accounts for the actual space that the message occupies (in wire format), including its header and its fields. It does not include address information, such as the subject or reply subject.

Programs can use this call as part of these tasks:

- Assess throughput rates.
- Limit output rates (also called *throttling*).

# TibrvMsg.getEvent()

*Method*

## Declaration

```
TibrvEvent getEvent();
```

## Purpose

Extract the event associated with a (dispatched) message object.

## Remarks

Dispatch associates the message with a listener event.

This call is valid only for an inbound message that has already been dispatched to a listener event. If the message is not associated with a listener event, then this method returns null.

## Restrictions

This method is available only in the JNI preferred implementation. It is not available in the JNI backward compatibility implementation, nor in the pure Java implementation.

## See Also

[TibrvEvent.getClosure\(\)](#)

[TibrvVectorCallback.onMsgs\(\)](#)

# TibrvMsg.getField()

*Method*

## Declaration

```
TibrvMsgField getField(  
    java.lang.String fieldName)  
throws TibrvException  
TibrvMsgField getField(  
    int fieldId)  
throws TibrvException  
TibrvMsgField getField(  
    java.lang.String fieldName,  
    int fieldId)  
throws TibrvException
```

## Purpose

Get a specified field from a message.

## Remarks

Programs specify the field to retrieve using the `fieldName` and `fieldId` parameters.

The method takes a snapshot of the field, and returns that information as a field object.

Programs can use a related method to loop through all the fields of a message; to retrieve each field by its integer index number, see [TibrvMsg.getFieldByIndex\(\)](#).

Parameter	Description
<code>fieldName</code>	Get a field with this name.
<code>fieldId</code>	Get the field with this identifier.

## Method Forms

With only a field name, find the field by name. If the field name is not present in the message, return null. If several fields with that name are present in the message, this method returns the first one that it finds.

With only a field identifier, find the field with that identifier (since identifiers are unique, the message can contain at most one such field). If the identifier is not present in the message, return null.

With both a field name and a field identifier, search first by identifier, and then by field name. If neither are present in the message, return null.

## See Also

[TibrvMsg.get\(\)](#)

# TibrvMsg.getFieldByIndex()

*Method*

## Declaration

```
TibrvMsgField getFieldByIndex(int fieldIndex)  
throws ArrayIndexOutOfBoundsException TibrvException
```

## Purpose

Get a field from a message by an index.

## Remarks

Programs can loop through all the fields of a message, to retrieve each field in turn using an integer index.

The method takes a snapshot of the field, and returns that information as a field object.

*Add*, *remove* and *update* calls can perturb the order of fields (which, in turn, affects the results when a program gets a field by index).

Parameter	Description
fieldIndex	Get the field with this index. Zero specifies the first field.

# TibrvMsg.getFieldInstance()

*Method*

## Declaration

```
TibrvMsgField getFieldInstance(  
    java.lang.String fieldName,  
    int instance)  
throws TibrvException
```

## Purpose

Get a specific instance of a field from a message.

## Remarks

When a message contains several field instances with the same field name, retrieve a specific instance by number (for example, get the *ith* field named foo). Programs can use this method in a loop that examines every field with a specified name.

The argument 1 denotes the first instance of the named field.

The method takes a snapshot of the field, and returns that information as a field object.

The method copies scalar data into the field object. Non-scalar data extracted from the field remain valid until the message is destroyed; that is, even removing the field or updating the field's value does *not* invalidate non-scalar data (such as arrays, strings, submessages, XML data, or opaque byte sequences).

When the instance argument is greater than the actual number of instances of the field in the message, this method returns null.

## Release 5 Interaction

Rendezvous 5 (and earlier) did not support array datatypes. Some older programs circumvented this limitation by using several fields with the same name to simulate arrays. This work-around is no longer necessary, since release 6 (and later) supports array datatypes within message fields. The method [TibrvMsg.getFieldInstance\(\)](#) ensures backward compatibility, so new programs can still receive and manipulate messages sent

from older programs. Nonetheless, we encourage programmers to use array types as appropriate, and we discourage storing several fields with the same name in a message.

Parameter	Description
fieldName	Get an instance of the field with this name. null specifies the empty string as the field name.
instance	Get this instance of the specified field name. The argument 1 denotes the first instance of the named field.

## See Also

[TibrvMsgField](#)



# TibrvMsg.getNumFields()

*Method*

## Declaration

```
int getNumFields()
```

## Purpose

Extract the number of fields in a message.

## Remarks

This method counts the immediate fields of the message; it does not descend into submessages to count their fields recursively.

# TibrvMsg.getReplySubject()

*Method*

## Declaration

```
java.lang.String getReplySubject()
```

## Purpose

Extract the reply subject from a message.

## Remarks

The reply subject string is part of a message's address information—it is *not* part of the message itself.

If the reply subject is not set, this method returns null.

## See Also

[TibrvMsg.setReplySubject\(\)](#)

Supplementary Information for Messages in TIBCO Rendezvous Concepts

# TibrvMsg.getSendSubject()

*Method*

## Declaration

```
java.lang.String getSendSubject()
```

## Purpose

Extract the subject from a message.

## Remarks

The subject string is part of a message's address information—it is *not* part of the message itself.

If the destination subject is not set, this method returns null.

## See Also

[TibrvMsg.setSendSubject\(\)](#)

[Supplementary Information for Messages](#) in TIBCO Rendezvous Concepts

# TibrvMsg.getStringEncoding()

*Method*

## Declaration

```
static java.lang.String getStringEncoding()
```

## Purpose

Return the character encoding for converting between Java Strings and wire format strings.

## See Also

[Strings and Character Encodings](#)

[TibrvMsg.setStringEncoding\(\)](#)

# TibrvMsg.getTypeName()

*Method*

## Declaration

```
static java.lang.String getTypeName(short type)
```

## Purpose

Convert a type designator to its corresponding string name.

## Remarks

This static method returns string representations of the [TibrvMsg](#) datatype constants. For example, it converts the type constant `TibrvMsg.I8ARRAY` to the string `"I8ARRAY"`.

For custom datatypes, this method returns a string of the form `"USER_FIRST+3"` (for example).

If the type is invalid, this method returns the string `INVALID`.

Parameter	Description
type	Convert this type constant to a string name.

## See Also

[Datatype Constants](#)

[TibrvMsg.setHandlers\(\)](#)

# TibrvMsg.removeField()

*Method*

## Declaration

```
boolean removeField(  
    java.lang.String fieldName)  
boolean removeField(  
    int fieldId)  
boolean removeField(  
    java.lang.String fieldName,  
    int fieldId)  
throws TibrvException
```

## Purpose

Remove a field from a message.

Parameter	Description
fieldName	Remove the field with this name.
fieldId	Remove the field with this identifier.

## Method Forms and the Field Search Algorithm

This method uses this algorithm to find and remove a field within a message, as specified by a field identifier and a field name.

### Procedure

1. If the program supplied zero as the identifier, or omitted any identifier, then begin at step 3.  
If the program supplied a *non-zero* field identifier, then search for the field with that identifier. If the search succeeds, remove the field and return true.  
On the search does not find a field, continue to step 2.

2. If the identifier search (in step 1) fails, and the program supplied a non-null field name, then search for a field with that name.

On the search does not find a field, or if the program supplied null as the field name, return false.

If the name search succeeds, but the actual identifier in the field is non-zero (so it does not match the identifier supplied) then throw an exception with the status code [TibrvStatus.ID\\_CONFLICT](#).

If the search succeeds, remove the field and return true.

3. When the program supplied zero as the identifier, or omitted any identifier, then begin here.

Search for a field with the specified name—even if that name is null.

If the search succeeds, remove the field and return true.

If the search does not find a field, return false.

If a message contains several fields with the same name, searching by name removes the first instance of the field with that name.

# TibrvMsg.removeFieldInstance()

*Method*

## Declaration

```
boolean removeFieldInstance(  
    java.lang.String fieldName,  
    int instance)  
throws TibrvException
```

## Purpose

Remove a specified instance of a field from a message.

## Remarks

When a message contains several field instances with the same field name, remove a specific instance by number (for example, remove the *ith* field named foo). Programs can use this method in a loop that examines every field with a specified name.

The argument 1 denotes the first instance of the named field.

If the specified instance does not exist, the method returns false.

Parameter	Description
fieldName	Remove the field with this name.
instance	Remove this instance of the field. The argument 1 specifies the first instance of the named field.



# TibrvMsg.reset()

*Method*

## Declaration

```
void reset()  
throws TibrvException
```

## Purpose

Clear a message, preparing it for re-use.

## Remarks

This method is the equivalent of creating a new message—except that the unmanaged storage is re-used.

When this method returns, the message has no fields; it is like a newly created message. The message's address information is also reset.

## See Also

[TibrvMsg\(\)](#)

# TibrvMsg.setReplySubject()

*Method*

## Declaration

```
void setReplySubject(java.lang.String replySubject)  
    throws TibrvException
```

## Purpose

Set the reply subject for a message.

## Remarks

A receiver can reply to an inbound message using its reply subject.

Rendezvous routing daemons modify subjects and reply subjects to enable transparent point-to-point communication across network boundaries. This modification does not apply to subject names stored in message data fields; we discourage storing point-to-point subject names in data fields.

Parameter	Description
replySubject	Use this string as the new reply subject, replacing any existing reply subject.  The reply subject null removes the previous reply subject.

## See Also

[TibrvMsg.getReplySubject\(\)](#)

Supplementary Information for Messages in TIBCO Rendezvous Concepts

# TibrvMsg.setSendSubject()

*Method*

## Declaration

```
void setSendSubject(java.lang.String subject)  
    throws TibrvException
```

## Purpose

Set the subject for a message.

## Remarks

The subject of a message can describe its content, as well as its destination set.

Rendezvous routing daemons modify subjects and reply subjects to enable transparent point-to-point communication across network boundaries. This modification does not apply to subject names stored in message data fields; we discourage storing point-to-point subject names in data fields.

Parameter	Description
subject	Use this string as the new subject, replacing any existing subject.  The subject null removes the previous subject, leaving the message unsendable.

## See Also

[TibrvMsg.getSendSubject\(\)](#)

[Supplementary Information for Messages](#) in TIBCO Rendezvous Concepts

# TibrvMsg.setStringEncoding()

*Method*

## Declaration

```
static void setStringEncoding(  
    java.lang.String encoding)  
    throws java.io.UnsupportedEncodingException
```

## Purpose

Set the character encoding for converting between Java Unicode strings and wire format strings.

## Remarks

This method overrides the default string encoding for all strings in all messages.

null indicates the ISO 8859-1 (Latin-1) encoding:

- To translate Unicode strings to Latin-1 strings, disregard the *null* high byte of each character. Unicode characters with non-null high bytes are out of the Latin-1 range.
- To translate Latin-1 strings to Unicode strings, pad each character with a null high byte.



### Warning

Do *not* call this method while any listener events are valid. We recommend setting it at program start, before creating any listeners.



### Note

Encoding changes are not retroactive; that is, changing the encoding affects only future string translations. For further details, see [Strings and Character Encodings](#).

Parameter	Description
encoding	Use this encoding.

## See Also

[Strings and Character Encodings](#)

[TibrvMsg.getStringEncoding\(\)](#)

[www.unicode.org](http://www.unicode.org)

# TibrvMsg.toString()

*Method*

## Declaration

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

## Purpose

Format a message as a string.

## Remarks

Programs can use this method to obtain a string representation of the message for printing.

For most datatypes, this method formats the full value of the field to the output string; these types are exceptions:

**TibrvMsg.OPAQUE** This method abbreviates the value of an opaque field; for example, [472 opaque bytes].

**TibrvMsg.XML** This method abbreviates the value of an XML field; for example, [XML document: 472 bytes].

The size measures *uncompressed* data.

This method formats **TibrvMsg.IPADDR32** fields as four dot-separated decimal integers.

This method formats **TibrvMsg.IPPORT16** fields as one decimal integer.

# TibrvMsg.update()

*Method*

## Declaration

```
void update(  
    java.lang.String fieldName,  
    java.lang.Object data)  
    throws TibrvException  
void update(  
    java.lang.String fieldName,  
    java.lang.Object data,  
    short          type)  
    throws TibrvException  
void update(  
    java.lang.String fieldName,  
    java.lang.Object data,  
    short          type,  
    int            fieldId)  
    throws TibrvException
```

## Purpose

Update a field within a message.

## Remarks

This method copies the new data into the message field. All related convenience methods behave similarly.

This method locates a field within the message by matching the `fieldName` and `fieldId` arguments. Then it updates the message field using the `data` argument. (Notice that only the value and count of the message field can change.)

If no existing field matches the specifications in the `fieldName` and `fieldId` arguments, then this method adds a new field to the message. Update convenience methods also add a field if it is not present.

The type of the existing message field and the type of the updating field argument must be identical; otherwise, the method throws an exception with the error status code

[TibrvStatus.INVALID\\_TYPE](#). However, when updating array or vector fields, the count (number of elements) can change.

Parameter	Description
fieldName	Update a field with this name.  When absent, locate the field by identifier only.
data	Update a field using this data value.  It is illegal to add or update a field with null data. To remove a field, use <a href="#">TibrvMsg.removeField()</a> .
type	Update a field with this type.  When absent, determine the field's type from the type of the data.  Default encodings and possible conversions are identical to methods that add fields; see <a href="#">Java to Wire Format Datatype Conversion Matrix</a> .
fieldId	Update a field with this identifier. All field identifiers must be unique within each message.  Zero is a special value, indicating no identifier. It is illegal to add a field that has both a null field name, and a non-zero field identifier.

## Field Search Algorithm

This method, and related methods that *update* message fields, all use this algorithm to find and update a field within a message, as specified by a field identifier and a field name.

### Procedure

1. If the program supplied zero as the identifier, or omitted any identifier, then begin at step 3.  
  
If the program supplied a *non-zero* field identifier, then search for the field with that identifier.  
  
If the search succeeds, then update that field.



On failure, continue to step 2.

2. If the identifier search (in step 1) fails, and the program supplied a non-null field name, then search for a field with that name.

If the search succeeds, then update that field.

If the name search succeeds, but the actual identifier in the field is non-null (so it does not match the identifier supplied) then throw an exception with the status code [TibrvStatus.ID\\_CONFLICT](#).

If the search fails, *add* the field as specified (with name and identifier).

However, if the program supplied null as the field name, then do not search for the field name; instead, throw an exception with the status code [TibrvStatus.NOT\\_FOUND](#).

3. When the program supplied zero as the identifier, or omitted any identifier, then begin here.

Search for a field with the specified name—even if that name is null.

If the search fails, *add* the field as specified (with name and identifier).

If a message contains several fields with the same name, searching by name finds the first instance of the field with that name.

## Nested Message

When the new value is a message object, this method uses only the data portion of the nested message (data); it does not include any address information or certified delivery information.

# Update Scalar

## Convenience Methods

### Declaration

```
void update(  
    java.lang.String fieldName,  
    scalar_type data)  
throws TibrvException  
void update(  
    java.lang.String fieldName,  
    scalar_type data,  
    int fieldId)  
throws TibrvException  
void updateUbits(  
    java.lang.String fieldName,  
    scalar_type data)  
throws TibrvException  
void updateUbits(  
    java.lang.String fieldName,  
    scalar_type data,  
    int fieldId)  
throws TibrvException
```

### Purpose

Update a field containing a scalar value.

### Method Forms

The convenience methods named **update()** determine the field type from the numeric type of the data.

The convenience methods named **updateU $nn$ ()** update unsigned integer fields, discarding the sign bit of a Java integer data value.

Method Name	Java Data Type	Field Type	Type Description
update	boolean	TibrvMsg.BOOL	boolean

Method Name	Java Data Type	Field Type	Type Description
update	float	TibrvMsg.F32	32-bit floating point
update	double	TibrvMsg.F64	64-bit floating point
update	byte	TibrvMsg.I8	8-bit integer
update	short	TibrvMsg.I16	16-bit integer
update	int	TibrvMsg.I32	32-bit integer
update	long	TibrvMsg.I64	64-bit integer
updateU8	byte	TibrvMsg.U8	8-bit unsigned integer
updateU16	short	TibrvMsg.U16	16-bit unsigned integer
updateU32	int	TibrvMsg.U32	32-bit unsigned integer
updateU64	long	TibrvMsg.U64	64-bit unsigned integer

Parameter	Description
data	Update a field with this data value.  It is illegal to add or update a field with null data.
fieldName	Update a field with this name.  When absent, locate the field by identifier only.
fieldId	Update a field with this identifier. All field identifiers must be unique within each message.  Zero is a special value, indicating no identifier. It is illegal to add a field that has both a null field name, and a non-zero field identifier.

# TibrvMsg.updateField()

*Method*

## Declaration

```
void updateField(TibrvMsgField field)  
    throws TibrvException
```

## Purpose

Update a field within a message.

## Remarks

This method copies the new data into the existing message field. All related convenience methods behave similarly.

This method locates a field within the message by matching the name and identifier of field. Then it updates the message field using the field argument. (Notice that the program may not supply a field object with a different field name, field identifier, or datatype.)

If no existing field matches the specifications in the field argument, then this method adds the field to the message. Update convenience methods also add the field if it is not present. It is illegal to add a field that has both a null field name, and a non-zero field identifier.

The type of the existing message field and the type of the updating field argument must be identical; otherwise, the method returns the error status code [TibrvStatus.INVALID\\_TYPE](#). However, when updating array or vector fields, the count (number of elements) can change.

Parameter	Description
field	Update the existing message field using this field.  It is illegal to add or update a field with null data.

# TibrvMsgField

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvMsgField  
    extends java.lang.Object
```

## Purpose

Represent a message field.

## Remarks

This class has no `destroy()` method. Instead, the Java garbage collector reclaims storage automatically.

Field	Description
name	<p>Field name, of type <code>java.lang.String</code>.</p> <p>Names must be strings; null is a special value that indicates no name.</p> <p>Field names use the character encoding appropriate to the ISO locale; see <a href="#">Strings and Character Encodings</a>.</p>
id	<p>Field identifier, of type <code>int</code>.</p> <p>Identifiers must be in the range [0-65535]; zero is a special value that indicates no identifier.</p>
data	<p>Data content of the field, a <code>java.lang.Object</code>.</p> <p>When storing data in this field, Rendezvous software does not verify that the data and type of the field are consistent. Methods that add the field object into a message verify consistency (for example <a href="#">TibrvMsg.addField()</a> and <a href="#">TibrvMsg.updateField()</a>).</p>

Field	Description
	It is illegal to add or update a field with null data.
type	<p>Type designator for the data, a short.</p> <p>For a list of types, see <a href="#">Datatype Constants</a>. To interpret a type designator as a string, see <a href="#">TibrvMsg.getTypeName()</a>.</p> <p>When storing a type designator in this field, Rendezvous software does not verify that the data and type of the field are consistent. Methods that add the field object into a message verify consistency (for example <a href="#">TibrvMsg.addField()</a> and <a href="#">TibrvMsg.updateField()</a>).</p> <p>If absent, the default type is <a href="#">TibrvMsg.DEFAULT</a>, which instructs <a href="#">TibrvMsg.addField()</a> and <a href="#">TibrvMsg.updateField()</a> to use the default type corresponding to the data value.</p>

Method	Description
<a href="#">TibrvMsgField()</a>	Create a message field object.
<a href="#">TibrvMsgField.toString()</a>	Format a field as a string.

### Inherited Methods

[java.lang.Object.equals](#)  
[java.lang.Object.getClass](#)  
[java.lang.Object.hashCode](#)  
[java.lang.Object.notify](#)  
[java.lang.Object.notifyAll](#)  
[java.lang.Object.wait](#)

## See Also

[TibrvMsg.addField\(\)](#)

[TibrvMsg.getField\(\)](#)

`TibrvMsg.updateField()`

# TibrvMsgField()

*Constructor*

## Declaration

```
TibrvMsgField()  
TibrvMsgField(TibrvMsgField field)  
TibrvMsgField(  
    java.lang.String name,  
    java.lang.Object data,  
    short type,  
    int id)  
TibrvMsgField(  
    java.lang.String name,  
    java.lang.Object data,  
    short type)  
TibrvMsgField(  
    java.lang.String name,  
    java.lang.Object data)
```

## Purpose

Create a message field object.

## Method Forms

With no arguments, create an empty field (no name, no identifier, no data, no type).

With a field argument, create an independent copy of the field.

Name, identifier, data and type arguments contribute to the contents of the field object.

## Remarks

This constructor does not verify that the data and type of the field are consistent. Methods that add the field object into a message verify consistency (for example [TibrvMsg.addField\(\)](#) and [TibrvMsg.updateField\(\)](#)).



Parameter	Description
field	Create an independent copy of this field.
name	Create a field with this name.  To create a field with no name, supply null.
id	Create a field with this identifier.  Identifiers must be in the range [0-65535]; zero is a special value that indicates no identifier.
data	Create a field with this data value.  The data value must be non-null.
type	Create a field with this type.  For a list of types, see <a href="#">Datatype Constants</a> .  If absent, the default type is <code>TibrvMsg.DEFAULT</code> , which instructs the methods <code>TibrvMsg.addField()</code> and <code>TibrvMsg.updateField()</code> to use the default type corresponding to the data value.

## See Also

[TibrvMsg.addField\(\)](#)

[TibrvMsg.updateField\(\)](#)

# TibrvMsgField.toString()

*Method*

## Declaration

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

## Purpose

Format a field as a string.

## Remarks

Programs can use this method to obtain a string representation of the field for printing.

# TibrvMsgView

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvMsgView  
    extends java.lang.Object
```

## Purpose

View the fields of Rendezvous messages.

## Remarks

A message view is a snapshot copy of a TIBCO Rendezvous message. All the data resides within the Java environment (not in the C environment). A view provides read-only access to field data of an inbound message.

To create a new view, see [TibrvMsgView.extract\(\)](#).

## Efficiency

When a callback method gets 10 or more fields, then a message view is usually more efficient than accessing the fields of the C message. Extracting a message view accesses the native C message only once, instead of repeatedly (with every get call).

When a callback method gets fewer than 10 fields, then getting fields from the native C message is usually more efficient.

## Restrictions

TibrvMsgView and its methods are available only in the JNI preferred implementation. They are not available in the JNI backward compatibility implementation, nor in the pure Java implementation.

Method	Description
--------	-------------

## Message View Life Cycle and Properties

<a href="#">TibrvMsgView.extract()</a>	Extract a snapshot view of a TIBCO Rendezvous message.
--	--

## Fields

<a href="#">TibrvMsgView.get()</a>	Get the value of a specified field from a message view snapshot.
------------------------------------	--

<a href="#">Get Scalar from TibrvMsgView</a>	Get the value of a field from a message view snapshot as a scalar value.
--	--

<a href="#">TibrvMsgView.getField()</a>	Get a specified field from a snapshot message view.
---	---

<a href="#">TibrvMsgView.getFieldByIndex()</a>	Get a field from a snapshot message view by an index.
--	---

<a href="#">TibrvMsgView.getFieldInstance()</a>	Get a specific instance of a field from a message.
---	--

<a href="#">TibrvMsgView.getNumFields()</a>	Extract the number of fields in a snapshot message view.
---	--

## Inherited Methods

[java.lang.Object.equals](#)  
[java.lang.Object.getClass](#)  
[java.lang.Object.hashCode](#)  
[java.lang.Object.notify](#)  
[java.lang.Object.notifyAll](#)  
[java.lang.Object.wait](#)

## Using a Message View

```
try {
    TibrvMsgView view = TibrvMsgView.extract(message);
    for (int i = 1; i <= this.fieldCount; i++) {
        view.get(i);
        ...
    }
} catch (final TibrvException e) {
    e.printStackTrace();
    System.exit(0);
}
```

## See Also

[Strings and Character Encodings](#)

[TibrvMsg](#)

[TibrvMsgField](#)

[Custom Datatypes](#)

# TibrvMsgView.extract()

*Method*

## Declaration

```
static TibrvMsgView extract(TibrvMsg message)
    throws TibrvException
```

## Purpose

Extract a snapshot view of a TIBCO Rendezvous message.

## Remarks

This static method parses a snapshot of the native C message into a view of the message data. Upon successful completion, all the message data are available as Java objects.

The message view contains snapshot data. That is, subsequently modifying the data in the message does not affect the data in the message view.

Parameter	Description
message	Extract the data from this message into a new view object.

## Submessages

When a field contains a submessage, this method creates a copy of the submessage—that is, a Java message object corresponding to an independent copy of the submessage in the C environment. You may access its fields as a message object, or you may explicitly extract the submessage into its own message view.

## Decoding and Type Conversion

The method [TibrvMsgView.extract\(\)](#) automatically decodes the extracted field data from its Rendezvous wire format type to a corresponding Java type. In [Wire Format to Java Datatype Conversion Matrix](#), filled circles (as well as + and - symbols) specify default

decodings between homologous types; for some types you can override the default decoding by using convenience calls that force specific types (see [Get Scalar from TibrvMsgView](#)).

Java does not admit unsigned integers. When extracting an unsigned integer from an inbound message field, this method automatically promotes the value to the corresponding Java type that is large enough to contain it ([Wire Format to Java Datatype Conversion Matrix](#) indicates this with +). When extracting an unsigned 64-bit integer, Java does not have a type that can contain a number that uses all 64 bits; this method decodes it to a Java long, interpreting the high bit as a sign bit ([Wire Format to Java Datatype Conversion Matrix](#) indicates this with -).

## See Also

[Wire Format to Java Datatype Conversion Matrix](#)

[TibrvMsgView](#)

# TibrvMsgView.get()

*Method*

## Declaration

```
java.lang.Object get(  
    final java.lang.String fieldName)  
    throws TibrvException  
java.lang.Object get(  
    final int fieldId)  
    throws TibrvException  
java.lang.Object get(  
    final java.lang.String fieldName,  
    final int fieldId)  
    throws TibrvException
```

## Purpose

Get the value of a specified field from a message view snapshot.

## Remarks

Programs specify the field to retrieve using the `fieldName` and `fieldId` parameters.

When the field contains a submessage, this method returns a copy of the submessage—that is, a Java message object corresponding to an independent copy of the submessage in the C environment. You may access its fields as a message object, or you may explicitly extract the submessage into its own message view.

Programs can use a related method to loop through all the fields of a message; to retrieve each field by its integer index number, see [TibrvMsg.getFieldByIndex\(\)](#).

Parameter	Description
<code>fieldName</code>	Get a field with this name.
<code>fieldId</code>	Get the field with this identifier.



## Field Search Algorithm

This method, and related methods that *get* fields from message views, use the same algorithm as [TibrvMsg.get\(\)](#). For complete details, see [Field Search Algorithm](#).

### Method Forms

With only a field name, find the field by name. If the field name is not present in the message, return null. If several fields with that name are present in the message, this method returns the first one that it finds.

With only a field identifier, find the field with that identifier (since identifiers are unique, the message can contain at most one such field). If the identifier is not present in the message, return null.

With both a field name and a field identifier, search first by identifier, and then by field name. If neither are present in the message, return null. If identifier search succeeds, return the field value. If the name search succeeds, but the actual identifier in the field is non-zero (so it does not match the identifier supplied) then throw a [TibrvException](#) with status code [TibrvStatus.ID\\_CONFLICT](#).

### See Also

[TibrvMsgView](#)

# Get Scalar from TibrvMsgView

## Convenience Methods

### Declaration

```
scalar_type getScalar_type(  
    java.lang.String fieldName,  
    int fieldId)  
    throws TibrvException  
scalar_type getAsScalar_type(  
    java.lang.String fieldName,  
    int fieldId)  
    throws TibrvException
```

### Purpose

Get the value of a field from a message view snapshot as a scalar value.

### Remarks

Each convenience method in this family retrieves a field and extracts its data. If the field's type (as it exists) does not match the type of the convenience method, then the **gettype** method throws an exception; in contrast, the **getAs** method attempts to convert the data (see [Decoding and Type Conversion](#), and [Wire Format to Java Datatype Conversion Matrix](#)). If conversion is not possible, the method throws an exception with status code [TibrvStatus.CONVERSION\\_FAILED](#).

Parameter	Description
fieldName	Get a field with this name.
fieldId	Get the field with this identifier.

Method Name	Field Type	Java Type	Type Description
<code>getBoolean</code> <code>getAsBoolean</code>	<code>TibrvMsg.BOOL</code>	<code>boolean</code>	<code>boolean</code>
<code>getFloat</code> <code>getAsFloat</code>	<code>TibrvMsg.F32</code>	<code>float</code>	32-bit floating point
<code>getDouble</code> <code>getAsDouble</code>	<code>TibrvMsg.F64</code>	<code>double</code>	64-bit floating point
<code>getByte</code> <code>getAsByte</code>	<code>TibrvMsg.I8</code>	<code>byte</code>	8-bit integer
<code>getShort</code> <code>getAsShort</code>	<code>TibrvMsg.I16</code>	<code>short</code>	16-bit integer
<code>getInt</code> <code>getAsInt</code>	<code>TibrvMsg.I32</code>	<code>int</code>	32-bit integer
<code>getLong</code> <code>getAsLong</code>	<code>TibrvMsg.I64</code>	<code>long</code>	64-bit integer

# TibrvMsgView.getField()

*Method*

## Declaration

```
TibrvMsgField getField(  
    java.lang.String fieldName)  
    throws TibrvException  
TibrvMsgField getField(  
    int fieldId)  
    throws TibrvException  
TibrvMsgField getField(  
    java.lang.String fieldName,  
    int fieldId)  
    throws TibrvException
```

## Purpose

Get a specified field from a snapshot message view.

## Remarks

Programs specify the field to retrieve using the `fieldName` and `fieldId` parameters.

The method [TibrvMsgView.extract\(\)](#) creates field objects for each field of the message snapshot. This returns one of those field objects.

Programs can use a related method to loop through all the fields of a message view; to retrieve each field by its integer index number, see [TibrvMsgView.getFieldByIndex\(\)](#).

Parameter	Description
<code>fieldName</code>	Get a field with this name.
<code>fieldId</code>	Get the field with this identifier.

## Method Forms

With only a field name, find the field by name. If the field name is not present in the message, return null. If several fields with that name are present in the message, this method returns the first one that it finds.

With only a field identifier, find the field with that identifier (since identifiers are unique, the message can contain at most one such field). If the identifier is not present in the message, return null.

With both a field name and a field identifier, search first by identifier, and then by field name. If neither are present in the message, return null.

## See Also

[TibrvMsgView](#)

[TibrvMsgView.get\(\)](#)

# TibrvMsgView.getFieldByIndex()

*Method*

## Declaration

```
TibrvMsgField getFieldByIndex(int fieldIndex)  
throws ArrayIndexOutOfBoundsException TibrvException
```

## Purpose

Get a field from a snapshot message view by an index.

## Remarks

Programs can loop through all the fields of a message, to retrieve each field in turn using an integer index.

The method [TibrvMsgView.extract\(\)](#) creates field objects for each field of the message snapshot. This returns one of those field objects.

Parameter	Description
fieldIndex	Get the field with this index. Zero specifies the first field.

## See Also

[TibrvMsgView](#)

[TibrvMsgView.get\(\)](#)

# TibrvMsgView.getFieldInstance()

*Method*

## Declaration

```
TibrvMsgField getFieldInstance(  
    java.lang.String fieldName,  
    int instance)  
throws TibrvException
```

## Purpose

Get a specific instance of a field from a message.

## Remarks

When a message contains several field instances with the same field name, retrieve a specific instance by number (for example, get the *ith* field named foo). Programs can use this method in a loop that examines every field with a specified name.

The argument 1 denotes the first instance of the named field.

The method [TibrvMsgView.extract\(\)](#) creates field objects for each field of the message snapshot. This returns one of those field objects.

When the instance argument is greater than the actual number of instances of the field in the message, this method returns null.

## Release 5 Interaction

Rendezvous 5 (and earlier) did not support array datatypes. Some older programs circumvented this limitation by using several fields with the same name to simulate arrays. This work-around is no longer necessary, since release 6 (and later) supports array datatypes within message fields. The method [TibrvMsgView.getFieldInstance\(\)](#) ensures backward compatibility, so new programs can still receive and manipulate messages sent from older programs. Nonetheless, we encourage programmers to use array types as appropriate, and we discourage storing several fields with the same name in a message.

Parameter	Description
fieldName	Get an instance of the field with this name. null specifies the empty string as the field name.
instance	Get this instance of the specified field name. The argument 1 denotes the first instance of the named field.

## See Also

[TibrvMsgField](#)

[TibrvMsgView](#)

[TibrvMsgView.get\(\)](#)



# TibrvMsgView.getNumFields()

*Method*

## Declaration

```
int getNumFields()
```

## Purpose

Extract the number of fields in a snapshot message view.

## Remarks

This method counts the immediate fields of the message; it does not descend into submessages to count their fields recursively.

# TibrvDate

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvDate  
    extends java.util.Date
```

## Purpose

Represent date and time.

This object relies on the method `java.util.Date.toString()` to convert a value to a string.

## Remarks

This class has no `destroy()` method. Instead, the Java garbage collector reclaims storage automatically.

Constant	Description
<code>TibrvDate.MAX_SECONDS</code>	<p>Maximum date (in seconds) that this class can represent.</p> <p>The value is 549,755,813,887 (the maximum value of a 40-bit signed integer), which is approximately 17,432 years after the year 1970.</p>
<code>TibrvDate.MIN_SECONDS</code>	<p>Minimum date (in seconds) that this class can represent.</p> <p>The value is -549,755,813,888 (the minimum value of a 40-bit signed integer), which is approximately 17,432 years before the year 1970.</p>

Method	Description
<a href="#">TibrvDate()</a>	Create a Rendezvous date object.

Method	Description
<a href="#">TibrvDate.getTimeNanoseconds()</a>	Extract the modifying value (in nanoseconds) of a date object.
<a href="#">TibrvDate.getTimeSeconds()</a>	Extract the partial value (in whole seconds) of a date object.
<a href="#">TibrvDate.setTime()</a>	Change the value of a date object.

### Inherited Methods

[java.util.Date.after](#)  
[java.util.Date.before](#)  
[java.util.Date.clone](#)  
[java.util.Date.compareTo](#)  
[java.util.Date.getDate](#)  
[java.util.Date.getDay](#)  
[java.util.Date.getHours](#)  
[java.util.Date.getMinutes](#)  
[java.util.Date.getMonth](#)  
[java.util.Date.getSeconds](#)  
[java.util.Date.getTime](#)  
[java.util.Date.getTimezoneOffset](#)  
[java.util.Date.getYear](#)  
[java.util.Date.hashCode](#)  
[java.util.Date.parse](#)  
[java.util.Date.setDate](#)  
[java.util.Date.setHours](#)  
[java.util.Date.setMinutes](#)  
[java.util.Date.setMonth](#)  
[java.util.Date.setSeconds](#)

Inherited Methods
java.util.Date.setYear
java.util.Date.toGMTString
java.util.Date.toLocaleString
java.util.Date.toString
java.util.Date.UTC
java.lang.Object.equals
java.lang.Object.getClass
java.lang.Object.hashCode
java.lang.Object.notify
java.lang.Object.notifyAll
java.lang.Object.wait

Representations

In all three representations, zero denotes the epoch, 12:00 midnight, January 1st, 1970. Range limits in this table denote the extreme value on either side of that center. Bold type indicates the primary unit of measurement for each representation.

Date and Time Representations

Representation	Details
java.util.Date	Milliseconds as a 64-bit signed integer.
	range in years292,471,208
	range in seconds9,223,372,036,854,775
	range in milliseconds <b>9,223,372,036,854,775,807</b>
TibrvDate	Seconds as a 64-bit signed integer, plus nanoseconds as a 32-bit unsigned integer.

Representation	Details
	However, values are restricted to the range and granularity supported by Rendezvous wire format. Forcing larger or finer values into this representation causes an exception.
	range in years 292,471,208,677
	<b>range in seconds 9,223,372,036,854,775,807</b>
	restricted range in seconds 549,755,813,887
	restricted range in milliseconds 549,755,813,887,000
Rendezvous wire format	Seconds as a 40-bit signed integer, plus microseconds as a 24-bit unsigned integer.
	range in years 17,432
	<b>range in seconds 549,755,813,887</b>
	range in milliseconds 549,755,813,887,000

## See Also

[TibrvMsg.get\(\)](#)

# TibrvDate()

## Constructor

## Declaration

```
TibrvDate()  
TibrvDate(java.util.Date date)  
TibrvDate(long milliseconds)  
TibrvDate(  
    long seconds,  
    int nanoseconds)
```

## Purpose

Create a Rendezvous date object.

## Remarks

If the time value specified is out of range, this constructor throws an `IllegalArgumentException`.

Parameter	Description
date	Copy the value of this Java date or Rendezvous date object into an independent copy object.
milliseconds	Create a date from this value (in milliseconds, centered on 12:00am, January 1, 1970).
seconds	Create a date from this value (in seconds, centered on 12:00am, January 1, 1970).
nanoseconds	Add this value (in nanoseconds) to the time specified by the seconds parameter. This argument must be non-negative.

## Method Forms

With no arguments, create a date object representing the current time.

With an argument of class `java.util.Date` (or a subclass), create a [TibrvDate](#) object representing the same date value.

With one argument, interpret that long as the date in milliseconds. For example, specify the time 1/2 second before midnight of December 31, 1969 as -500 milliseconds.

With two arguments, interpret them as seconds and (non-negative) nanoseconds. For example, specify the time 1/2 second before midnight of December 31, 1969 as -1 seconds plus 500,000,000 nanoseconds.

## See Also

[TibrvDate](#)

[Representations](#)

# TibrvDate.getTimeNanoseconds()

*Method*

## Declaration

```
int getTimeSeconds()
```

## Purpose

Extract the modifying value (in nanoseconds) of a date object.

## Remarks

This value is always non-negative, between zero and 999999999.

It modifies the date in whole seconds (as returned by [TibrvDate.getTimeSeconds\(\)](#)), by specifying the number of nanoseconds *after* that date. For example, the time 1/2 second before midnight of December 31, 1969 is -1 seconds plus 500,000,000 nanoseconds.

## See Also

[TibrvDate.getTimeSeconds\(\)](#)



# TibrvDate.getTimeSeconds()

*Method*

## Declaration

```
long getTimeSeconds()
```

## Purpose

Extract the partial value (in whole seconds) of a date object.

## Remarks

The value is the date in seconds, centered on 12:00am, January 1, 1970.

To get the modifying nanosecond value, use [TibrvDate.getTimeNanoseconds\(\)](#).

## See Also

[TibrvDate.getTimeNanoseconds\(\)](#)

# TibrvDate.setTime()

*Method*

## Declaration

```
void setTime(long milliseconds)
void setTime(
    long seconds,
    int nanoseconds)
```

## Purpose

Change the value of a date object.

## Remarks

If the time value specified is out of range, this constructor throws an `IllegalArgumentException`.

Parameter	Description
milliseconds	Set the date from this value (in milliseconds, centered on 12:00am, January 1, 1970).
seconds	Set the a date from this value (in seconds, centered on 12:00am, January 1, 1970). This argument must be in the range [ <a href="#">TibrvDate.MIN_SECONDS</a> ; <a href="#">TibrvDate.MAX_SECONDS</a> ].
nanoseconds	Add this value (in nanoseconds) to the time specified by the seconds parameter. This argument must be non-negative, in the range [0;999999999].

## Method Forms

With one argument, interpret it as the date in milliseconds. For example, specify the time 1/2 second before midnight of December 31, 1969 as -500 milliseconds.

With two arguments, interpret them as seconds and (non-negative) nanoseconds. For example, specify the time 1/2 second before midnight of December 31, 1969 as -1 seconds plus 500,000,000 nanoseconds.

## See Also

[Representations](#)

# TibrvIPAddr

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvIPAddr  
    extends java.lang.Object
```

## Purpose

Represent an IP address.

## Remarks

In general, an IP address consists of four 8-bit unsigned integers, in network byte order.

This class has no `destroy()` method. Instead, the Java garbage collector reclaims storage automatically.

Method	Description
<a href="#">TibrvIPAddr()</a>	Create an IP address object.
<a href="#">TibrvIPAddr.getAddr()</a>	Get an IP address as a 32-bit integer.
<a href="#">TibrvIPAddr.getAsBytes()</a>	Get an IP address as a 4-byte array.
<a href="#">TibrvIPAddr.getAsString()</a>	Get an IP address as a String.

### Inherited Methods

`java.lang.Object.equals`

`java.lang.Object.getClass`

### Inherited Methods

---

`java.lang.Object.hashCode`

`java.lang.Object.notify`

`java.lang.Object.notifyAll`

`java.lang.Object.toString` (override)

`java.lang.Object.wait`

---

### See Also

[TibrvMsg.get\(\)](#)

# TibrvIPAddr()

## Constructor

## Declaration

```
TibrvIPAddr(byte[] bytes)
TibrvIPAddr(
    java.net.InetAddress inetAddr)
TibrvIPAddr(
    byte b1,
    byte b2,
    byte b3,
    byte b4)
TibrvIPAddr(
    java.lang.String ipString)
    throws java.lang.NumberFormatException
TibrvIPAddr(int address)
TibrvIPAddr(TibrvIPAddr ipAddr)
```

## Purpose

Create an IP address object.

Parameter	Description
bytes	Create an IP address from an array of 4 bytes. For aaa.bbb.ccc.ddd, let bytes[0] be the high byte aaa, and bytes[3] be the low byte ddd.
b1, b2, b3, b4	Create an IP address from these 4 bytes. For aaa.bbb.ccc.ddd, let b1 be the high byte aaa, and b4 be the low byte ddd.
address	Create an IP address from a 32-bit integer, interpreted as 4 bytes in network byte order.
ipString	Create an IP address from a string representation (for example, "aaa.bbb.ccc.ddd").

Parameter	Description
inetAddr	Copy this Java 4-byte Internet address into a <a href="#">TibrvIPAddr</a> object with an equivalent value. ( <a href="#">TibrvIPAddr</a> supports only 4-byte IP addresses.)
ipAddr	Make an independent copy of this <a href="#">TibrvIPAddr</a> object.

## Remarks

If the ipString argument does not represent a valid IP address, this constructor throws a `java.lang.NumberFormatException`.

# TibrvIPAddr.getAddr()

*Method*

## Declaration

```
final int getAddr()
```

## Purpose

Get an IP address as a 32-bit integer.



# TibrvIPAddr.getAsBytes()

*Method*

## Declaration

```
final byte[] getAsBytes()
```

## Purpose

Get an IP address as a 4-byte array.

## Remarks

The zero<sup>th</sup> element is the high byte.

# TibrvIPAddr.getAsString()

*Method*

## Declaration

```
final java.lang.String getAsString()
```

## Purpose

Get an IP address as a String.

## Remarks

This method returns a string composed of four decimal integers (in the form "aaa.bbb.ccc.ddd").

# TibrvIPPort

## Class

## Declaration

```
class com.tibco.tibrv.TibrvIPPort  
    extends java.lang.Object
```

## Purpose

Represent an IP port number.

## Remarks

In general, an IP Port number is an unsigned 16-bit integer [0;65535], in network byte order. This class represents a port number as a 32-bit integer, because Java does not support unsigned numbers.

This class has no `destroy()` method. Instead, the Java garbage collector reclaims storage automatically.

Constant	Description
TibrvIPPort.MAX_PORT	Maximum port number that this class can represent (65535).
TibrvIPPort.MIN_PORT	Minimum port number that this class can represent (zero).

Method	Description
<a href="#">TibrvIPPort()</a>	<a href="#">Create an IP port object.</a>
<a href="#">TibrvIPPort.getPort()</a>	<a href="#">Get an IP port as a 32-bit integer.</a>
<a href="#">TibrvIPPort.getAsBytes()</a>	<a href="#">Get an IP port as a 2-byte array.</a>

## Inherited Methods

`java.lang.Object.equals`  
`java.lang.Object.getClass`  
`java.lang.Object.hashCode`  
`java.lang.Object.notify`  
`java.lang.Object.notifyAll`  
`java.lang.Object.toString` (override)  
`java.lang.Object.wait`

## See Also

[TibrvMsg.get\(\)](#)

# TibrvIPPort()

*Constructor*

## Declaration

```
TibrvIPPort(  
    byte highByte,  
    byte lowByte)  
TibrvIPPort(int port)  
TibrvIPPort(TibrvIPPort ipPort)
```

## Purpose

Create an IP port object.

Parameter	Description
highByte, lowByte	Create an IP port from these 2 bytes.
port	Create an IP port from the 2 low bytes of this 32-bit integer, which must be in the range [0;65535].
ipPort	Make an independent copy of this <a href="#">TibrvIPPort</a> object.

# TibrvIPPort.getPort()

*Method*

## Declaration

```
int getPort()
```

## Purpose

Get an IP port as a 32-bit integer.

## Remarks

The value is always in the range [0;65535].

# TibrvIPPort.getAsBytes()

*Method*

## Declaration

```
byte[] getAsBytes()
```

## Purpose

Get an IP port as a 2-byte array.

## Remarks

The high byte is the zero<sup>th</sup> element.

# TibrvXml

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvXml  
    extends java.lang.Object
```

## Purpose

Represent an XML byte array.

## Remarks

Within programs, XML data is represented as a byte array. Within message fields, Rendezvous software compresses the bytes for efficient network transmission.

This class has no `destroy()` method. Instead, the Java garbage collector reclaims storage automatically.

Method	Description
<a href="#">TibrvXml()</a>	Create an XML data object.
<a href="#">TibrvXml.getBytes()</a>	Get the data bytes from an XML data object.

### Inherited Methods

`java.lang.Object.equals`  
`java.lang.Object.getClass`  
`java.lang.Object.hashCode`  
`java.lang.Object.notify`  
`java.lang.Object.notifyAll`



### Inherited Methods

---

`java.lang.Object.toString (override)`

`java.lang.Object.wait`

---

### See Also

[TibrvMsg.get\(\)](#)

# TibrvXml()

*Constructor*

## Declaration

```
TibrvXml (byte[] xmlBytes)
```

## Purpose

Create an XML data object.

Parameter	Description
xmlBytes	Create an XML data object from this byte array.

# TibrvXml.getBytes()

*Method*

## Declaration

```
byte[] getBytes()
```

## Purpose

Get the data bytes from an XML data object.

# Events and Queues

---

Programs can express interest in events of two kinds—inbound messages and timers. When an event occurs, it triggers a program callback method to process the event. Events wait in queues until programs dispatch them. Dispatching an event runs its callback method to process the event.

Event queues organize events awaiting dispatch. Programs dispatch events to run callback methods.

Queue groups add flexibility and fine-grained control to the event queue dispatch mechanism. Programs can create groups of queues and dispatch them according to their queue priorities.

This section presents classes, methods, interfaces and types associated with event interest and event processing.

# TibrvEvent

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvEvent  
    extends java.lang.Object
```

## Purpose

Event objects represent program interest in events, and event occurrences.

## Remarks

Programs create instances of event subclasses of [TibrvEvent](#), but not of this superclass.

Each call to a Rendezvous event constructor results in a new event object, which represents your program's interest in a set of events. Rendezvous software uses the same event object to signal each occurrence of such an event.

Destroying an event object cancels the program's interest in that event. Destroying the queue or transport of an event automatically destroys the event as well.

Although the fault tolerance classes are technically events, they are sufficiently different from listeners and timers that they require separate description. See [Fault Tolerance](#).

Method	Description
<a href="#">TibrvEvent.destroy()</a>	<a href="#">Destroy an event, canceling interest.</a>
<a href="#">TibrvEvent.getClosure()</a>	<a href="#">Extract the closure data of an event object.</a>
<a href="#">TibrvEvent.getQueue()</a>	<a href="#">Extract the queue of an event object.</a>
<a href="#">TibrvEvent.isValid()</a>	<a href="#">Test whether an event has been destroyed.</a>
<a href="#">TibrvEvent.isVectorListener()</a>	<a href="#">Test whether this event object is a vector listener.</a>

## Inherited Methods

`java.lang.Object.equals`  
`java.lang.Object.getClass`  
`java.lang.Object.hashCode`  
`java.lang.Object.notify`  
`java.lang.Object.notifyAll`  
`java.lang.Object.toString` (override)  
`java.lang.Object.wait`

## Descendants

[TibrvListener](#)  
[TibrvCmListener](#)  
[TibrvTimer](#)  
[TibrvFtMember](#)  
[TibrvFtMonitor](#)

# TibrvEvent.destroy()

*Method*

## Declaration

```
void destroy()
```

## Purpose

Destroy an event, canceling interest.

## Remarks

Destroying an event object cancels interest in it. Upon return from [TibrvEvent.destroy\(\)](#), the destroyed event is no longer dispatched. However, all active callback methods of this event continue to run and return normally, even though the event is invalid.

It is legal for an event callback method to destroy its own event argument.

Destroying event interest invalidates the event object; subsequent API calls involving the invalid event throw exceptions, unless explicitly documented to the contrary.

## See Also

[TibrvEvent.isValid\(\)](#)

# TibrvEvent.getClosure()

*Method*

## Declaration

```
java.lang.Object getClosure()
```

## Purpose

Extract the closure data of an event object.

## Remarks

This method can extract the closure data even from invalid events.



# TibrvEvent.getQueue()

*Method*

## Declaration

```
TibrvQueue getQueue()
```

## Purpose

Extract the queue of an event object.

## Remarks

If the event is invalid, this method returns null.

# TibrvEvent.isValid()

*Method*

## Declaration

```
boolean isValid()
```

## Purpose

Test whether an event has been destroyed.

## Remarks

This method returns `false` if the event has been destroyed (using the `destroy` method); `true` otherwise.

Notice that [TibrvEvent.destroy\(\)](#) invalidates the event immediately, even though active callback methods may continue to run.

## See Also

[TibrvEvent.destroy\(\)](#)

# TibrvEvent.isVectorListener()

*Method*

## Declaration

```
java.lang.boolean isVectorListener();
```

## Purpose

Test whether this event object is a vector listener.

## Remarks

This method returns true when the event is a vector listener. Otherwise, it returns false.

## Restrictions

This method is available only in the JNI preferred implementation. It is not available in the JNI backward compatibility implementation, nor in the pure Java implementation.

## See Also

[TibrvVectorListener](#)

# TibrvListener

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvListener  
    extends TibrvEvent
```

## Purpose

Listen for inbound messages.

## Remarks

A listener object continues listening for messages until the program destroys it.

Programs must destroy instances of this class. Rendezvous software keeps internal references to these objects, so the Java garbage collector does not delete them automatically.

Destroying the queue or transport of an event automatically destroys the listener as well.

Method	Description
<a href="#">TibrvListener()</a>	Create a listener object to listen for inbound messages.
<a href="#">TibrvListener.getSubject()</a>	Extract the subject from a listener event object.
<a href="#">TibrvListener.getTransport()</a>	Extract the transport from a listener event object.

### Inherited Methods

[TibrvEvent.destroy\(\)](#)

### Inherited Methods

[TibrvEvent.getClosure\(\)](#)

[TibrvEvent.getQueue\(\)](#)

[TibrvEvent.isValid\(\)](#)

---

`java.lang.Object.equals`

`java.lang.Object.getClass`

`java.lang.Object.hashCode`

`java.lang.Object.notify`

`java.lang.Object.notifyAll`

`java.lang.Object.toString (override)`

`java.lang.Object.wait`

---

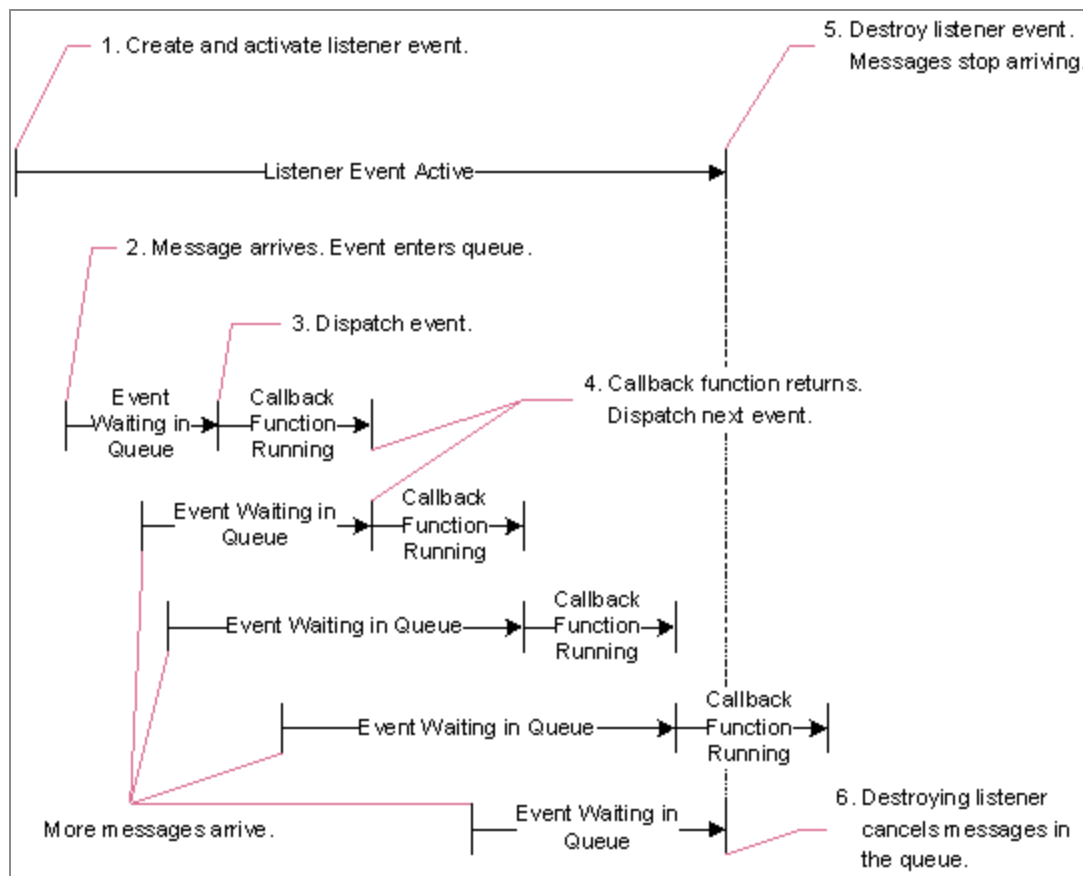
## Activation and Dispatch

Inbound messages on the transport that match the subject trigger the event.

This constructor creates a listener event object, and *activates* the event—that is, it begins listening for all inbound messages with matching subjects. When a message arrives, Rendezvous software places the event object and message on its event queue. Dispatch removes the event object from the queue, and runs the callback method to process the message. (To stop receiving inbound messages on the subject, destroy the event object; this action cancels all messages already queued for the listener event; see also [TibrvEvent.destroy\(\)](#).)

[Listener Activation and Dispatch](#) illustrates that messages can continue to accumulate in the queue, even while the callback method is processing.

Figure 4: Listener Activation and Dispatch



When the callback method is I/O-bound, messages can arrive faster than the callback method can process them, and the queue can grow unacceptably long. In programs where a delay in processing messages is unacceptable, consider dispatching from several threads to process messages concurrently.

## Descendants

[TibrvVectorListener](#)

[TibrvCmListener](#)

# TibrvListener()

*Constructor*

## Declaration

```
TibrvListener(  
    TibrvQueue    queue,  
    TibrvMsgCallback callback,  
    TibrvTransport transport,  
    java.lang.String subject,  
    java.lang.Object closure)  
throws TibrvException
```

## Purpose

Create a listener object to listen for inbound messages.

## Remarks

For each inbound message, place this event on the event queue.

Parameter	Description
queue	For each inbound message, place the event on this event queue.
callback	On dispatch, process the event with this interface implementation.
transport	Listen for inbound messages on this transport.
subject	Listen for inbound messages with subjects that match this specification. Wildcard subjects are permitted. The empty string is not a legal subject name.
closure	Store this closure data in the event object.

## Inbox Listener

To receive unicast (point-to-point) messages, listen to a unique inbox subject name. First call [TibrvTransport.createInbox\(\)](#) to create the unique inbox name; then call [TibrvListener\(\)](#) to begin listening. Remember that other programs have no information about an inbox until the listening program uses it as a reply subject in an outbound message.

## See Also

[TibrvEvent.destroy\(\)](#)

[TibrvMsgCallback](#)

[TibrvMsgCallback.onMsg\(\)](#)



# TibrvListener.getSubject()

*Method*

## Declaration

```
java.lang.String getSubject()
```

## Purpose

Extract the subject from a listener event object.

# TibrvListener.getTransport()

*Method*

## Declaration

```
TibrvTransport getTransport()
```

## Purpose

Extract the transport from a listener event object.

# TibrvMsgCallback

*Interface*

## Declaration

```
interface com.tibco.tibrv.TibrvMsgCallback
```

## Purpose

Process inbound messages (listener events).

## Remarks

Implement this interface to process inbound messages.

Method	Description
<a href="#">TibrvMsgCallback.onMsg()</a>	Process inbound messages (listener events).

## See Also

[TibrvListener\(\)](#)

# TibrvMsgCallback.onMsg()

*Method*

## Declaration

```
void onMsg(  
    TibrvListener listener,  
    TibrvMsg msg)
```

## Purpose

Process inbound messages (listener events).

Parameter	Description
listener	This parameter receives the listener event.
msg	This parameter receives the inbound message.

## Remarks

Implement this method to process inbound messages.

If your application requires a more complex processing arrangement, it can detach individual messages, and pass them to other threads for processing. To detach a message, use the copy constructor to make an independent copy of it. In the JNI implementation, you must detach a copy in order to use the message outside the scope of the callback. (If your program detaches a message, then you must also ensure that no references to the copy remain; such references could interfere with garbage collection. Furthermore, when using the TIBCO Rendezvous JNI preferred library, we recommend that programs call [TibrvMsg.dispose\(\)](#) to explicitly release the copy's storage within the native C environment.)

## CM Label Information

The callback method for certified delivery messages can use certified delivery (CM) label information to discriminate these situations:

- If [TibrvCmMsg.getSender\(\)](#) returns null, then the message uses the reliable protocol (that is, it was sent from an ordinary transport).
- If [TibrvCmMsg.getSender\(\)](#) returns a valid sender name, then the message uses the certified delivery protocol (that is, it is a labeled message, sent from a CM transport).

## See Also

[TibrvCmListener\(\)](#)

[TibrvCmMsg.getSender\(\)](#)

[TibrvCmMsg.getSequence\(\)](#)

[TibrvCmMsg.getTimeLimit\(\)](#)

# TibrvVectorListener

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvVectorListener  
    extends TibrvEvent
```

## Purpose

Listen for inbound messages, and receive them in a vector.

## Remarks

A vector listener object continues listening for messages until the program destroys it.

Programs must destroy instances of this class. Rendezvous software keeps internal references to these objects, so the Java garbage collector does not delete them automatically.

Destroying the queue or transport of a vector listener event automatically invalidates the vector listener as well.

Method	Description
<a href="#">TibrvVectorListener()</a>	Listen for inbound messages, and receive them in a vector.
<a href="#">TibrvVectorListener.getSubject()</a>	Extract the subject from a vector listener event object.
<a href="#">TibrvVectorListener.getTransport()</a>	Extract the transport from a vector listener event object.

## Inherited Methods

[TibrvEvent.destroy\(\)](#)[TibrvEvent.getClosure\(\)](#)[TibrvEvent.getQueue\(\)](#)[TibrvEvent.isValid\(\)](#)[TibrvEvent.isVectorListener\(\)](#)[java.lang.Object.equals](#)[java.lang.Object.getClass](#)[java.lang.Object.hashCode](#)[java.lang.Object.notify](#)[java.lang.Object.notifyAll](#)[java.lang.Object.toString \(override\)](#)[java.lang.Object.wait](#)

## Restrictions

This class is available only in the JNI preferred implementation. It is not available in the JNI backward compatibility implementation, nor in the pure Java implementation.

## Related Classes

[TibrvEvent](#)[TibrvListener](#)[TibrvVectorCallback](#)

# TibrvVectorListener()

*Method*

## Declaration

```
TibrvVectorListener (  
    TibrvQueue queue,  
    TibrvVectorCallback callback,  
    TibrvTransport transport,  
    java.lang.String subject,  
    java.lang.Object closure)  
throws TibrvException
```

## Purpose

Create a vector listener object to listen for inbound messages, and receive them in a vector.

Parameter	Description
queue	Place each inbound message on this event queue.
callback	On dispatch, process the message vector with this callback interface implementation.
transport	Listen for inbound messages on this transport.
subject	Listen for inbound messages with subjects that match this specification. Wildcard subjects are permitted. The empty string is <i>not</i> a legal subject name.
closure	Store this closure data in the event object.

## Motivation

The standard way of receiving messages—one at a time—has the advantage of simplicity. However, if your application requires high throughput and low latency, consider receiving



data messages in a vector instead. Vector listeners can boost performance for programs that receive a large number of messages by reducing the overhead associated with message dispatch. Applications that require high throughput (that is, many messages arriving rapidly) could benefit from vector listeners.

**Warning**

We do *not* recommend vector listeners for command messages, administrative messages, advisory messages, nor any other out-of-band purpose.

## Activation and Dispatch

This method creates a vector listener event object, and *activates* the event—that is, it begins listening for all inbound messages with matching subjects. Dispatch removes a group of matching messages from the queue, and runs the callback method to process the message vector.

To stop receiving inbound messages on the subject, destroy the event object; this action cancels all messages already queued for the vector listener event.

## Interoperability

Vector listeners and ordinary listeners can listen on the same queue.

## Grouping Messages into Vectors

When several vector listeners use the same queue, the dispatcher groups messages into vectors with the following properties:

- The sequence of messages in a vector reflect consecutive arrival in the queue.
- All messages in a vector share the same callback object (though they need not match the same listener).

From these properties we can derive further inferences:

- If two vector listeners use the same callback object, then the dispatcher can group messages on their subjects into the same vector.
- If two messages are adjacent in the queue, but require different callback objects, then the dispatcher cannot group them into the same vector.

## Vector Listeners: Same Callback

Two vector listeners, F and P, listen on subjects FOO and PHU, respectively. Both F and P designate the same queue, Q1, and the same callback object, C1, to process their messages. In this situation, the dispatcher for Q1 can group messages on subjects FOO and PHU into the same vector (as long as the messages constitute a contiguous sequence within Q1).

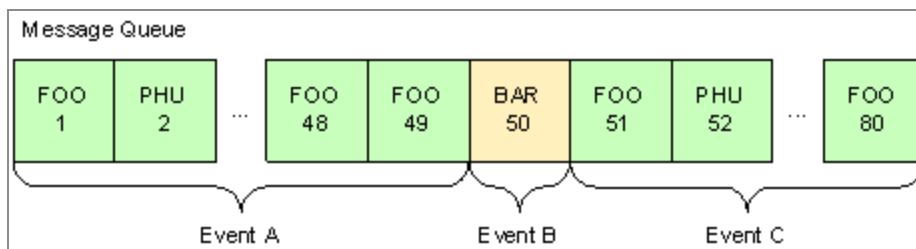
## Vector Listeners: Different Callbacks

Extend the previous example by adding a third vector listener, B, which listens on subject BAR. B designates the same queue, Q1, but uses a new callback object, C2 to process its messages. In this situation, the dispatcher for Q1 must group messages on subject BAR separately from messages on subjects FOO and PHU.

Suppose the Q1 contains 49 messages with subjects FOO or PHU, then 1 message with subject BAR, then 30 more messages with subjects FOO and PHU. [Grouping Messages into Vectors](#) shows this message queue. The dispatcher produces at least three separate events.

Because messages 49 and 50 require different callbacks, the dispatcher must close the vector of FOO and PHU messages at message 49, and start a new vector for message 50 with subject BAR. When the dispatcher encounters message 51 with subject FOO again, it closes the BAR vector after only one message, and starts a third vector for FOO.

*Figure 5: Grouping Messages into Vectors*



## Vector Listeners: Mixing Vector and Ordinary Listeners

Altering the previous example, suppose that B is an ordinary listener, instead of a vector listener. B necessarily specifies a different callback object than F and P (because ordinary listeners and vector listeners require different callback types with different signatures).

The behavior of the dispatcher remains the same as in [Vector Listeners: Different Callbacks](#).

## Dispatch Order vs. Processing Order

Messages dispatch in the order that they arrive in the queue. However, the order in which callbacks process messages can differ from dispatch order. The following examples illustrate this possibility by contrasting three scenarios.

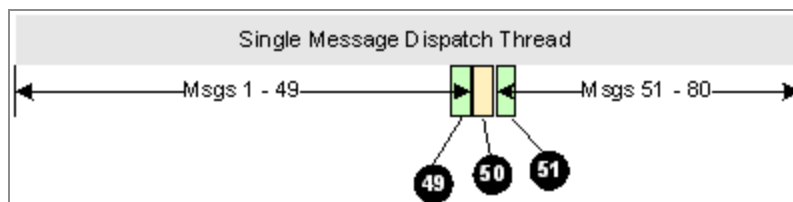
### Vector Listeners: Deliberately Processing Out of Order

The simplest callback (from the programmer's perspective) processes the messages within a vector in order (that is, the order that dispatcher moves them from the queue into the vector, which mirrors the order in which the messages arrive in the queue). Nonetheless you could program a callback that processes messages in reverse order, or any other order (though one would need a convincing reason to do so).

### Vector Listeners: Processing Message Vectors in a Single Dispatcher Thread

[Vector Listener Callbacks in a Single Dispatch Thread](#) shows a closer look at the situation of [Vector Listeners: Different Callbacks](#), in which several vector listeners all designate Q1 for their events. If a single thread dispatches Q1, then the callbacks are guaranteed to run in sequence. If the callbacks process messages in the order that they appear within the vectors, then message processing order is identical to dispatch order, which is also identical to arrival order. [Vector Listener Callbacks in a Single Dispatch Thread](#) shows this effect.

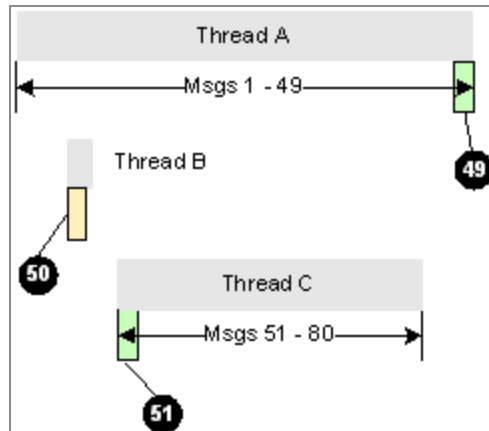
Figure 6: Vector Listener Callbacks in a Single Dispatch Thread



### Vector Listeners: Processing Message Vectors in Separate Threads

However, if several threads dispatch Q1 in parallel, then the callbacks can run concurrently. In this situation, message processing order could differ dramatically from arrival order.

[Vector Listener Callbacks in Multiple Dispatch Threads](#) shows this possibility.

*Figure 7: Vector Listener Callbacks in Multiple Dispatch Threads*

Although message number 49 dispatches (in event A) before message 50 (in event B), it is possible for the BAR callback (in thread B) to process message 50 before the FOO callback (in thread A) processes message 49. Furthermore, it is even possible for the FOO callback (in thread C) to process message 51 before the FOO callback (in thread A) processes message 49.



Before developing a program that processes inbound message vectors in several threads, consider carefully whether it is important (in the context of your application's semantics) to process messages in order of arrival.

## Restrictions

This method is available only in the JNI preferred implementation. It is not available in the JNI backward compatibility implementation, nor in the pure Java implementation.

## See Also

[TibrvEvent.destroy\(\)](#)

[TibrvVectorCallback](#)

[TibrvVectorCallback.onMsgs\(\)](#)

[TibrvVectorListener.getSubject\(\)](#)

# TibrvVectorListener.getSubject()

*Method*

## Declaration

```
java.lang.String getSubject();
```

## Purpose

Extract the subject from a vector listener event object.

## Restrictions

This method is available only in the JNI preferred implementation. It is not available in the JNI backward compatibility implementation, nor in the pure Java implementation.

# TibrvVectorListener.getTransport()

*Method*

## Declaration

```
TibrvTransport getTransport();
```

## Purpose

Extract the transport from a vector listener event object.

## Restrictions

This method is available only in the JNI preferred implementation. It is not available in the JNI backward compatibility implementation, nor in the pure Java implementation.

# TibrvVectorCallback

*Class*

## Declaration

```
abstract class com.tibco.tibrv.TibrvVectorCallback
```

## Purpose

Process inbound message vectors (vector listener events).

## Remarks

Implement a subclass to process inbound message vectors.

Method	Description
<a href="#">TibrvVectorCallback.onMsgs()</a>	Process inbound message vectors (vector listener events).

## Related Classes

[TibrvVectorListener](#)

## Restrictions

This abstract class is available only in the JNI preferred implementation. It is not available in the JNI backward compatibility implementation, nor in the pure Java implementation.

## See Also

[TibrvVectorListener\(\)](#)

# TibrvVectorCallback.onMsgs()

*Method*

## Declaration

```
void onMsgs(  
    TibrvMsg messages[])
```

## Purpose

Process inbound message vectors (vector listener events).

Parameter	Description
messages	This parameter receives an array of inbound messages.

## Remarks

Implement this method to process inbound message vectors.

In the simplest arrangement, your callback method processes the messages in the array. When the callback method returns, the Rendezvous library deallocates the array.

If your application requires a more complex processing arrangement, it can detach individual messages, and pass them to other threads for processing. To detach a message, use the copy constructor to make an independent copy of it. In the JNI implementation, you must detach a copy in order to use the message outside the scope of the callback. (If your program detaches a message, then you must also ensure that no references to the copy remain; such references could interfere with garbage collection. Furthermore, when using the TIBCO Rendezvous JNI preferred library, we recommend that programs call [TibrvMsg.dispose\(\)](#) to explicitly release the copy's storage within the native C environment.)

It is illegal to pass the message array to a different thread for processing, or to use it as dynamically-allocated storage.

Notice that in contrast to [TibrvMsgCallback.onMsg\(\)](#), this vector callback does not receive the listener event as an argument. You can use [TibrvMsg.getEvent\(\)](#) to get it from the individual message objects.



## Restrictions

This interface method is available only in the JNI preferred implementation. It is not available in the JNI backward compatibility implementation, nor in the pure Java implementation.

## See Also

[TibrvMsg\(\)](#)

[TibrvMsg.dispose\(\)](#)

[TibrvMsg.getEvent\(\)](#)

[TibrvVectorListener\(\)](#)

# TibrvTimer

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvTimer  
    extends TibrvEvent
```

## Purpose

Timer event.

## Remarks

All timers are repeating timers. To simulate a once-only timer, code the callback method to destroy the timer.

Programs must destroy instances of this class. Rendezvous software keeps internal references to these objects, so the Java garbage collector does not delete them automatically.

Destroying the queue of a timer automatically destroys the timer as well.

## Activation and Dispatch

The constructor creates a timer event object, and *activates* the timer event—that is, it requests notification from the operating system when the timer’s interval elapses. When the interval elapses, Rendezvous software places the event object on its event queue. Dispatch removes the event object from the queue, and runs the callback method to process the timer event. When the callback method begins, Rendezvous software automatically reactivates the event, using the same interval. On dispatch Rendezvous software also determines whether the next interval has already elapsed, and requeues the timer event if appropriate. (To stop the cycle, destroy the event object; see [TibrvEvent.destroy\(\)](#).)

Notice that time waiting in the event queue until dispatch can increase the effective interval of the timer. It is the programmer’s responsibility to ensure timely dispatch of events.

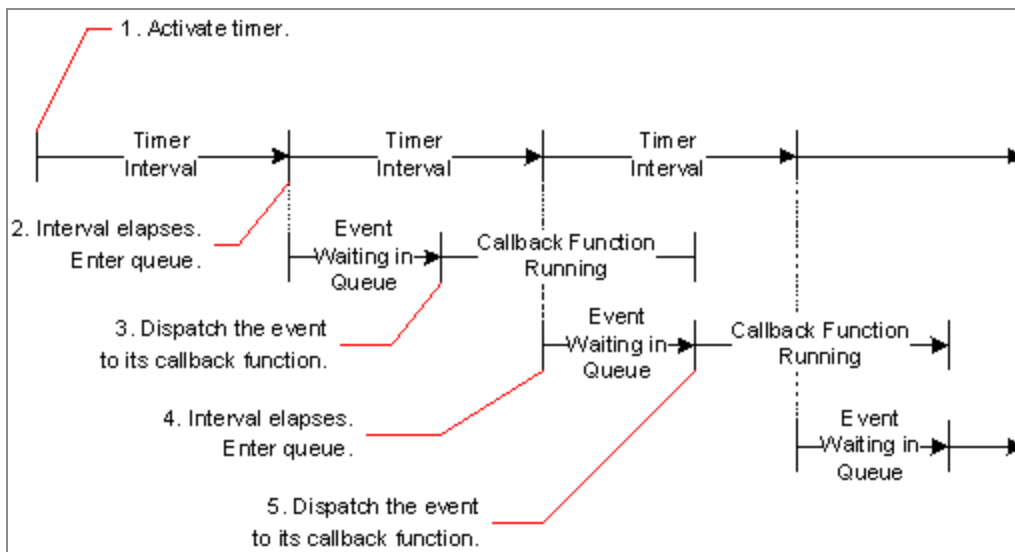
**Timer Activation and Dispatch** illustrates a sequence of timer intervals. The number of elapsed timer intervals directly determines the number of event callbacks.

At any moment the timer object appears on the event queue at most once—not several times as multiple copies. Nonetheless, Rendezvous software arranges for the appropriate number of timer event callbacks based the number of intervals that have elapsed since the timer became active or reset its interval.

Destroying or invalidating the timer object *immediately* halts the sequence of timer events. The timer object ceases to queue new events, and an event already in the queue does not result in a callback. (However, callback methods that are already running in other threads continue to completion.)

Resetting the timer interval *immediately* interrupts the sequence of timer events and begins a new sequence, counting the new interval from that moment. The reset operation is equivalent to destroying the timer and creating a new object in its place.

Figure 8: Timer Activation and Dispatch



## Timer Granularity

Express the timer interval (in seconds) as a 64-bit floating point number. This representation allows microsecond granularity for intervals for over 100 years. The actual granularity of intervals depends on hardware, Java and operating system constraints. Most releases of the JVM limit timer granularity to 10 milliseconds.

## Zero as Interval

Many programmers traditionally implement user events as timers with interval zero. Instead, we recommend implementing user events as messages on the intra-process transport. For more information, see [Intra-Process Transport and User Events in TIBCO Rendezvous Concepts](#).

Method	Description
<a href="#">TibrvTimer()</a>	Start a timer.
<a href="#">TibrvTimer.getInterval()</a>	Extract the interval from a timer event object.
<a href="#">TibrvTimer.resetInterval()</a>	Reset the interval of a timer event object.

### Inherited Methods

[TibrvEvent.destroy\(\)](#)

[TibrvEvent.getClosure\(\)](#)

[TibrvEvent.getQueue\(\)](#)

[TibrvEvent.isValid\(\)](#)

[java.lang.Object.equals](#)

[java.lang.Object.getClass](#)

[java.lang.Object.hashCode](#)

[java.lang.Object.notify](#)

[java.lang.Object.notifyAll](#)

[java.lang.Object.toString \(override\)](#)

[java.lang.Object.wait](#)

## Descendants

[TibrvEvent](#)

# TibrvTimer()

*Constructor*

## Declaration

```
TibrvTimer(  
    TibrvQueue    queue,  
    TibrvTimerCallback callback,  
    double        interval,  
    java.lang.Object closure)  
throws TibrvException
```

## Purpose

Start a timer.

## Remarks

All timers are repeating timers. To simulate a once-only timer, code the callback method to destroy the timer.

Parameter	Description
queue	At each time interval, place the event on this event queue.
callback	On dispatch, process the event with this interface implementation.
interval	The timer triggers its callback method at this repeating interval (in seconds).
closure	Store this closure data in the event object.

## Timer Granularity

Express the timer interval (in seconds) as a double (64-bit floating point number). This representation allows microsecond granularity for intervals for over 100 years. The actual granularity of intervals depends on Java implementation constraints, as well as hardware and operating system constraints.

The JNI implementation supports fine-resolution timers through the underlying C layer. Most releases of the JVM limit the Java implementation to 10 millisecond resolution.

## See Also

[TibrvEvent.destroy\(\)](#)

[TibrvTimerCallback](#)

[TibrvTimerCallback.onTimer\(\)](#)

# TibrvTimer.getInterval()

*Method*

## Declaration

```
double getInterval()
```

## Purpose

Extract the interval from a timer event object.

# TibrvTimer.resetInterval()

*Method*

## Declaration

```
void resetInterval(double newInterval)  
throws TibrvException
```

## Purpose

Reset the interval of a timer event object.

## Remarks

The timer begins counting the new interval immediately.

Parameter	Description
newInterval	The timer triggers its callback method at this new repeating interval (in seconds).

## Timer Granularity

Express the timer interval (in seconds) as a 64-bit floating point number. This representation allows microsecond granularity for intervals up to approximately 146 years. The actual granularity of intervals depends on hardware and operating system constraints.

## Limit of Effectiveness

This method can affect a timer only before or during its interval—but not after its interval has elapsed.

This method neither examines, changes nor removes an event that is already waiting in a queue for dispatch. If the next event for the timer object is already in the queue, then that event remains in the queue, representing the old interval. The change takes effect with the



subsequent interval. (To circumvent this limitation, a program can destroy the old timer object and replace it with a new one.)

# TibrvTimerCallback

*Interface*

## Declaration

```
interface com.tibco.tibrv.TibrvTimerCallback
```

## Purpose

Process timer events.

## Remarks

Implement this interface to process timer events.

Method	Description
<a href="#">TibrvTimerCallback.onTimer()</a>	<a href="#">Process timer events.</a>

## See Also

[TibrvTimer\(\)](#)

# TibrvTimerCallback.onTimer()

*Method*

## Declaration

```
void onTimer(TibrvTimer timer)
```

## Purpose

Process timer events.

## Remarks

Implement this method to process timer events.

Parameter	Description
timer	This parameter receives the timer event.

# TibrvDispatchable

*Interface*

## Declaration

```
interface com.tibco.tibrv.TibrvDispatchable
```

## Purpose

Common interface for queues and queue groups.

## Remarks

Both [TibrvQueue](#) and [TibrvQueueGroup](#) implement this interface, so programs can call the common methods on objects of either class. For example, consider a dispatcher routine that receives an object of type [TibrvDispatchable](#); it can call the `dispatch()` method, without needing to determine whether the object is queue or a queue group.

Method	Description
<a href="#">TibrvDispatchable.dispatch()</a>	Dispatch an event; if no event is ready, block.
<a href="#">TibrvDispatchable.poll()</a>	Dispatch an event, if possible.
<a href="#">TibrvDispatchable.timedDispatch()</a>	Dispatch an event, but if no event is ready to dispatch, limit the time that this call blocks while waiting for an event.

## See Also

[Interrupting Event Dispatch Threads](#)

[TibrvQueue](#)

[TibrvQueueGroup](#)

# TibrvDispatchable.dispatch()

*Method*

## Declaration

```
void dispatch()  
    throws TibrvException, java.lang.InterruptedException
```

## Purpose

Dispatch an event; if no event is ready, block.

## Remarks

If an event is ready to dispatch, then this call dispatches it, and then returns. If no events are waiting, then this call blocks indefinitely while waiting for the object to receive an event.

Both [TibrvQueue](#) and [TibrvQueueGroup](#) implement this method.

## See Also

[Interrupting Event Dispatch Threads](#)

[TibrvDispatchable](#)

[TibrvQueue.dispatch\(\)](#)

[Interrupting a Dispatch Call](#)

[TibrvQueueGroup.dispatch\(\)](#)

# TibrvDispatchable.poll()

*Method*

## Declaration

```
boolean poll()  
throws TibrvException, java.lang.InterruptedException
```

## Purpose

Dispatch an event, if possible.

## Remarks

If an event is ready to dispatch, then this call dispatches it, and then returns. If no events are waiting, then this call returns immediately.

When the call dispatches an event, it returns true. When the call does not dispatch an event, it returns false.

This call is equivalent to `timedDispatch(0)`.

Both [TibrvQueue](#) and [TibrvQueueGroup](#) implement this method.

## See Also

[Interrupting Event Dispatch Threads](#)

[TibrvDispatchable](#)

[TibrvQueue.poll\(\)](#)

[TibrvQueueGroup.poll\(\)](#)

# TibrvDispatchable.timedDispatch()

*Method*

## Declaration

```
boolean timedDispatch(double timeout)  
throws TibrvException, java.lang.InterruptedException
```

## Purpose

Dispatch an event, but if no event is ready to dispatch, limit the time that this call blocks while waiting for an event.

## Remarks

If an event is ready to dispatch, then this call dispatches it, and then returns. If no events are waiting, this call waits for an event to arrive. If an event arrives before the waiting time elapses, then it dispatches the event and returns. If the waiting time elapses first, then the call returns without dispatching an event.

When the call dispatches an event, it returns true. When the call does not dispatch an event, it returns false.

Both [TibrvQueue](#) and [TibrvQueueGroup](#) implement this method.

Parameter	Description
timeout	Maximum time (in seconds) that this call can block while waiting for an event to arrive.  Zero indicates no blocking (immediate timeout).  -1 indicates no timeout.

## See Also

[Interrupting Event Dispatch Threads](#)

[TibrvDispatchable](#)

[TibrvQueue.timedDispatch\(\)](#)

[TibrvQueueGroup.timedDispatch\(\)](#)



# TibrvQueue

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvQueue  
    extends java.lang.Object  
    implements TibrvDispatchable
```

## Purpose

Event queue.

## Remarks

Each event is associated with a [TibrvQueue](#) object; when the event occurs, Rendezvous software places the event object in its queue. Programs dispatch queues to process events.

Programs must explicitly destroy instances of this class. Rendezvous software keeps internal references to these objects, so the Java garbage collector does not delete them automatically.

## Default Queue

The method [Tibrv.defaultQueue\(\)](#) returns a pre-defined queue. Programs that need only one event queue can use this default queue (instead of using [TibrvQueue\(\)](#) to create one). The default queue has priority 1, can hold an unlimited number of events, and never discards an event (since it never exceeds an event limit).

Rendezvous software places all advisories pertaining to queue overflow on the default queue.

Programs cannot destroy the default queue, except as a side effect of [Tibrv.close\(\)](#). Programs cannot change the parameters of the default queue.

## Limit Policy

These constants specify the possible strategies for resolving overflow of queue limit.

Constant	Description
<code>TibrvQueue.DISCARD_NONE</code>	Never discard events; use this policy when a queue has no limit on then number of events it can contain.
<code>TibrvQueue.DISCARD_FIRST</code>	Discard the first event in the queue (that is, the oldest event in the queue, which would otherwise be the next event to dispatch).
<code>TibrvQueue.DISCARD_LAST</code>	Discard the last event in the queue (that is, the youngest event in the queue).
<code>TibrvQueue.DISCARD_NEW</code>	Discard the new event (which would otherwise cause the queue to overflow its maximum events limit).

Method	Description
--------	-------------

## Life Cycle

<code>TibrvQueue()</code>	Create an event queue.
<code>TibrvQueue.destroy()</code>	Shared Library Files
<code>TibrvQueue.isDefault()</code>	Test whether a queue is the default queue.
<code>TibrvQueue.isValid()</code>	Test validity of a queue.

## Dispatch

<code>TibrvQueue.dispatch()</code>	Dispatch an event; if no event is ready, block.
<code>TibrvQueue.poll()</code>	Dispatch an event, if possible.
<code>TibrvQueue.timedDispatch()</code>	Dispatch an event, but if no event is ready to dispatch, limit the time that this call blocks while waiting for an event.

## Properties

Method	Description
<a href="#">TibrvQueue.getCount()</a>	Extract the number of events in a queue.
<a href="#">TibrvQueue.getDiscardAmount()</a>	Extract the discard amount of a queue.
<a href="#">TibrvQueue.getLimitPolicy()</a>	Extract the limit policy of a queue.
<a href="#">TibrvQueue.getMaxEvents()</a>	Extract the maximum event limit of a queue.
<a href="#">TibrvQueue.getName()</a>	Extract the name of a queue.
<a href="#">TibrvQueue.getPriority()</a>	Extract the priority of a queue.
<a href="#">TibrvQueue.setName()</a>	Set the name of a queue.
<a href="#">TibrvQueue.setLimitPolicy()</a>	Set the limit properties of a queue.
<a href="#">TibrvQueue.setPriority()</a>	Set the priority of a queue.

#### Inherited Methods

[java.lang.Object.equals](#)  
[java.lang.Object.getClass](#)  
[java.lang.Object.hashCode](#)  
[java.lang.Object.notify](#)  
[java.lang.Object.notifyAll](#)  
[java.lang.Object.toString \(override\)](#)  
[java.lang.Object.wait](#)

# TibrvQueue()

*Constructor*

## Declaration

```
TibrvQueue()
```

## Purpose

Create an event queue.

## Remarks

Upon creation, new queues use these default values.

Property	Default Value	Set Method
limitPolicy	<a href="#">TibrvQueue.DISCARD_NONE</a>	<a href="#">TibrvQueue.setLimitPolicy()</a>
maxEvents	zero (unlimited)	
discardAmount	zero	
name	tibrvQueue	<a href="#">TibrvQueue.setName()</a>
priority	1	<a href="#">TibrvQueue.setPriority()</a>

# TibrvQueue.destroy()

*Method*

## Declaration

```
void destroy()
```

## Purpose

Destroy an event queue.

## Remarks

When a queue is destroyed, events that remain in the queue are discarded.

Destroying a queue invalidates all events associated with the queue.

A program must not call [TibrvQueue.destroy\(\)](#) on the default queue. Closing [Tibrv](#) destroys the default queue; see [Tibrv.close\(\)](#).

# TibrvQueue.dispatch()

*Method*

## Declaration

```
void dispatch()  
    throws TibrvException, java.lang.InterruptedException
```

## Purpose

Dispatch an event; if no event is ready, block.

## Remarks

If the queue is not empty, then this call dispatches the event at the head of the queue, and then returns. If the queue is empty, then this call blocks indefinitely while waiting for the queue to receive an event.

## Interrupting a Dispatch Call

To interrupt an event dispatch thread in the Java implementation (see [Tibrv.open\(\)](#)), use the Java method `Thread.interrupt()`.

In the JNI (native) implementation, this call does not throw `InterruptedException`; instead, programs must explicitly check for interruptions by calling `Thread.interrupted()` before or after [TibrvQueue.dispatch\(\)](#).

The more reliable way to interrupt a JNI event dispatch thread is to destroy the [TibrvDispatchable](#) object that the thread dispatches. When the dispatch call encounters the invalid queue or queue group, it throws a [TibrvException](#) in the event dispatch thread.

## See Also

[Interrupting Event Dispatch Threads](#)

[TibrvDispatchable](#)

[TibrvDispatchable.dispatch\(\)](#)

[TibrvQueue.poll\(\)](#)

[TibrvQueue.timedDispatch\(\)](#)

[TibrvDispatcher](#)

# TibrvQueue.getCount()

*Method*

## Declaration

```
int getCount()  
    throws TibrvException
```

## Purpose

Extract the number of events in a queue.



# TibrvQueue.getDiscardAmount()

*Method*

## Declaration

```
int getDiscardAmount()  
    throws TibrvException
```

## Purpose

Extract the discard amount of a queue.

## Remarks

When the queue exceeds its maximum event limit, discard a block of events. This property specifies the number of events to discard.

## See Also

[TibrvQueue.setLimitPolicy\(\)](#)

# TibrvQueue.getLimitPolicy()

*Method*

## Declaration

```
int getLimitPolicy()  
    throws TibrvException
```

## Purpose

Extract the limit policy of a queue.

## Remarks

Each queue has a policy for discarding events when a new event would cause the queue to exceed its maxEvents limit. For an explanation of the policy values, see [Limit Policy](#).

## See Also

[TibrvQueue.setLimitPolicy\(\)](#)

# TibrvQueue.getMaxEvents()

*Method*

## Declaration

```
int getMaxEvents()  
    throws TibrvException
```

## Purpose

Extract the maximum event limit of a queue.

## Remarks

Programs can limit the number of events that a queue can hold—either to curb queue growth, or implement a specialized dispatch semantics.

Zero specifies an unlimited number of events.

## See Also

[TibrvQueue.setLimitPolicy\(\)](#)

# TibrvQueue.getName()

*Method*

## Declaration

```
java.lang.String getName()
```

## Purpose

Extract the name of a queue.

## Remarks

Queue names assist programmers and administrators in troubleshooting queues. When Rendezvous software delivers an advisory message pertaining to a queue, it includes the queue's name; administrators can use queue names to identify specific queues within a program.

The default name of every queue is `tibrvQueue`. We strongly recommend that you relabel each queue with a distinct and informative name, for use in debugging.

This method returns the queue's name, even when the queue is invalid.

## See Also

[TibrvQueue.setName\(\)](#)

# TibrvQueue.getPriority()

*Method*

## Declaration

```
int getPriority()  
    throws TibrvException
```

## Purpose

Extract the priority of a queue.

## Remarks

Each queue has a single priority value, which controls its dispatch precedence within queue groups. Higher values dispatch before lower values; queues with equal priority values dispatch in round-robin fashion.

When the queue is invalid, this method throws an exception.

## See Also

[TibrvQueue.setPriority\(\)](#)

# TibrvQueue.isDefault()

*Method*

## Declaration

```
final boolean isDefault()
```

## Purpose

Test whether a queue is the default queue.

## Remarks

Returns true if the queue is the default queue; false otherwise.

## See Also

[Tibrv.defaultQueue\(\)](#)

# TibrvQueue.isValid()

*Method*

## Declaration

```
final boolean isValid()
```

## Purpose

Test validity of a queue.

## Remarks

Returns true if the queue is valid; false if the queue has been destroyed.

## See Also

[Tibrv.close\(\)](#)

[TibrvQueue.destroy\(\)](#)

# TibrvQueue.poll()

*Method*

## Declaration

```
boolean poll()  
throws TibrvException, java.lang.InterruptedException
```

## Purpose

Dispatch an event, if possible.

## Remarks

If the queue is not empty, then this call dispatches the event at the head of the queue, and then returns. If the queue is empty, then this call returns immediately.

When the call dispatches an event, it returns true. When the call does not dispatch an event, it returns false.

This call is equivalent to `timedDispatch(0)`.

## See Also

[Interrupting Event Dispatch Threads](#)

[TibrvDispatchable](#)

[TibrvDispatchable.poll\(\)](#)

[TibrvQueue.dispatch\(\)](#)

[TibrvQueue.timedDispatch\(\)](#)



# TibrvQueue.setName()

*Method*

## Declaration

```
void setName(java.lang.String queueName)  
throws TibrvException
```

## Purpose

Set the name of a queue.

## Remarks

Queue names assist programmers and administrators in troubleshooting queues. When Rendezvous software delivers an advisory message pertaining to a queue, it includes the queue's name; administrators can use queue names to identify specific queues within a program.

The default name of every queue is `tibrvQueue`. We strongly recommend that you relabel each queue with a distinct and informative name, for use in debugging.

Parameter	Description
queueName	Replace the name of the queue with this new name.  It is illegal to supply null as the new queue name.

## See Also

[TibrvQueue.getName\(\)](#)

# TibrvQueue.setLimitPolicy()

*Method*

## Declaration

```
void setLimitPolicy(  
    int limitPolicy,  
    int maxEvents,  
    int discardAmount)  
throws TibrvException
```

## Purpose

Set the limit properties of a queue.

## Remarks

This method simultaneously sets three related properties, which together describe the behavior of a queue in overflow situations. Each call must explicitly specify all three properties.

Parameter	Description
limitPolicy	<p>Each queue has a policy for discarding events when a new event would cause the queue to exceed its maxEvents limit. Choose from the values of <a href="#">Limit Policy</a>.</p> <p>When maxEvents is zero (unlimited), the policy must be <a href="#">TibrvQueue.DISCARD_NONE</a>.</p>
maxEvents	<p>Programs can limit the number of events that a queue can hold—either to curb queue growth, or implement a specialized dispatch semantics.</p> <p>Zero specifies an unlimited number of events; in this case, the policy must be <a href="#">TibrvQueue.DISCARD_NONE</a>.</p>

Parameter	Description
discardAmount	<p>When the queue exceeds its maximum event limit, discard a block of events. This property specifies the number of events to discard.</p> <p>When discardAmount is zero, the policy must be <a href="#">TibrvQueue.DISCARD_NONE</a>.</p>

## See Also

[TibrvQueue.getDiscardAmount\(\)](#)

[TibrvQueue.getLimitPolicy\(\)](#)

[TibrvQueue.getMaxEvents\(\)](#)

# TibrvQueue.setPriority()

*Method*

## Declaration

```
void setPriority(int priority)  
throws TibrvException
```

## Purpose

Set the priority of a queue.

## Remarks

Each queue has a single priority value, which controls its dispatch precedence within queue groups. Higher values dispatch before lower values; queues with equal priority values dispatch in round-robin fashion.

Changing the priority of a queue affects its position in all the queue groups that contain it.

Parameter	Description
priority	Replace the priority of the queue with this new value.  The priority must be a non-negative integer. Priority zero signifies the last queue to dispatch.

## See Also

[TibrvQueue.getPriority\(\)](#)

# TibrvQueue.timedDispatch()

*Method*

## Declaration

```
boolean timedDispatch(double timeout)  
throws TibrvException, java.lang.InterruptedException
```

## Purpose

Dispatch an event, but if no event is ready to dispatch, limit the time that this call blocks while waiting for an event.

## Remarks

If an event is already in the queue, this call dispatches it, and returns immediately. If the queue is empty, this call waits for an event to arrive. If an event arrives before the waiting time elapses, then it dispatches the event and returns. If the waiting time elapses first, then the call returns without dispatching an event.

When the call dispatches an event, it returns true. When the call does not dispatch an event, it returns false.

Parameter	Description
timeout	Maximum time (in seconds) that this call can block while waiting for an event to arrive in the queue.  Zero indicates no blocking (immediate timeout).  -1 indicates no timeout.

## See Also

[Interrupting Event Dispatch Threads](#)

[TibrvDispatchable](#)

[TibrvDispatchable.timedDispatch\(\)](#)

[TibrvQueue.dispatch\(\)](#)

[TibrvQueue.poll\(\)](#)

# TibrvQueueGroup

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvQueueGroup  
    extends java.lang.Object  
    implements TibrvDispatchable
```

## Purpose

Prioritized dispatch of several queues with one call.

## Remarks

Queue groups add flexibility and fine-grained control to the event queue dispatch mechanism. Programs can create groups of queues and dispatch them according to their queue priorities.

Programs must explicitly destroy instances of this class. Rendezvous software keeps internal references to these objects, so the Java garbage collector does not delete them automatically.

Method	Description
--------	-------------

### Life Cycle

<a href="#">TibrvQueueGroup()</a>	Create an event queue group.
<a href="#">TibrvQueueGroup.destroy()</a>	Destroy an event queue group.
<a href="#">TibrvQueueGroup.isValid()</a>	Test validity of a queue group.

### Dispatch

Method	Description
<a href="#">TibrvQueueGroup.dispatch()</a>	Dispatch an event from a queue group; if no event is ready, block.
<a href="#">TibrvQueueGroup.poll()</a>	Dispatch an event, but if no event is ready to dispatch, return immediately (without blocking).
<a href="#">TibrvQueueGroup.timedDispatch()</a>	Dispatch an event, but if no event is ready to dispatch, limit the time that this call blocks while waiting for an event.

## Queues

<a href="#">TibrvQueueGroup.add()</a>	Add an event queue to a queue group.
<a href="#">TibrvQueueGroup.contains()</a>	Test whether a queue is in a queue group.
<a href="#">TibrvQueueGroup.elements()</a>	Extract an enumeration of the queues in a queue group.
<a href="#">TibrvQueueGroup.getCount()</a>	Extract the number of queues in a queue group.
<a href="#">TibrvQueueGroup.remove()</a>	Remove an event queue from a queue group.

## Inherited Methods

[java.lang.Object.equals](#)  
[java.lang.Object.getClass](#)  
[java.lang.Object.hashCode](#)  
[java.lang.Object.notify](#)  
[java.lang.Object.notifyAll](#)  
[java.lang.Object.toString \(override\)](#)  
[java.lang.Object.wait](#)



# TibrvQueueGroup()

*Constructor*

## Declaration

```
TibrvQueueGroup()  
throws TibrvException
```

## Purpose

Create an event queue group.

## Remarks

The new queue group is empty.

The queue group remains valid until the program explicitly destroys it.

## See Also

[TibrvQueueGroup.add\(\)](#)

[TibrvQueueGroup.destroy\(\)](#)

# TibrvQueueGroup.add()

*Method*

## Declaration

```
void add(TibrvQueue eventQueue)  
    throws TibrvException
```

## Purpose

Add an event queue to a queue group.

## Remarks

If the queue is already in the group, adding it again has no effect.

If either the queue or the group is invalid, this method throws a [TibrvException](#).

Parameter	Description
eventQueue	Add this event queue to a queue group.

## See Also

[TibrvQueue](#)

# TibrvQueueGroup.contains()

*Method*

## Declaration

```
boolean contains(TibrvQueue eventQueue)  
throws TibrvException
```

## Purpose

Test whether a queue is in a queue group.

## Remarks

If the queue is in the group, return true; otherwise false.

If the group is invalid, return false.

Parameter	Description
eventQueue	Test the membership of this event queue in the queue group.

## See Also

[TibrvQueue](#)

# TibrvQueueGroup.destroy()

*Method*

## Declaration

```
void destroy()
```

## Purpose

Destroy an event queue group.

## Remarks

The individual queues in the group continue to exist, even though the group has been destroyed.

## See Also

[TibrvQueueGroup\(\)](#)

# TibrvQueueGroup.dispatch()

*Method*

## Declaration

```
void dispatch()  
throws TibrvException, java.lang.InterruptedException
```

## Purpose

Dispatch an event from a queue group; if no event is ready, block.

## Remarks

If any queue in the group contains an event, then this call searches the queues in priority order, dispatches an event from the first non-empty queue that it finds, and then returns. If all the queues are empty, then this call blocks indefinitely while waiting for any queue in the group to receive an event.

When searching the group for a non-empty queue, this call searches according to the priority values of the queues. If two or more queues have identical priorities, subsequent dispatch and poll calls rotate through them in round-robin fashion.

## See Also

[Interrupting Event Dispatch Threads](#)

[TibrvDispatchable](#)

[TibrvDispatchable.dispatch\(\)](#)

[Interrupting a Dispatch Call](#)

[TibrvQueueGroup.timedDispatch\(\)](#)

[TibrvQueueGroup.poll\(\)](#)

# TibrvQueueGroup.elements()

*Method*

## Declaration

```
java.util Enumeration elements()
```

## Purpose

Extract an enumeration of the queues in a queue group.

## Remarks

If the group is invalid, return null.

# TibrvQueueGroup.getCount()

*Method*

## Declaration

```
int getCount()
```

## Purpose

Extract the number of queues in a queue group.

## Remarks

If the group is invalid, return zero.

# TibrvQueueGroup.isValid()

*Method*

## Declaration

```
boolean isValid()
```

## Purpose

Test validity of a queue group.

## Remarks

Returns true if the queue group is valid; false if the queue group has been destroyed.

## See Also

[Tibrv.close\(\)](#)

[TibrvQueueGroup.destroy\(\)](#)



# TibrvQueueGroup.poll()

*Method*

## Declaration

```
boolean poll()  
throws TibrvException, java.lang.InterruptedException
```

## Purpose

Dispatch an event, but if no event is ready to dispatch, return immediately (without blocking).

## Remarks

If any queue in the group contains an event, then this call searches the queues in priority order, dispatches an event from the first non-empty queue that it finds, and then returns. If all the queues are empty, then this call returns immediately.

When searching the group for a non-empty queue, this call searches according to the priority values of the queues. If two or more queues have identical priorities, subsequent dispatch and poll calls rotate through them in round-robin fashion.

When the call dispatches an event, it returns true. When the call does not dispatch an event, it returns false.

This call is equivalent to `timedDispatch(0)`.

## See Also

[Interrupting Event Dispatch Threads](#)

[TibrvDispatchable](#)

[TibrvDispatchable.poll\(\)](#)

[TibrvQueueGroup.dispatch\(\)](#)

[TibrvQueueGroup.timedDispatch\(\)](#)

# TibrvQueueGroup.remove()

*Method*

## Declaration

```
void remove(TibrvQueue eventQueue)  
throws TibrvException
```

## Purpose

Remove an event queue from a queue group.

## Remarks

If the queue is not in the group, or if the group is invalid, this call throws an exception with the status code [TibrvStatus.INVALID\\_QUEUE](#).

Parameter	Description
eventQueue	Remove this event queue from a queue group.

## See Also

[TibrvQueue](#)

# TibrvQueueGroup.timedDispatch()

*Method*

## Declaration

```
boolean timedDispatch(double timeout)  
throws TibrvException, java.lang.InterruptedException
```

## Purpose

Dispatch an event, but if no event is ready to dispatch, limit the time that this call blocks while waiting for an event.

## Remarks

If any queue in the group contains an event, then this call searches the queues in priority order, dispatches an event from the first non-empty queue that it finds, and then returns. If the queue is empty, this call waits for an event to arrive in any queue. If an event arrives before the waiting time elapses, then the call searches the queues, dispatches the event, and returns. If the waiting time elapses first, then the call returns without dispatching an event.

When searching the group for a non-empty queue, this call searches according to the priority values of the queues. If two or more queues have identical priorities, subsequent dispatch calls rotate through them in round-robin fashion.

When the call dispatches an event, it returns true. When the call does not dispatch an event, it returns false.

Parameter	Description
timeout	Maximum time (in seconds) that this call can block while waiting for an event to arrive in the queue group.  Zero indicates no blocking (immediate timeout).  -1 indicates no timeout.

## See Also

[Interrupting Event Dispatch Threads](#)

[TibrvDispatchable](#)

[TibrvDispatchable.timedDispatch\(\)](#)

[TibrvQueueGroup.dispatch\(\)](#)

[TibrvQueueGroup.poll\(\)](#)

# TibrvDispatcher

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvDispatcher  
    extends java.lang.Thread
```

## Purpose

Dispatch events from a queue or queue group.

## Remarks

Upon creation, this thread class loops indefinitely, repeatedly dispatching a queue or queue group.

This class is a programming convenience. Programs can implement specialized dispatcher threads, and use them instead of this class.

## Exceptions

If the thread catches a [TibrvException](#), it presents the [DISPATCHER.THREAD\\_EXITED](#) advisory on the process transport, and exits. (The program destroyed the queue or queue group object, so this thread can no longer dispatch it.)

If the thread catches an interruption ([java.lang.InterruptedException](#)), it presents the [DISPATCHER.THREAD\\_EXITED](#) advisory on the process transport, and exits.

## Interrupting a Dispatcher Thread

To interrupt a dispatcher thread in the Java implementation (see [Tibrv.open\(\)](#)), use either the Java method [Thread.interrupt\(\)](#) or [TibrvDispatcher.destroy\(\)](#).

In either the JNI (native) implementation or the Java implementation, you can interrupt a dispatcher thread by calling its [TibrvDispatcher.destroy\(\)](#) method. Destroying the thread insures prompt thread exit—at the latest, after the return of a program callback method (if one is in progress).

Method	Description
<a href="#">TibrvDispatcher()</a>	Create a dispatcher thread.

Constant	Description
<code>TibrvDispatcher.DEFAULT_NAME</code>	When the constructor does not receive a thread name, it gives the new dispatcher thread this default name.

### Inherited Methods

[java.lang.Thread.activeCount](#)  
[java.lang.Thread.checkAccess](#)  
[java.lang.Thread.countStackFrames](#)  
[java.lang.Thread.currentThread](#)  
[java.lang.Thread.destroy \(override\)](#)  
[java.lang.Thread.dumpStack](#)  
[java.lang.Thread.enumerate](#)  
[java.lang.Thread.getContextClassLoader](#)  
[java.lang.Thread.getName](#)  
[java.lang.Thread.getPriority](#)  
[java.lang.Thread.getThreadGroup](#)  
[java.lang.Thread.interrupt](#)  
[java.lang.Thread.interrupted](#)  
[java.lang.Thread.isAlive](#)  
[java.lang.Thread.isDaemon](#)  
[java.lang.Thread.isInterrupted](#)  
[java.lang.Thread.join](#)  
[java.lang.Thread.resume](#)  
[java.lang.Thread.run \(override\)](#)

## Inherited Methods

`java.lang.Thread.setContextClassLoader`

`java.lang.Thread.setDaemon`

`java.lang.Thread.setName`

`java.lang.Thread.setPriority`

`java.lang.Thread.sleep`

`java.lang.Thread.start`

`java.lang.Thread.stop`

`java.lang.Thread.suspend`

`java.lang.Thread.toString`

`java.lang.Thread.yield`

---

`java.lang.Object.equals`

`java.lang.Object.getClass`

`java.lang.Object.hashCode`

`java.lang.Object.notify`

`java.lang.Object.notifyAll`

`java.lang.Object.wait`

---

## See Also

[Tibrv.isNativeImpl\(\)](#)

[Tibrv.open\(\)](#)

[TibrvQueue.dispatch\(\)](#)

# TibrvDispatcher()

*Constructor*

## Declaration

```
TibrvDispatcher(  
    java.lang.ThreadGroup group,  
    java.lang.String      name,  
    TibrvDispatchable     dispatchable)  
TibrvDispatcher(  
    java.lang.String      name,  
    TibrvDispatchable     dispatchable)  
TibrvDispatcher(  
    TibrvDispatchable     dispatchable)  
TibrvDispatcher(  
    java.lang.ThreadGroup group,  
    java.lang.String      name,  
    TibrvDispatchable     dispatchable,  
    double                 timeout)  
TibrvDispatcher(  
    java.lang.String      name,  
    TibrvDispatchable     dispatchable,  
    double                 timeout)  
TibrvDispatcher(  
    TibrvDispatchable     dispatchable,  
    double                 timeout)
```

## Purpose

Create a dispatcher thread.

## Remarks

This constructor immediately starts the thread.

Parameter	Description
group	Create a dispatcher thread in this Java thread group.



Parameter	Description
	When absent, use the main thread group.
name	Create a dispatcher thread with this name. When absent, use the default thread name.
dispatchable	Create a thread that dispatches this <a href="#">TibrvQueue</a> or <a href="#">TibrvQueueGroup</a> .
timeout	When this time period (in seconds) elapses without dispatching an event, the thread exits. When absent, the default is to run indefinitely (with no timeout).

## See Also

[TibrvDispatcher](#)

[DISPATCHER.THREAD\\_EXITED](#) on [page 242](#) in TIBCO Rendezvous Concepts

# Transports

---

Transports manage network connections and send outbound messages.

This section presents the various transport classes and their methods.

## See Also

[TibrvCmTransport](#)

[TibrvCmQueueTransport](#)

# TibrvTransport

*Class*

## Declaration

```
abstract class com.tibco.tibrv.TibrvTransport  
    extends java.lang.Object
```

## Purpose

A transport object represents a delivery mechanism for messages.

## Remarks

A transport describes a carrier for messages—whether across a network, among processes on a single computer, or within a process. Transports manage network connections, and send outbound messages.

A transport also defines the delivery scope of a message—that is, the set of *possible* destinations for the messages it sends.

Destroying a transport object invalidates subsequent send calls on that transport, and invalidates any listeners using that transport.

Programs must explicitly destroy instances of these classes. Rendezvous software keeps internal references to these objects, so the Java garbage collector does not delete them automatically.

This abstract class is the superclass of all other transport classes. Methods defined by this class are implemented by all transport subclasses (except [TibrvCmQueueTransport](#), for which some methods do not apply).

## Intra-Process Transport

Each process has exactly one intra-process transport; the call [Tibrv.open\(\)](#) automatically creates it, and the call [Tibrv.processTransport\(\)](#) extracts it. Programs must not destroy the intra-process transport.

Method	Description
<a href="#">TibrvTransport.createInbox()</a>	Create a unique inbox subject name.
<a href="#">TibrvTransport.destroy()</a>	Destroy a transport.
<a href="#">TibrvTransport.isValid()</a>	Test validity of a transport.
<a href="#">TibrvTransport.getDescription()</a>	Extract the program description parameter from a transport.
<a href="#">TibrvTransport.requestReliability()</a>	Request reliability interval (message retention time) for a service.
<a href="#">TibrvTransport.send()</a>	Send a message.
<a href="#">TibrvTransport.sendReply()</a>	Send a reply message.
<a href="#">TibrvTransport.sendRequest()</a>	Send a request message and wait for a reply.
<a href="#">TibrvTransport.setBatchSize()</a>	Enable outbound batching of data from IPM, and set the batch size (in bytes).
<a href="#">TibrvTransport.setDescription()</a>	Set the program description parameter of a transport.

### Inherited Methods

[java.lang.Object.equals](#)  
[java.lang.Object.getClass](#)  
[java.lang.Object.hashCode](#)  
[java.lang.Object.notify](#)  
[java.lang.Object.notifyAll](#)  
[java.lang.Object.toString](#)  
[java.lang.Object.wait](#)

## Descendants

[TibrvProcessTransport](#)

[TibrvNetTransport](#)

[TibrvRvdTransport](#)

[TibrvCmTransport](#)

[TibrvCmQueueTransport](#)

## See Also

[Transport](#) in TIBCO Rendezvous Concepts

# TibrvTransport.createInbox()

*Method*

## Declaration

```
java.lang.String createInbox()  
throws TibrvException
```

## Purpose

Create a unique inbox subject name.

## Remarks

This method creates inbox names that are unique throughout the transport scope.

- For network transports, inbox subject names are unique across all processes within the local router domain—that is, anywhere that direct multicast contact is possible. The inbox name is not necessarily unique outside of the local router domain.
- For the intra-process transport, inbox names are unique across all threads of the process.

This method creates only the unique name for an inbox; it does not begin listening for messages on that subject name. To begin listening, pass the inbox name as the subject argument to [TibrvListener\(\)](#). The inbox name is only valid for use with the same transport that created it. When calling [TibrvListener\(\)](#), you *must* pass the same transport object that created the inbox subject name.

Remember that other programs have no information about an inbox subject name until the listening program uses it as a reply subject in an outbound message.

Use inbox subject names for delivery to a specific destination. In the context of a network transport, an inbox destination specifies unicast (point-to-point) delivery.

Rendezvous routing daemons (rvrd) translate inbox subject names that appear as the send subject or reply subject of a message. They do not translate inbox subject names within the data fields of a message.

This inherited method is disabled for [TibrvCmQueueTransport](#) objects.



### Warning

This method is the only legal way for programs to create inbox subject names.

## See Also

[TibrvMsg.setReplySubject\(\)](#)

# TibrvTransport.destroy()

*Method*

## Declaration

```
void destroy()
```

## Purpose

Destroy a transport.

## Remarks

Programs must explicitly destroy each transport object.

Destroying a transport achieves these effects:

- The transport flushes all outbound data to the Rendezvous daemon.  
This effect is especially important, and neither exiting the program nor calling [Tibrv.close\(\)](#) is sufficient to flush outbound data.
- The transport invalidates (but does not destroy) all associated events.
- Subsequent calls that use the destroyed transport throw the exception [TibrvStatus.INVALID\\_TRANSPORT](#).

It is illegal to destroy the intra-process transport (see [Tibrv.processTransport\(\)](#)).

## See Also

[TibrvTransport.isValid\(\)](#)



# TibrvTransport.isValid()

*Method*

## Declaration

```
boolean isValid()
```

## Purpose

Test validity of a transport.

## Remarks

Returns true if the transport is valid; false if the transport has been destroyed.

## See Also

[Tibrv.close\(\)](#)

[TibrvTransport.destroy\(\)](#)

[TibrvCmTransport.destroy\(\)](#)

[TibrvCmQueueTransport.destroy\(\)](#)

# TibrvTransport.getDescription()

*Method*

## Declaration

```
String getDescription()
```

## Purpose

Extract the program description parameter from a transport.

## Remarks

The description identifies your program to Rendezvous components. Browser administration interfaces display the description string.

## See Also

[TibrvTransport](#)

[TibrvTransport.setDescription\(\)](#)

# TibrvTransport.requestReliability()

*Method*

## Declaration

```
void requestReliability(double reliability)  
    throws TibrvException
```

## Purpose

Request reliability interval (message retention time) for a service.

Parameter	Description
reliability	Request this reliability interval (in seconds).  This value must be greater than zero.

## Remarks

This call lets application programs shorten the reliability interval of the specific service associated with a transport object. Successful calls change the daemon's reliability interval for all transports within the application process that use the same service.

Programs can request reliability only from daemons of release 8.2 or later.

An application can request a shorter retention time than the value that governs the daemon as a whole (either the factory default or the daemons -reliability parameter). The daemon's governing value silently overrides calls that request a longer retention time.

## Maximum Value Rule

Client transport objects that connect to the same daemon could specify different reliability intervals on the same service—whether by requesting a reliability value, or by using the daemon's effective value. In this situation, the daemon selects the *largest* potential value from among all the transports on that service, and uses that maximum value as the effective reliability interval for the service (that is, for all the transports on the service). This

method of resolution favors the more stringent reliability requirements. (Contrast this rule with the [Lower Value Rule](#) that applies between two daemons.)

## Recomputing the Reliability

Whenever a transport connects, requests reliability, or disconnects from the daemon, the daemon recalculates the reliability interval for the corresponding service, by selecting the largest value of all transports communicating on that service.

When recomputing the reliability interval would result in a shorter retention time, the daemon delays using the new value until after an interval equivalent to the older (longer) retention time. This delay ensures that the daemon retains message data at least as long as the effective reliability interval at the time the message is sent.

## See Also

[TibrvTransport](#)

[Reliability and Message Retention Time](#) in TIBCO Rendezvous Administration

[Lower Value Rule](#) in TIBCO Rendezvous Administration

[Changing the Reliability Interval within an Application Program](#) in TIBCO Rendezvous Administration

[Reliable Message Delivery](#) in TIBCO Rendezvous Concepts

# TibrvTransport.send()

*Method*

## Declaration

```
void send(TibrvMsg message)
    throws TibrvException
void send(TibrvMsg[] messages)
    throws TibrvException
```

## Purpose

Send a message.

## Remarks

The message must have a valid destination subject; see [TibrvMsg.setSendSubject\(\)](#).

Parameter	Description
message	Send this message.
messages	Send this array of messages with one call. In most applications this call is more efficient than a series of send calls on individual messages.

## Restrictions

Sending an array of messages is available only in the JNI preferred implementation. It is not available in the JNI backward compatibility implementation, nor in the pure Java implementation.

## See Also

[TibrvMsg.setSendSubject\(\)](#)

# TibrvTransport.sendReply()

*Method*

## Declaration

```
void sendReply(  
    TibrvMsg replyMsg,  
    TibrvMsg requestMsg)  
    throws TibrvException
```

## Purpose

Send a reply message.

## Remarks

This convenience call extracts the reply subject of an inbound request message, and sends an outbound reply message to that subject. In addition to the convenience, this call is marginally faster than using separate calls to extract the subject and send the reply.

This method overwrites any existing send subject of the reply message with the reply subject of the request message.

Parameter	Description
replyMessage	Send this <i>outbound</i> reply message.
requestMessage	Send a reply to this inbound request message; extract its reply subject to use as the subject of the outbound reply message.



### Warning

Give special attention to the order of the arguments to this method. Reversing the inbound and outbound messages can cause an infinite loop, in which the program repeatedly resends the inbound message to itself (and all other recipients).

## See Also

[TibrvMsg.getReplySubject\(\)](#)

# TibrvTransport.sendRequest()

*Method*

## Declaration

```
TibrvMsg sendRequest(  
    TibrvMsg message,  
    double timeout)  
throws TibrvException
```

## Purpose

Send a request message and wait for a reply.

## Blocking can Stall Event Dispatch



### Warning

This call blocks all other activity on its program thread. If appropriate, programmers must ensure that other threads continue dispatching events on its queues.

Parameter	Description
message	Send this message.
timeout	Maximum time (in seconds) that this call can block while waiting for a reply.  -1 indicates no timeout (wait without limit for a reply).

## Remarks

When the method receives a reply, it returns the reply. When the call does not receive a reply, it returns null, indicating timeout.

Programs that receive and process the request message cannot determine that the sender has blocked until a reply arrives.



The request message must have a valid destination subject; see [TibrvMsg.setSendSubject\(\)](#).

## Operation

This method operates in several synchronous steps:

### Procedure

1. Create an inbox name, and an event that listens to it. Overwrite any existing reply subject of message with the inbox name.
2. Send the outbound message.
3. Block until the listener receives a reply; if the time limit expires before a reply arrives, then return null. (The reply circumvents the event queue mechanism, so it is not necessary to explicitly call dispatch methods in the program.)
4. Return the reply as the value of this method.

# TibrvTransport.setBatchSize()

## Declaration

```
void setBatchSize(int numBytes)  
throws TibrvException
```

## Purpose

Enable outbound batching of data from IPM, and set the batch size (in bytes).

## Remarks

This type of batching is available only with the IPM library. It is not available with the standard (daemon-based) Rendezvous library.

When the batch size is greater than zero, IPM transfers data to the network in batches. This option can increase throughput, at the cost of higher latency.

When the batch size is zero, IPM transfers data to the network immediately, for lowest latency.

If you do not explicitly set the batch size using this call, then the default behavior disables outbound batching.



### Note

#### Contraindications

These conditions characterize situations in which we do not recommend batching:

- Data latency is *not* acceptable.
- Batch behavior does *not* produce measurable improvements in the performance of your application.

Parameter	Description
numBytes	Set the batch size (in bytes). Zero is a special value, which disables batching for the transport.

# TibrvTransport.setDescription()

*Method*

## Declaration

```
void setDescription(  
    String description)  
throws TibrvException
```

## Purpose

Set the program description parameter of a transport.

## Remarks

The description identifies your program to Rendezvous components. Browser administration interfaces display the description string of ordinary transport objects (however they do not display the description string of [TibrvCmTransport](#) or [TibrvCmQueueTransport](#) objects).

As a debugging aid, we recommend setting a unique description string for each transport. Use a string that distinguishes both the application and the role of the transport within it.

Parameter	Description
description	Use this string as the new program description.

## See Also

[TibrvTransport](#)

[TibrvTransport.getDescription\(\)](#)

# TibrvProcessTransport

*Class*

## Declaration

```
final class com.tibco.tibrv.TibrvProcessTransport  
    extends TibrvTransport
```

## Purpose

The intra-process transport delivers messages among the threads of a program.

## Remarks

The intra-process transport does not access the network.

The call [Tibrv.open\(\)](#) automatically creates the intra-process transport; [Tibrv.close\(\)](#) automatically destroys it; [Tibrv.processTransport\(\)](#) extracts it from the Rendezvous environment. Programs cannot create additional instances of this class, and cannot destroy the intra-process transport.

### Inherited Methods

[TibrvTransport.createInbox\(\)](#)

[TibrvTransport.isValid\(\)](#)

[TibrvTransport.send\(\)](#)

[TibrvTransport.sendReply\(\)](#)

[TibrvTransport.sendRequest\(\)](#)

---

[java.lang.Object.equals](#)

[java.lang.Object.getClass](#)

[java.lang.Object.hashCode](#)

[java.lang.Object.notify](#)

---

### Inherited Methods

`java.lang.Object.notifyAll`

`java.lang.Object.toString` (override)

`java.lang.Object.wait`

### Related Classes

[TibrvTransport](#)

[TibrvNetTransport](#)

### See Also

[Tibrv.processTransport\(\)](#)

# TibrvNetTransport

*Class*

## Declaration

```
abstract class com.tibco.tibrv.TibrvNetTransport  
    extends TibrvTransport
```

## Purpose

Deliver messages across a network.

## Remarks

This abstract class is the superclass of all network transport classes.

### Inherited Methods

[TibrvTransport.createInbox\(\)](#)

[TibrvTransport.destroy\(\)](#)

[TibrvTransport.isValid\(\)](#)

[TibrvTransport.send\(\)](#)

[TibrvTransport.sendReply\(\)](#)

[TibrvTransport.sendRequest\(\)](#)

---

[java.lang.Object.equals](#)

[java.lang.Object.getClass](#)

[java.lang.Object.hashCode](#)

[java.lang.Object.notify](#)

[java.lang.Object.notifyAll](#)

[java.lang.Object.toString](#) (override)

[java.lang.Object.wait](#)

---

## Related Classes

[TibrvTransport](#)

[TibrvProcessTransport](#)

[TibrvRvdTransport](#)

[TibrvCmTransport](#)

[TibrvCmQueueTransport](#)



# TibrvRvdTransport

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvRvdTransport  
    extends TibrvNetTransport
```

## Purpose

Deliver network messages through a Rendezvous daemon.

## Remarks

[TibrvRvdTransport](#) must operate in an [Tibrv.IMPL\\_NATIVE](#) environment (that is, it requires the JNI library). To determine the implementation, see [Tibrv.isNativeImpl\(\)](#).

Programs must explicitly destroy instances of this class. Rendezvous software keeps internal references to these objects, so the Java garbage collector does not delete them automatically.

Constant	Description
<code>TibrvRvdTransport.DEFAULT_RVD_PORT</code>	Unless you specify another port when creating the transport, <a href="#">TibrvRvdTransport</a> connects to the Rendezvous daemon through this default TCP port (7500).
<code>TibrvRvdTransport.DEFAULT_BATCH</code>	Default batch behavior. The transport transmits outbound messages to rvd as soon as possible.  This value is the initial default for all transports.
<code>TibrvRvdTransport.TIMER_BATCH</code>	Timer batch behavior. The transport accumulates outbound messages, and transmits them to rvd in batches—either when its buffer is full, or when a timer interval expires. (Programs cannot adjust the timer interval.)

Method	Description
<a href="#">TibrvRvdTransport()</a>	Create a transport that connects to a Rendezvous daemon.
<a href="#">TibrvRvdTransport.getDaemon()</a>	Return the socket where this transport connects to the Rendezvous daemon.
<a href="#">TibrvRvdTransport.getNetwork()</a>	Return the network interface that this transport uses for communication.
<a href="#">TibrvRvdTransport.getService()</a>	Return the effective service that this transport uses for communication.
<a href="#">TibrvRvdTransport.setBatchMode()</a>	Set the batch mode parameter of a transport.

### Inherited Methods

[TibrvTransport.createInbox\(\)](#)  
[TibrvTransport.destroy\(\)](#)  
[TibrvTransport.isValid\(\)](#)  
[TibrvTransport.send\(\)](#)  
[TibrvTransport.sendReply\(\)](#)  
[TibrvTransport.sendRequest\(\)](#)

[java.lang.Object.equals](#)  
[java.lang.Object.getClass](#)  
[java.lang.Object.hashCode](#)  
[java.lang.Object.notify](#)  
[java.lang.Object.notifyAll](#)  
[java.lang.Object.toString \(override\)](#)  
[java.lang.Object.wait](#)

## Related Classes

[TibrvTransport](#)

[TibrvProcessTransport](#)

[TibrvNetTransport](#)

[TibrvCmTransport](#)

[TibrvCmQueueTransport](#)

# TibrvRvdTransport()

*Constructor*

## Declaration

```
TibrvRvdTransport()  
    throws TibrvException  
TibrvRvdTransport(  
    java.lang.String service,  
    java.lang.String network,  
    java.lang.String daemon)  
    throws TibrvException  
TibrvRvdTransport(  
    java.lang.String service,  
    java.lang.String network,  
    java.lang.String daemon,  
    java.lang.String licenseTicket)  
    throws TibrvException
```

## Purpose

Create a transport that connects to a Rendezvous daemon.

## Method Forms

With no arguments, connect to rvd on the local computer using default values the service, network, and daemon parameters.

All arguments specify options for connecting to rvd.

## Connecting to the Rendezvous Daemon

Rendezvous daemon processes do the work of moving messages across a network. Every [TibrvRvdTransport](#) must connect to a Rendezvous daemon.

If a Rendezvous daemon process with a corresponding daemon parameter is already running, the transport connects to it.

If an appropriate Rendezvous local daemon is *not* running, the transport tries to start it. However, the transport does not attempt to start a *remote* daemon when none is running.

If the transport cannot connect to the Rendezvous daemon, the constructor throws an exception with the status code [TibrvStatus.DAEMON\\_NOT\\_FOUND](#).

## Description String

As a debugging aid, we recommend setting a unique description string for each transport. Use a string that distinguishes both the application and the role of the transport within it. See [TibrvTransport.setDescription\(\)](#).

Parameter	Description
service	<p>The Rendezvous daemon divides the network into logical partitions. Each <a href="#">TibrvRvdTransport</a> communicates on a single service; a transport can communicate only with other transports on the same service.</p> <p>To communicate on more than one service, a program must create more than one transport—one transport for each service.</p> <p>You can specify the service in several ways. For details, see <a href="#">Service Parameter</a> in TIBCO Rendezvous Concepts.</p> <p>null specifies the default rendezvous service.</p>
network	<p>Every network transport communicates with other transports over a single network interface. On computers with more than one network interface, the network parameter instructs the Rendezvous daemon to use a particular network for all outbound messages from this transport.</p> <p>To communicate over more than one network, programs must create more than one transport.</p> <p>You can specify the network in several ways. For details, see <a href="#">Network Parameter</a> in TIBCO Rendezvous Concepts.</p> <p>null specifies the primary network interface for the host computer.</p>
daemon	<p>The daemon parameter instructs the transport object about how and where to find the Rendezvous daemon and establish communication.</p> <p>For details, see <a href="#">Daemon Parameter</a> in TIBCO Rendezvous Concepts.</p> <p>You can specify a daemon on a remote computer. For details, see <a href="#">Remote Daemon</a> in TIBCO Rendezvous Concepts.</p>

Parameter	Description
	<p>If you specify a secure daemon, this string must be identical to as the <code>daemonName</code> argument of <a href="#">TibrvSdContext.setDaemonCert()</a>. See also, <a href="#">Secure Daemon</a> in TIBCO Rendezvous Concepts.</p> <p><code>null</code> specifies the default—find the local daemon on TCP socket 7500. (This default is not valid when the local daemon is a secure daemon.)</p>
<code>licenseTicket</code>	License tickets are no longer required. Values for this parameter are ignored.

## See Also

[TibrvRvdTransport.getDaemon\(\)](#)

[TibrvRvdTransport.getNetwork\(\)](#)

[TibrvRvdTransport.getService\(\)](#)

# TibrvRvdTransport.getDaemon()

*Method*

## Declaration

```
java.lang.String getDaemon()
```

## Purpose

Return the socket where this transport connects to the Rendezvous daemon.

# TibrvRvdTransport.getNetwork()

*Method*

## Declaration

```
java.lang.String getNetwork()
```

## Purpose

Return the network interface that this transport uses for communication.



# TibrvRvdTransport.getService()

*Method*

## Declaration

```
java.lang.String getService()
```

## Purpose

Return the effective service that this transport uses for communication.

# TibrvRvdTransport.setBatchMode()

*Method*

## Declaration

```
void setBatchMode(  
    int mode)  
    throws TibrvException  
static final int DEFAULT_BATCH = 0;  
static final int TIMER_BATCH = 1;
```

## Purpose

Set the batch mode parameter of a transport.

## Remarks

This type of batching is available only with the standard (daemon-based) Rendezvous library. It is not available with the IPM library.

The batch mode determines when the transport transmits outbound message data to rvd:

- As soon as possible (the initial default for all transports)
- Either when its buffer is full, or when a timer interval expires—either event triggers transmission to the daemon

Parameter	Description
mode	Use this value as the new batch mode.

Constant	Description
DEFAULT_BATCH	Default batch behavior. The transport transmits outbound messages to rvd as soon as possible.  This value is the initial default for all transports.

Constant	Description
TIMER_BATCH	Timer batch behavior. The transport accumulates outbound messages, and transmits them to rvd in batches—either when its buffer is full, or when a timer interval expires. (Programs cannot adjust the timer interval.)

## See Also

[Batch Modes for Transports](#) in TIBCO Rendezvous Concepts

# Virtual Circuits

---

Virtual circuits feature Rendezvous communication between two terminals over an exclusive, continuous, monitored connection.

## See Also

[Virtual Circuits](#) in TIBCO Rendezvous Concepts

# TibrvVcTransport

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvVcTransport  
    extends TibrvTransport
```

## Purpose

A virtual circuit transport object represents a terminal in a potential circuit.

## Remarks

A virtual circuit transport can fill the same roles as an ordinary transport. Programs can use them to create inbox names, send messages, create listeners and other events.

Instead of a constructor, this class has two create methods. These two methods also determine the protocol role of the transport object—one method creates a terminal that *accepts* connections, and another method creates a terminal that attempts to *connect*.

The two terminals play complementary roles as they attempt to establish a connection. However, this difference soon evaporates. After the connection is complete, the two terminals behave identically.

Method	Description
<a href="#">TibrvVcTransport.createAcceptVc()</a>	Create a virtual circuit accept object.
<a href="#">TibrvVcTransport.createConnectVc()</a>	Create a virtual circuit connect object
<a href="#">TibrvVcTransport.getConnectSubject()</a>	Return the connect subject of an accept terminal.

Method	Description
<a href="#">TibrvVcTransport.waitForVcConnection()</a>	Test the connection status of a virtual circuit.

## Broken Connection

The following conditions can close a virtual circuit connection:

- Contact is broken between the object and its terminal.
- The virtual circuit loses data in either direction (see [DATALOSS on page 240](#) in TIBCO Rendezvous Concepts).
- The partner program destroys its terminal object (or that terminal becomes invalid).
- The program destroys the object.
- The program destroys the object's ordinary transport.

## Direct Communication

Because virtual circuits rely on point-to-point messages between the two terminals, they can use direct communication to good advantage. To do so, both terminals must use network transports that enable direct communication.

For an overview, see [Direct Communication](#) in TIBCO Rendezvous Concepts.

For programming details, see [Specifying Direct Communication](#) in TIBCO Rendezvous Concepts.

### Inherited Methods

[TibrvTransport.createInbox\(\)](#)

[TibrvTransport.destroy\(\)](#)

[TibrvTransport.isValid\(\)](#)

[TibrvTransport.send\(\)](#)

[TibrvTransport.sendReply\(\)](#)

[TibrvTransport.sendRequest\(\)](#)

## Inherited Methods

Disabled Methods inherited from [TibrvTransport](#)

[TibrvTransport.getDescription\(\)](#)

[TibrvTransport.setDescription\(\)](#)

---

`java.lang.Object.equals`

`java.lang.Object.getClass`

`java.lang.Object.hashCode`

`java.lang.Object.notify`

`java.lang.Object.notifyAll`

`java.lang.Object.toString` (override)

`java.lang.Object.wait`

---

## Related Classes

[TibrvTransport](#)

[TibrvProcessTransport](#)

[TibrvNetTransport](#)

## See Also

[Virtual Circuits](#) in TIBCO Rendezvous Concepts

# TibrvVcTransport.createAcceptVc()

*Method*

## Declaration

```
TibrvVcTransport createAcceptVc(  
    TibrvTransport transport)  
    throws TibrvException
```

## Purpose

Create a virtual circuit accept object.

## Remarks

After this call returns, the program must extract the object's connect subject, and send it in a message to another program, inviting it to establish a virtual circuit. Furthermore, the *reply subject* of that invitation message must be this connect subject. To complete the virtual circuit, the second program must extract this subject from the invitation, and supply it to [TibrvVcTransport.createConnectVc\(\)](#).

Parameter	Description
transport	<p>The virtual circuit terminal uses this ordinary transport for communications.</p> <p>Programs may use this transport for other purposes.</p> <p>It is illegal to supply a virtual circuit transport object for this parameter (that is, you cannot nest a virtual circuit within another virtual circuit).</p>

## Test Before Using

Either of two conditions indicate that the connection is ready to use:

- The transport presents the [VC.CONNECTED](#) advisory.
- [TibrvVcTransport.waitForVcConnection\(\)](#) returns without error.



## Procedure

Immediately after this call, test *both* conditions with these two steps (in this order):

1. Listen on the virtual circuit transport object for the [VC.CONNECTED](#) advisory.
2. Call [TibrvVcTransport.waitForVcConnection\(\)](#) with zero as the timeout parameter.

For an explanation, see [Testing the New Connection](#) in TIBCO Rendezvous Concepts.

## See Also

[TibrvVcTransport.createConnectVc\(\)](#)

[TibrvVcTransport.getConnectSubject\(\)](#)

[TibrvVcTransport.waitForVcConnection\(\)](#)

[VC.CONNECTED](#) on [page 258](#) in TIBCO Rendezvous Concepts

[VC.DISCONNECTED](#) on [page 259](#) in TIBCO Rendezvous Concepts

# TibrvVcTransport.createConnectVc()

*Method*

## Declaration

```
TibrvVcTransport createConnectVc(  
    java.lang.String connectSubject,  
    TibrvTransport transport)  
throws TibrvException
```

## Purpose

Create a virtual circuit connect object

Parameter	Description
connectSubject	<p>The terminal uses this connect subject to establish a virtual circuit with an <i>accept</i> transport in another program.</p> <p>The program must receive this connect subject from the accepting program. The call to <a href="#">TibrvVcTransport.createAcceptVc()</a> creates and returns this subject.</p>
transport	<p>The virtual circuit terminal uses this ordinary transport for communications.</p> <p>Programs may use this transport for other purposes.</p> <p>It is illegal to supply a virtual circuit transport object for this parameter (that is, you cannot nest a virtual circuit within another virtual circuit).</p>

## Test Before Using

Either of two conditions indicate that the connection is ready to use:

- The transport presents the [VC.CONNECTED](#) advisory.
- [TibrvVcTransport.waitForVcConnection\(\)](#) returns without error.

## Procedure

Immediately after this call, test *both* conditions with these two steps (in this order):

1. Listen on the virtual circuit transport object for the [VC.CONNECTED](#) advisory.
2. Call [TibrvVcTransport.waitForVcConnection\(\)](#) with zero as the timeout parameter.

For an explanation, see [Testing the New Connection](#) in TIBCO Rendezvous Concepts.

## See Also

[TibrvVcTransport.createAcceptVc\(\)](#)

[TibrvVcTransport.getConnectSubject\(\)](#)

[TibrvVcTransport.waitForVcConnection\(\)](#)

[VC.CONNECTED](#) on [page 258](#) in TIBCO Rendezvous Concepts

[VC.DISCONNECTED](#) on [page 259](#) in TIBCO Rendezvous Concepts

# TibrVcTransport.getConnectSubject()

*Method*

## Declaration

```
java.lang.String getConnectSubject()  
throws TibrVcException
```

## Purpose

Return the connect subject of an accept terminal.

## Remarks

After creating an accept terminal, the program must use this method to extract its connect subject, and send it in a message to another program, inviting it to establish a virtual circuit. Furthermore, the *reply subject* of that invitation message must be this connect subject. To complete the virtual circuit, the second program must extract this subject from the invitation, and supply it to [TibrVcTransport.createConnectVc\(\)](#).

## See Also

[TibrVcTransport.createAcceptVc\(\)](#)

[TibrVcTransport.createConnectVc\(\)](#)

# TibrvVcTransport.waitForVcConnection()

*Method*

## Declaration

```
void waitForVcConnection(  
    java.lang.double timeout)  
    throws TibrvException
```

## Purpose

Test the connection status of a virtual circuit.

## Remarks

This method tests (and can block) until this virtual circuit transport object has established a connection with its opposite terminal. You may call this method for either an accept terminal or a connect terminal.

This method produces the same information as the virtual circuit advisory messages—but it produces it synchronously (while advisories are asynchronous). Programs can use this method not only to test the connection, but also to block until the connection is ready to use.

For example, a program can create a terminal object, then call this method to wait until the connection completes.

Parameter	Description
timeout	<p>This parameter determines the behavior of the call:</p> <ul style="list-style-type: none"><li>• For a quick test of current connection status, supply zero. The call returns immediately, without blocking.</li><li>• To wait for a new terminal to establish a connection, supply a reasonable positive value. The call returns either when the connection is complete, or when this time limit elapses.</li></ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>To wait indefinitely for a usable connection, supply -1. The call returns when the connection is complete. If the connection was already complete and is now broken, the call returns immediately.</li> </ul>

## Results

When the connection is complete (ready to use), this method returns normally. Otherwise it throws an exception; the status value in the exception yields additional information about the unusable connection.

Status	Description
<a href="#">TibrvStatus.TIMEOUT</a>	The connection is not yet complete, but the non-negative time limit for waiting has expired.
<a href="#">TibrvStatus.TIBRV_VC_NOT_CONNECTED</a>	The connection was formerly complete, but is now irreparably broken.

## See Also

[TibrvVcTransport.createAcceptVc\(\)](#)

[TibrvVcTransport.createConnectVc\(\)](#)

[Testing the New Connection](#) in TIBCO Rendezvous Concepts

[VC.CONNECTED on page 258](#) in TIBCO Rendezvous Concepts

[VC.DISCONNECTED on page 259](#) in TIBCO Rendezvous Concepts

# Fault Tolerance

---

Rendezvous fault tolerance software coordinates a group of redundant processes into a fault-tolerant distributed program. Some processes actively fulfill the tasks of the program, while other processes wait in readiness. When one of the active processes fails, another process rapidly assumes active duty.

# Fault Tolerance Road Map

For a complete discussion of concepts and operating principles, see Fault Tolerance Concepts in TIBCO Rendezvous Concepts.

For suggestions to help you design programs using fault tolerance features, see Fault Tolerance Programming in TIBCO Rendezvous Concepts.

For step-by-step hints for implementing fault-tolerant systems, see Developing Fault-Tolerant Programs in TIBCO Rendezvous Concepts.

Fault tolerance software uses advisory messages to inform programs of status changes. For details, see Fault Tolerance (RVFT) Advisory Messages in TIBCO Rendezvous Concepts.

If your application distributes fault-tolerant processes across network boundaries, you must configure the Rendezvous routing daemons to exchange `_RVFT` administrative messages. For details, see Fault Tolerance in TIBCO Rendezvous Administration, and discuss with your network administrator.



# TibrvFtMember

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvFtMember  
    extends TibrvEvent
```

## Purpose

Represent membership in a fault tolerance group.

## Remarks

Upon creating this object, the program joins a fault tolerance group.

By destroying a member object, the program withdraws its membership in the fault tolerance group.

Programs must destroy instances of this class. Rendezvous software keeps internal references to these objects, so the Java garbage collector does not delete them automatically.

Destroying the queue or transport of a member object automatically destroys the member object as well.

## Constants

Each of these constant fields is a token designating a command to a fault tolerance callback method. The program's callback method receives one of these tokens in a parameter, and interprets it as an instruction from the Rendezvous fault tolerance software as described in this table (see also, Fault Tolerance Callback Actions in TIBCO Rendezvous Concepts).

Constant	Description
<code>TibrvFtMember.PREPARE_TO_ACTIVATE</code>	<p>Prepare to activate (hint).</p> <p>Rendezvous fault tolerance software passes this token to the callback method to instruct the program to make itself ready to activate on short notice—so that if the callback method subsequently receives the instruction to activate, it can do so without delay.</p> <p>This token is a hint, indicating that the program might soon receive an instruction to activate. It does not guarantee that an activate instruction will follow, nor that any minimum time will elapse before an activate instruction follows.</p>
<code>TibrvFtMember.ACTIVATE</code>	<p>Activate immediately.</p> <p>Rendezvous fault tolerance software passes this token to the callback method to instruct the program to activate.</p>
<code>TibrvFtMember.DEACTIVATE</code>	<p>Deactivate immediately.</p> <p>Rendezvous fault tolerance software passes this token to the callback method to instruct the program to deactivate.</p>

Method	Description
<code>TibrvFtMember()</code>	Create a member of a fault tolerance group.
<code>TibrvFtMember.destroy()</code>	Destroy a member of a fault tolerance group.
<code>TibrvFtMember.getActivationInterval()</code>	Extract the activation interval of a fault tolerance member.

Method	Description
<a href="#">TibrvFtMember.getActiveGoal()</a>	Extract the active goal of a fault tolerance member.
<a href="#">TibrvFtMember.getGroupName()</a>	Extract the group name of a fault tolerance member.
<a href="#">TibrvFtMember.getHeartbeatInterval()</a>	Extract the heartbeat interval of a fault tolerance member.
<a href="#">TibrvFtMember.getPreparationInterval()</a>	Extract the preparation interval of a fault tolerance member.
<a href="#">TibrvFtMember.getQueue()</a>	Extract the event queue of a fault tolerance member.
<a href="#">TibrvFtMember.getTransport()</a>	Extract the transport of a fault tolerance member.
<a href="#">TibrvFtMember.getWeight()</a>	Extract the weight of a fault tolerance member.
<a href="#">TibrvFtMember.setWeight()</a>	Change the weight of a fault tolerance member within its group.

### Inherited Methods

[TibrvEvent.getClosure\(\)](#)

[TibrvEvent.isValid\(\)](#)

[java.lang.Object.equals](#)

[java.lang.Object.getClass](#)

[java.lang.Object.hashCode](#)

[java.lang.Object.notify](#)

[java.lang.Object.notifyAll](#)

[java.lang.Object.toString \(override\)](#)

### Inherited Methods

`java.lang.Object.wait`

## Related Classes

[TibrvEvent](#)

[TibrvFtMonitor](#)

## See Also

[TibrvFtMemberCallback](#)

# TibrvFtMember()

*Constructor*

## Declaration

```
TibrvFtMember(  
    TibrvQueue queue,  
    TibrvFtMemberCallback callback,  
    TibrvTransport transport,  
    java.lang.String groupName,  
    int weight,  
    int activeGoal,  
    double heartbeatInterval,  
    double preparationInterval,  
    double activationInterval,  
    java.lang.Object closure)  
throws TibrvException
```

## Purpose

Create a member of a fault tolerance group.

## Remarks

Upon creating a member object, the program becomes a member of the group.

A program may hold simultaneous memberships in several distinct fault tolerance groups. For examples, see [Multiple Groups](#) in TIBCO Rendezvous Concepts.

Avoid joining the same group twice. It is illegal for a program to maintain more than one membership in any one fault tolerance group. The constructor does not guard against this illegal situation, and results are unpredictable.

All arguments are required except for preparationInterval (which may be zero) and closure (which may be null).

## Intervals

The heartbeat interval must be less than the activation interval. If the preparation interval is non-zero, it must be greater than the heartbeat interval and less than the activation

interval. It is an error to violate these rules.

In addition, intervals must be reasonable for the hardware and network conditions. For information and examples, see [Step 4: Choose the Intervals](#) in TIBCO Rendezvous Concepts.

## Group Name

The group name must be a legal Rendezvous subject name (see [Subject Names](#) in TIBCO Rendezvous Concepts). You may use names with several elements; for examples, see [Multiple Groups](#) in TIBCO Rendezvous Concepts.

Parameter	Description
queue	Place fault tolerance events for this member on this event queue.
callback	On dispatch, process the event with this callback object.
transport	Use this transport for fault tolerance internal protocol messages (such as heartbeat messages).
groupName	Join the fault tolerant group with this name.  The group name must conform to the syntax required for Rendezvous subject names. For details, see <a href="#">Subject Names</a> in TIBCO Rendezvous Concepts.
weight	Weight represents the ability of this member to fulfill its purpose, relative to other members of the same fault tolerance group. Rendezvous fault tolerance software uses relative weight values to select which members to activate; members with higher weight take precedence over members with lower weight.  Acceptable values range from 1 to 65535. Zero is a special, reserved value; Rendezvous fault tolerance software assigns zero weight to processes with resource errors, so they only activate when no other members are available.  For more information, see <a href="#">Rank and Weight</a> in TIBCO Rendezvous Concepts.
activeGoal	Rendezvous fault tolerance software sends callback instructions to

Parameter	Description
	<p>maintain this number of active members.</p> <p>Acceptable values range from 1 to 65535.</p>
heartbeatInterval	<p>When this member is active, it sends heartbeat messages at this interval (in seconds).</p> <p>The interval must be positive. To determine the correct value, see <a href="#">Step 4: Choose the Intervals</a> in TIBCO Rendezvous Concepts.</p>
preparationInterval	<p>When the heartbeat signal from one or more active members has been silent for this interval (in seconds), Rendezvous fault tolerance software issues an early warning hint (<a href="#">TibrvFtMember.PREPARE_TO_ACTIVATE</a>) to the ranking inactive member. This warning lets the inactive member prepare to activate, for example, by connecting to a database server, or allocating memory.</p> <p>The interval must be non-negative. Zero is a special value, indicating that the member does not need advance warning to activate; Rendezvous fault tolerance software never issues a <a href="#">TibrvFtMember.PREPARE_TO_ACTIVATE</a> hint when this value is zero. To determine the correct value, see <a href="#">Step 4: Choose the Intervals</a> in TIBCO Rendezvous Concepts.</p>
activationInterval	<p>When the heartbeat signal from one or more active members has been silent for this interval (in seconds), Rendezvous fault tolerance software considers the silent member to be lost, and issues the instruction to activate (<a href="#">TibrvFtMember.ACTIVATE</a>) to the ranking inactive member.</p> <p>When a new member joins a group, Rendezvous fault tolerance software identifies the new member to existing members (if any), and then waits for this interval to receive identification from them in return. If, at the end of this interval, it determines that too few members are active, it issues the activate instruction (<a href="#">TibrvFtMember.ACTIVATE</a>) to the new member.</p> <p>Then interval must be positive. To determine the correct value,</p>

Parameter	Description
	see <a href="#">Step 4: Choose the Intervals</a> in TIBCO Rendezvous Concepts.
closure	Store this closure data in the member object.

## See Also

[TibrvFtMember](#).

[TibrvFtMemberCallback](#).

[TibrvFtMember.destroy\(\)](#).

[Step 1: Choose a Group Name](#) in TIBCO Rendezvous Concepts

[Step 2: Choose the Active Goal](#) in TIBCO Rendezvous Concepts

[Step 4: Choose the Intervals](#) in TIBCO Rendezvous Concepts

[Step 5: Program Start Sequence](#) in TIBCO Rendezvous Concepts



# TibrvFtMember.destroy()

*Method*

## Declaration

```
void destroy()
```

## Purpose

Destroy a member of a fault tolerance group.

## Remarks

By destroying a member object, the program cancels or withdraws its membership in the group.

This method has two effects:

- If this member is active, stop sending the heartbeat signal.
- Reclaim the program storage associated with this member.

Once a program withdraws from a group, it no longer receives fault tolerance events. One direct consequence is that an active program that withdraws can never receive an instruction to deactivate.

## See Also

[TibrvFtMember\(\)](#)

[TibrvFtMember.isValid\(\)](#)

# TibrvFtMember.getActivationInterval()

*Method*

## Declaration

```
double getActivationInterval()
```

## Purpose

Extract the activation interval of a fault tolerance member.

## See Also

[TibrvFtMember\(\)](#)

# TibrvFtMember.getActiveGoal()

*Method*

## Declaration

```
int getActiveGoal()
```

## Purpose

Extract the active goal of a fault tolerance member.

## See Also

[TibrvFtMember\(\)](#)

# TibrvFtMember.getGroupName()

*Method*

## Declaration

```
java.lang.String getGroupName()
```

## Purpose

Extract the group name of a fault tolerance member.

## See Also

[TibrvFtMember\(\)](#)

# TibrvFtMember.getHeartbeatInterval()

*Method*

## Declaration

```
double getHeartbeatInterval()
```

## Purpose

Extract the heartbeat interval of a fault tolerance member.

## Remarks

If the member is invalid, this method returns null.

## See Also

[TibrvFtMember\(\)](#)

# TibrvFtMember.getPreparationInterval()

*Method*

## Declaration

```
double getPreparationInterval()
```

## Purpose

Extract the preparation interval of a fault tolerance member.

## Remarks

If the member is invalid, this method returns null.

## See Also

[TibrvFtMember\(\)](#)

# TibrvFtMember.getQueue()

*Method*

## Declaration

```
TibrvQueue getQueue()
```

## Purpose

Extract the event queue of a fault tolerance member.

## Remarks

If the member is invalid, this method returns null.

## See Also

[TibrvQueue](#)

[TibrvFtMember\(\)](#)

# TibrvFtMember.getTransport()

*Method*

## Declaration

```
TibrvTransport getTransport()
```

## Purpose

Extract the transport of a fault tolerance member.

## See Also

[TibrvTransport](#)

[TibrvFtMember\(\)](#)



# TibrvFtMember.getWeight()

*Method*

## Declaration

```
int getWeight()
```

## Purpose

Extract the weight of a fault tolerance member.

## Remarks

If the member is invalid, this method returns null.

## See Also

[TibrvFtMember\(\)](#)

# TibrvFtMember.isValid()

*Method*

## Declaration

```
boolean isValid()
```

## Purpose

Test validity of a fault tolerance member object.

## Remarks

Returns true if the member is valid; false if the member has been destroyed or is otherwise invalid.

## See Also

[TibrvFtMember.destroy\(\)](#)

# TibrvFtMember.setWeight()

*Method*

## Declaration

```
void setWeight(int weight)  
    throws TibrvException
```

## Purpose

Change the weight of a fault tolerance member within its group.

## Remarks

Weight summarizes the relative suitability of a member for its task, relative to other members of the same fault tolerance group. That suitability is a combination of computer speed and load factors, network bandwidth, computer and network reliability, and other factors. Programs may reset their weight when any of these factors change, overriding the previous assigned weight.

You can use relative weights to indicate priority among group members.

Zero is a special value; Rendezvous fault tolerance software assigns zero weight to processes with resource errors, so they only activate when no other members are available. Programs must always assign weights greater than zero.

When Rendezvous fault tolerance software requests a resource but receives an error (for example, the member process cannot allocate memory, or start a timer), it attempts to send the member process a [DISABLING\\_MEMBER](#) advisory message, and sets the member's weight to zero, effectively disabling the member. Weight zero implies that this member is active only as a last resort—when no other members outrank it. (However, if the disabled member process does become active, it might not operate correctly.)

Parameter	Description
weight	The new weight value. See <a href="#">weight</a> .

## See Also

[Adjusting Member Weights](#) in TIBCO Rendezvous Concepts.

# TibrvFtMemberCallback

*Interface*

## Declaration

```
interface com.tibco.tibrv.TibrvFtMemberCallback
```

## Purpose

Process fault tolerance events for a group member.

## Remarks

Implement this interface to process fault tolerance events.

Method	Description
<a href="#">TibrvFtMemberCallback.onFtAction()</a>	Process fault tolerance events for a group member.

## See Also

[TibrvFtMember\(\)](#)

# TibrvFtMemberCallback.onFtAction()

*Method*

## Declaration

```
void onFtAction(  
    TibrvFtMember member,  
    java.lang.String groupName,  
    int action)
```

## Purpose

Process fault tolerance events for a group member.

## Remarks

Each member program of a fault tolerance group must implement this method. Programs register a member callback object (and this method) with each call to [TibrvFtMember\(\)](#).

Rendezvous fault tolerance software queues a member action event in three situations. In each case, it passes a different action argument, instructing the callback method to activate, deactivate, or prepare to activate the program.

- When the number of active members drops below the active goal, the fault tolerance callback method (in the ranking inactive member process) receives the token [TibrvFtMember.ACTIVATE](#); the callback method must respond by assuming the duties of an active member.
- When the number of active members exceeds the active goal, the fault tolerance callback method (in any active member that is outranked by another active member) receives the action token [TibrvFtMember.DEACTIVATE](#); the callback method must respond by switching the program to its inactive state.
- When the number of active members equals the active goal, and Rendezvous fault tolerance software detects that it might soon decrease below the active goal, the fault tolerance callback method (in the ranking inactive member) receives the action token [TibrvFtMember.PREPARE\\_TO\\_ACTIVATE](#); the callback method must respond by making the program ready to activate immediately. For example, preparatory steps might include time-consuming tasks such as connecting to a database. If the

callback method subsequently receives the [TibrvFtMember.ACTIVATE](#) token, it will be ready to activate without delay.

For additional information see [Fault Tolerance Callback Actions](#) in TIBCO Rendezvous Concepts.

Parameter	Description
member	This parameter receives the member object.
groupName	This parameter receives a string denoting the name of the fault tolerance group.
action	This parameter receives a token that instructs the callback method to activate, deactivate or prepare to activate. See <a href="#">Constants</a> .

## See Also

[TibrvFtMember\(\)](#).

# TibrvFtMonitor

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvFtMonitor  
    extends TibrvEvent
```

## Purpose

Monitor a fault tolerance group.

## Remarks

Upon creating this object, the program monitors a fault tolerance group.

Monitors are passive—they do not affect the group members in any way.

Rendezvous fault tolerance software queues a monitor event whenever the number of active members in the group changes—either it detects a new heartbeat, or it detects that the heartbeat from a previously active member is now silent, or it receives a message from the fault tolerance component of an active member indicating deactivation or termination.

The monitor callback method receives the number of active members as an argument.

By destroying a monitor object, the program stops monitoring the fault tolerance group.

Programs must destroy instances of this class. Rendezvous software keeps internal references to these objects, so the Java garbage collector does not delete them automatically.

Destroying the queue or transport of a monitor automatically destroys the monitor as well.

Method	Description
<a href="#">TibrvFtMonitor()</a>	<a href="#">Monitor a fault tolerance group.</a>
<a href="#">TibrvFtMonitor.destroy()</a>	<a href="#">Stop monitoring a fault tolerance group, and</a>



Method	Description
	<a href="#">free associated resources.</a>
<a href="#">TibrvFtMonitor.getGroupName()</a>	Extract the group name of a fault tolerance monitor.
<a href="#">TibrvFtMonitor.getTransport()</a>	Extract the transport of a fault tolerance monitor.

### Inherited Methods

[TibrvEvent.getClosure\(\)](#)

[TibrvEvent.getQueue\(\)](#)

[TibrvEvent.isValid\(\)](#)

[java.lang.Object.equals](#)

[java.lang.Object.getClass](#)

[java.lang.Object.hashCode](#)

[java.lang.Object.notify](#)

[java.lang.Object.notifyAll](#)

[java.lang.Object.toString \(override\)](#)

[java.lang.Object.wait](#)

## Related Classes

[TibrvEvent](#)

[TibrvFtMember](#)

## See Also

[TibrvFtMonitorCallback](#)

# TibrvFtMonitor()

*Constructor*

## Declaration

```
TibrvFtMonitor(  
    TibrvQueue      queue,  
    TibrvFtMonitorCallback callback,  
    TibrvTransport  transport,  
    java.lang.String groupName,  
    double          lostInterval,  
    java.lang.Object closure)  
throws TibrvException
```

## Purpose

Monitor a fault tolerance group.

## Remarks

The monitor callback method receives the number of active members as an argument.

The group need not have any members at the time of this constructor call.

Parameter	Description
queue	Place events for this monitor on this event queue.
callback	On dispatch, process the event with this callback method.
transport	Listen on this transport for fault tolerance internal protocol messages (such as heartbeat messages).
groupName	Monitor the fault tolerant group with this name.  The group name must conform to the syntax required for Rendezvous subject names. For details, see <a href="#">Subject</a>

Parameter	Description
	<a href="#">Names</a> in TIBCO Rendezvous Concepts. See also, <a href="#">Group Name</a> .
lostInterval	<p>When the heartbeat signal from an active member has been silent for this interval (in seconds), Rendezvous fault tolerance software considers that member lost, and queues a monitor event.</p> <p>The interval must be positive. To determine the correct value, see <a href="#">Step 4: Choose the Intervals</a> in TIBCO Rendezvous Concepts.</p> <p>See also, <a href="#">Lost Interval</a>.</p>
closure	Store this closure data in the monitor object.

## Lost Interval

The monitor uses the lostInterval to determine whether a member is still active. When the heartbeat signal from an active member has been silent for this interval (in seconds), the monitor considers that member lost, and queues a monitor event.

We recommend setting the lostInterval identical to the group's activationInterval, so the monitor accurately reflects the behavior of the group members.

## Group Name

The group name must be a legal Rendezvous subject name (see [Subject Names](#) in TIBCO Rendezvous Concepts). You may use names with several elements; for examples, see [Multiple Groups](#) in TIBCO Rendezvous Concepts.

## See Also

[TibrvFtMonitorCallback](#).

[TibrvFtMonitor.destroy\(\)](#).

# TibrvFtMonitor.destroy()

*Method*

## Declaration

```
void destroy()
```

## Purpose

Stop monitoring a fault tolerance group, and free associated resources.

## Remarks

This method throws an exception when the monitor object is already invalid, or when its queue or transport are invalid.

## See Also

[TibrvFtMonitor\(\)](#)

# TibrvFtMonitor.getGroupName()

*Method*

## Declaration

```
java.lang.String getGroupName()
```

## Purpose

Extract the group name of a fault tolerance monitor.

## Remarks

If the monitor is invalid, this method returns null.

## See Also

[TibrvFtMonitor\(\)](#)

# TibrvFtMonitor.getTransport()

*Method*

## Declaration

```
TibrvTransport getTransport()
```

## Purpose

Extract the transport of a fault tolerance monitor.

## See Also

[TibrvTransport](#)

[TibrvFtMonitor\(\)](#)

# TibrvFtMonitorCallback

*Interface*

## Declaration

```
interface com.tibco.tibrv.TibrvFtMonitorCallback
```

## Purpose

Process fault tolerance events for a monitor.

## Remarks

Implement this interface to process fault tolerance monitor events.

Method	Description
<a href="#">TibrvFtMonitorCallback.onFtMonitor()</a>	<a href="#">Process fault tolerance events for a monitor.</a>

## See Also

[TibrvFtMonitor\(\)](#)

# TibrvFtMonitorCallback.onFtMonitor()

## Declaration

```
void onFtMonitor(  
    TibrvFtMonitor Monitor,  
    java.lang.String groupName,  
    int numActiveMembers)
```

## Purpose

Process fault tolerance events for a monitor.

## Remarks

A program must define a method of this type as a prerequisite to monitor a fault tolerance group. Programs register a monitor callback method with each call to [TibrvFtMonitor\(\)](#).

Rendezvous fault tolerance software queues a monitor event whenever the number of active members in the group changes.

A program need not be a member of a group in order to monitor that group. Programs that do not monitor need not define a monitor callback method.

Parameter	Description
monitor	This parameter receives the monitor object.
groupName	This parameter receives a string denoting the name of the fault tolerance group.
numActiveMembers	This parameter receives the number of group members now active.

## See Also

[TibrvFtMonitor\(\)](#).



# Certified Message Delivery

---

Although Rendezvous communications are highly reliable, some applications require even stronger assurances of delivery. Certified delivery features offers greater certainty of delivery—even in situations where processes and their network connections are unstable.

## See Also

This API implements Rendezvous certified delivery features. For a complete discussion, see [Certified Message Delivery](#) in TIBCO Rendezvous Concepts.

Certified delivery software uses advisory messages extensively. For example, advisories inform sending and receiving programs of the delivery status of each message. For complete details, see [Certified Message Delivery \(RVCM\) Advisory Messages](#) in TIBCO Rendezvous Concepts.

If your application sends or receives certified messages across network boundaries, you must configure the Rendezvous routing daemons to exchange `_RVCM` administrative messages. For details, see [Certified Message Delivery](#) in TIBCO Rendezvous Administration.

Some programs require certified delivery to *one of  $n$*  worker processes. See [Distributed Queue](#) in TIBCO Rendezvous Concepts.

# TibrvCmListener

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvCmListener  
    extends TibrvListener
```

## Purpose

A certified delivery listener object listens for labeled messages and certified messages.

## Remarks

Each call to the constructor [TibrvCmListener\(\)](#) results in a new certified delivery listener, which represents your program's listening interest in a stream of labeled messages and certified messages. Rendezvous software uses the same listener object to signal each occurrence of such an event.

We recommend that programs explicitly destroy each certified delivery listener object using [TibrvCmListener.destroy\(\)](#). Destroying a certified listener object cancels the program's immediate interest in that event, and frees its storage; nonetheless, a parameter to the destroy call determines whether certified delivery agreements continue to persist beyond the destroy call.

Programs must destroy instances of this class. Rendezvous software keeps internal references to these objects, so the Java garbage collector does not delete them automatically.

Destroying the queue or the certified delivery transport of a listener object automatically destroys the listener as well (but certified delivery agreements continue to persist).

Method	Description
<a href="#">TibrvCmListener()</a>	<a href="#">Listen for messages that match the subject, and request certified delivery when available.</a>

Method	Description
<a href="#">TibrvCmListener.destroy()</a>	Destroy a certified delivery listener.
<a href="#">TibrvCmListener.confirmMsg()</a>	Explicitly confirm delivery of a certified message.
<a href="#">TibrvCmListener.isExplicitConfirm()</a>	Test whether this listener expects explicit confirmation of delivery.
<a href="#">TibrvCmListener.setExplicitConfirm()</a>	Override automatic confirmation of delivery for this listener.

### Inherited Methods

[TibrvListener.getSubject\(\)](#)

[TibrvListener.getTransport\(\)](#)

[TibrvEvent.getClosure\(\)](#)

[TibrvEvent.getQueue\(\)](#)

[TibrvEvent.isValid\(\)](#)

[java.lang.Object.equals](#)

[java.lang.Object.getClass](#)

[java.lang.Object.hashCode](#)

[java.lang.Object.notify](#)

[java.lang.Object.notifyAll](#)

[java.lang.Object.toString \(override\)](#)

[java.lang.Object.wait](#)

## Related Classes

[TibrvEvent](#)

[TibrvListener](#)

# TibrvCmListener()

*Constructor*

## Declaration

```
TibrvCmListener(  
    TibrvQueue queue,  
    TibrvMsgCallback callback,  
    TibrvCmTransport cmTransport,  
    java.lang.String subject,  
    java.lang.Object closure)  
throws TibrvException
```

## Purpose

Listen for messages that match the subject, and request certified delivery when available.

Parameter	Description
queue	For each inbound message, place the listener event on this event queue.
callback	On dispatch, process the event with this callback object.
cmTransport	Listen for inbound messages on this certified delivery transport.
subject	Listen for inbound messages with subjects that match this specification. Wildcard subjects are permitted. The empty string is not a legal subject name.
closure	Store this closure data in the event object.

## Activation and Dispatch

Details of listener event semantics are identical to those for ordinary listeners; see [Activation and Dispatch](#).

## Inbox Listener

To receive unicast (point-to-point) messages, listen to a unique inbox subject name. First call [TibrvTransport.createInbox\(\)](#) to create the unique inbox name; then call [TibrvCmListener\(\)](#) to begin listening. Remember that other programs have no information about an inbox until the listening program uses it as a reply subject in an outbound message.

## See Also

[TibrvCmListener](#)

[TibrvCmTransport.destroy\(\)](#)

[TibrvListener.getSubject\(\)](#)

[TibrvTransport.createInbox\(\)](#)

# TibrvCmListener.destroy()

*Method*

## Declaration

```
void destroy()  
void destroy(  
    boolean  cancelAgreements)
```

## Purpose

Destroy a certified delivery listener.

Parameter	Description
cancelAgreements	<p>true cancels all certified delivery agreements of this listener; certified senders delete from their ledgers all messages sent to this listener.</p> <p>false leaves all certified delivery agreements in effect, so certified senders continue to store messages.</p>

## Canceling Agreements

When destroying a certified delivery listener, a program can either cancel its certified delivery agreements with senders, or let those agreements persist (so a successor listener can receive the messages covered by those agreements).

When canceling agreements, each (previously) certified sender transport receives a [REGISTRATION.CLOSED](#) advisory. Successor listeners cannot receive old messages.

## See Also

[TibrvCmListener](#)

# TibrvCmListener.confirmMsg()

*Method*

## Declaration

```
void confirmMsg(  
    TibrvMsg message)  
throws TibrvException
```

## Purpose

Explicitly confirm delivery of a certified message.

## Remarks

Use this method only in programs that override automatic confirmation (see [TibrvCmListener.setExplicitConfirm\(\)](#)). The default behavior of certified listeners is to automatically confirm delivery when the callback method returns.

Parameter	Description
message	Confirm receipt of this message.

## Unregistered Message

When a CM listener receives a labeled message, its behavior depends on context:

- If a CM listener is registered for certified delivery, it presents the supplementary information to the callback method. If the sequence number is present, then the receiving program can confirm delivery.
- If a CM listener is *not* registered for certified delivery with the sender, it presents the sender's name to the callback method, but omits the sequence number. In this case, the receiving program cannot confirm delivery; [TibrvCmListener.confirmMsg\(\)](#) throws an exception with the status code [TibrvStatus.NOT\\_PERMITTED](#).

Notice that the first labeled message that a program receives on a subject might not be certified; that is, the sender has not registered a certified delivery agreement with

the listener. If appropriate, the certified delivery library automatically requests that the sender register the listener for certified delivery. (See Discovery and Registration for Certified Delivery in TIBCO Rendezvous Concepts.)

A labeled but uncertified message can also result when the sender explicitly disallows or removes the listener.

## See Also

[TibrvCmListener](#)

[TibrvCmListener.setExplicitConfirm\(\)](#)



# TibrvCmListener.isExplicitConfirm()

*Method*

## Declaration

```
boolean isExplicitConfirm()
```

## Purpose

Test whether this listener expects explicit confirmation of delivery.

## Remarks

The default behavior of certified listeners is to automatically confirm delivery when the callback method returns (see [TibrvMsgCallback.onMsg\(\)](#)).

[TibrvCmListener.setExplicitConfirm\(\)](#) selectively overrides this behavior for this specific listener (without affecting other listeners).

## See Also

[TibrvCmListener](#)

[TibrvMsgCallback.onMsg\(\)](#)

[TibrvCmListener.confirmMsg\(\)](#)

[TibrvCmListener.setExplicitConfirm\(\)](#)

# TibrvCmListener.setExplicitConfirm()

*Method*

## Declaration

```
void setExplicitConfirm()  
throws TibrvException
```

## Purpose

Override automatic confirmation of delivery for this listener.

## Remarks

The default behavior of certified listeners is to automatically confirm delivery when the callback method returns (see [TibrvMsgCallback.onMsg\(\)](#)). This call selectively overrides this behavior for this specific listener (without affecting other listeners).

By overriding automatic confirmation, the listener assumes responsibility to explicitly confirm each inbound certified message by calling [TibrvCmListener.confirmMsg\(\)](#).

Consider overriding automatic confirmation when the processing of inbound messages involves activity that is asynchronous with respect to the message callback method; for example, computations in other threads or additional network communications.

No method exists to restore the default behavior—that is, to reverse the effect of this method.

## See Also

[TibrvCmListener](#)

[TibrvMsgCallback.onMsg\(\)](#)

[TibrvCmListener.confirmMsg\(\)](#)

[TibrvCmListener.isExplicitConfirm\(\)](#)

# TibrvCmTransport

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvCmTransport  
    extends TibrvTransport
```

## Purpose

A certified delivery transport object implements the CM delivery protocol for messages.

## Remarks

Each certified delivery transport employs a [TibrvTransport](#) for network communications. The [TibrvCmTransport](#) adds the accounting mechanisms needed for delivery tracking and certified delivery.

Several [TibrvCmTransport](#) objects can employ a [TibrvTransport](#), which also remains available for its own ordinary listeners and for sending ordinary messages. Destroying this [TibrvTransport](#) causes the [TibrvCmTransport](#) to destroy itself as well (along with all its listeners, while preserving their certified delivery agreements).

Destroying a certified delivery transport object invalidates subsequent certified send calls on that object, invalidates any certified listeners using that transport (while preserving the certified delivery agreements of those listeners).

Programs must explicitly destroy each certified delivery transport object. Rendezvous software keeps internal references to these objects, so the Java garbage collector does not delete them automatically.

Method	Description
<a href="#">TibrvCmTransport()</a>	Create a transport for certified delivery.

Method	Description
<a href="#">TibrvCmTransport.addListener()</a>	Pre-register an anticipated listener.
<a href="#">TibrvCmTransport.allowListener()</a>	Invite the named receiver to reinstate certified delivery for its listeners, superseding the effect of any previous disallow calls.
<a href="#">TibrvCmTransport.destroy()</a> <a href="#">TibrvCmTransport.destroyEx()</a>	Destroy a certified delivery transport.
<a href="#">TibrvCmTransport.disallowListener()</a>	Cancel certified delivery to all listeners at a specific correspondent. Deny subsequent certified delivery registration requests from those listeners.
<a href="#">TibrvCmTransport.expireMessages()</a>	Mark specified outbound CM messages as expired.
<a href="#">TibrvCmTransport.getDefaultTimeLimit()</a>	Get the default message time limit for all outbound certified messages from a transport.
<a href="#">TibrvCmTransport.getLedgerName()</a>	Extract the ledger name of a certified delivery transport.
<a href="#">TibrvCmTransport.getName()</a>	Extract the correspondent name of a certified delivery transport or distributed queue member.
<a href="#">TibrvCmTransport.getRequestOld()</a>	Extract the request old messages flag of a certified delivery transport.

Method	Description
<a href="#">TibrvCmTransport.getSyncLedger()</a>	Extract the sync ledger flag of a certified delivery transport.
<a href="#">TibrvCmTransport.getTransport()</a>	Extract the transport employed by a certified delivery transport or a distributed queue member.
<a href="#">TibrvCmTransport.removeListener()</a>	Unregister a specific listener at a specific correspondent, and free associated storage in the sender's ledger.
<a href="#">TibrvCmTransport.removeSendState()</a>	Reclaim ledger space from obsolete subjects.
<a href="#">TibrvCmTransport.reviewLedger()</a>	Query the ledger for stored items related to a subject name.
<a href="#">TibrvCmTransport.send()</a>	Send a labeled message.
<a href="#">TibrvCmTransport.sendReply()</a>	Send a labeled reply message.
<a href="#">TibrvCmTransport.sendRequest()</a>	Send a labeled request message and wait for a reply.
<a href="#">TibrvCmTransport.setDefaultTimeLimit()</a>	Set the default message time limit for all outbound certified messages from a transport.
<a href="#">TibrvCmTransport.setPublisherInactivityDiscardInterval()</a>	Set a time limit after which a listening CM transport can discard state for inactive CM senders.
<a href="#">TibrvCmTransport.syncLedger()</a>	Synchronize the ledger to its storage medium.

## Inherited Methods

[TibrvTransport.createInbox\(\)](#)

[TibrvTransport.destroy\(\)](#)

[TibrvTransport.isValid\(\)](#)

[TibrvTransport.send\(\)](#)

[TibrvTransport.sendReply\(\)](#)

[TibrvTransport.sendRequest\(\)](#)

[java.lang.Object.equals](#)

[java.lang.Object.getClass](#)

[java.lang.Object.hashCode](#)

[java.lang.Object.notify](#)

[java.lang.Object.notifyAll](#)

[java.lang.Object.toString](#)

[java.lang.Object.wait](#)

Field	Description
DEFAULT_TIME_LIMIT	<p>The default time limit (in seconds) for certified messages sent on this transport.</p> <p>The initial value is zero, a special value indicating that messages do not expire.</p> <p>Programs can change this value using <a href="#">TibrvCmTransport.setDefaultTimeLimit()</a>.</p>

## Related Classes

[TibrvTransport](#)

[TibrvRvdTransport](#)

[TibrvCmTransport](#)

[TibrvCmQueueTransport](#)

# TibrvCmTransport()

*Method*

## Declaration

```
TibrvCmTransport(  
    TibrvRvdTransport transport)  
    throws TibrvException  
TibrvCmTransport(  
    TibrvRvdTransport transport,  
    java.lang.String cmName,  
    boolean requestOld)  
    throws TibrvException  
TibrvCmTransport(  
    TibrvRvdTransport transport,  
    java.lang.String cmName,  
    boolean requestOld,  
    java.lang.String ledgerName,  
    boolean syncLedger)  
    throws TibrvException  
TibrvCmTransport(  
    TibrvRvdTransport transport,  
    java.lang.String cmName,  
    boolean requestOld,  
    java.lang.String ledgerName,  
    boolean syncLedger)  
    throws TibrvException
```

## Purpose

Create a transport for certified delivery.

## Remarks

The new certified delivery transport must employ a valid transport for network communications.

The certified delivery transport remains valid until the program explicitly destroys it.

Parameter	Description
transport	<p>The new <a href="#">TibrvCmTransport</a> employs this transport object for network communications.</p> <p>This object must be a <a href="#">TibrvRvdTransport</a>.</p> <p>Destroying the <a href="#">TibrvCmTransport</a> does not affect this <a href="#">TibrvRvdTransport</a> object.</p>
cmName	<p>Bind this reusable name to the new <a href="#">TibrvCmTransport</a>, so the <a href="#">TibrvCmTransport</a> represents a persistent correspondent with this name.</p> <p>If non-null, the name must conform to the syntax rules for Rendezvous subject names. It cannot begin with reserved tokens. It cannot be a non-reusable name generated by another call to <a href="#">TibrvCmTransport()</a>. It cannot be the empty string.</p> <p>If omitted or null, then <a href="#">TibrvCmTransport()</a> generates a unique, non-reusable name for the duration of the transport.</p> <p>For more information, see <a href="#">Name</a>.</p>
requestOld	<p>This parameter indicates whether a persistent correspondent requires delivery of messages sent to a previous certified delivery transport with the same name, for which delivery was not confirmed. Its value affects the behavior of other CM sending transports.</p> <p>If this parameter is true <i>and</i> cmName is non-null, then the new <a href="#">TibrvCmTransport</a> requires certified senders to retain unacknowledged messages sent to this persistent correspondent. When the new <a href="#">TibrvCmTransport</a> begins listening to the appropriate subjects, the senders can complete delivery. (It is an error to supply true when cmName is null.)</p> <p>If this parameter is false (or omitted), then the new <a href="#">TibrvCmTransport</a> does not require certified senders to retain unacknowledged messages. Certified senders may delete those messages from their ledgers.</p>
ledgerName	<p>If this argument is non-null, then the new <a href="#">TibrvCmTransport</a> uses a file-based ledger. The argument must represent a valid file name. Actual locations corresponding to relative file names conform to operating system conventions. We strongly discourage using the empty string as a ledger file</p>



Parameter	Description
	<p>name.</p> <p>If omitted or null, then the new <a href="#">TibrvCmTransport</a> uses a process-based ledger.</p> <p>For more information, see <a href="#">Ledger File</a>.</p>
syncLedger	<p>If this argument is true, then operations that update the ledger file do not return until the changes are written to the storage medium.</p> <p>If this argument is false (or omitted), the operating system writes changes to the storage medium asynchronously.</p>

## Method Forms

With only a transport, create a transient correspondent, with a unique, non-reusable name. (Supplying null as the cmName has the same effect.)

All other parameters are optional, with default values when omitted.

## Name

If cmName is null, then [TibrvCmTransport\(\)](#) generates a unique, non-reusable name for the new certified delivery transport.

If cmName is non-null, then the new transport binds that name. A correspondent can persist beyond transport destruction only when it has *both* a reusable name *and* a file-based ledger.

For more information about the use of reusable names, see CM Correspondent Name in TIBCO Rendezvous Concepts, and Persistent Correspondents in TIBCO Rendezvous Concepts. For details of reusable name syntax, see Reusable Names in TIBCO Rendezvous Concepts.

## Ledger File

Every certified delivery transport stores the state of its certified communications in a ledger.

If `ledgerFile` is null, then the new transport stores its ledger exclusively in process-based storage. When you destroy the transport or the process terminates, all information in the ledger is lost.

If `ledgerFile` specifies a valid file name, then the new transport uses that file for ledger storage. If the transport is destroyed or the process terminates with incomplete certified communications, the ledger file records that state. When a new transport binds the same reusable name, it reads the ledger file and continues certified communications from the state stored in the file.

Even though a transport uses a ledger file, it may sometimes replicate parts of the ledger in process-based storage for efficiency; however, programmers cannot rely on this replication.

The `syncLedger` parameter determines whether writing to the ledger file is a synchronous operation:

- To specify synchronous writing, supply `true`. Each time Rendezvous software writes a ledger item, the call does not return until the data is safely stored in the storage medium.
- To specify asynchronous writing (the default), supply `false`. Certified delivery calls may return before the data is safely stored in the storage medium, which results in greater speed at the cost of certainty. The ledger file might not accurately reflect program state in cases of hardware or operating system kernel failure (but it is accurate in cases of sudden program failure). Despite this small risk, we strongly recommend this option for maximum performance.

A program that uses an asynchronous ledger file can explicitly synchronize it by calling [TibrvCmTransport.syncLedger\(\)](#).

Destroying a transport with a file-based ledger always leaves the ledger file intact; it neither erases nor removes a ledger file.

The ledger file must reside on the same host computer as the program that uses it.

## See Also

[TibrvCmTransport.destroy\(\)](#)

# TibrvCmTransport.addListener()

*Method*

## Declaration

```
void addListener(  
    java.lang.String cmName,  
    java.lang.String subject)  
throws TibrvException
```

## Purpose

Pre-register an anticipated listener.

## Remarks

Some sending programs can anticipate requests for certified delivery—even before the listening programs actually register. In such situations, the sending transport can pre-register listeners, so Rendezvous software begins storing outbound messages in the sender's ledger; when the listener requests certified delivery, it receives the backlogged messages.

If the correspondent with this `cmName` already receives certified delivery of this subject from this sender transport, then [TibrvCmTransport.addListener\(\)](#) has no effect.

If the correspondent with this `cmName` is disallowed, [TibrvCmTransport.addListener\(\)](#) returns an exception with status code [TibrvStatus.NOT\\_PERMITTED](#). You can call [TibrvCmTransport.allowListener\(\)](#) to supersede the effect of a prior call to [TibrvCmTransport.disallowListener\(\)](#); then call [TibrvCmTransport.addListener\(\)](#) again.

It is not sufficient for a sender to use this method to anticipate listeners; the anticipated listening programs must also require old messages when creating certified delivery transports.

Parameter	Description
<code>cmName</code>	Anticipate a listener from a correspondent with this

Parameter	Description
	reusable name.
subject	Anticipate a listener for this subject. Wildcard subjects are illegal.

## See Also

### Name

[TibrvCmTransport.allowListener\(\)](#)

[TibrvCmTransport.disallowListener\(\)](#)

[TibrvCmTransport.removeListener\(\)](#)

[Anticipating a Listener](#) in TIBCO Rendezvous Concepts

# TibrvCmTransport.allowListener()

*Method*

## Declaration

```
void allowListener(  
    java.lang.String cmName)  
    throws TibrvException
```

## Purpose

Invite the named receiver to reinstate certified delivery for its listeners, superseding the effect of any previous *disallow* calls.

## Remarks

Upon receiving the invitation to reinstate certified delivery, Rendezvous software at the listening program automatically sends new registration requests. The sending program accepts these requests, restoring certified delivery.

Parameter	Description
cmName	Accept requests for certified delivery to listeners at the transport with this correspondent name.

## See Also

[Name](#)

[TibrvCmTransport.disallowListener\(\)](#)

Disallowing Certified Delivery in TIBCO Rendezvous Concepts

# TibrvCmTransport.destroy()

*Method*

## Declaration

```
void destroy()  
void destroyEx()
```

## Purpose

Destroy a certified delivery transport.

## Remarks

Destroying a certified delivery transport with a file-based ledger always leaves the ledger file intact; it neither erases nor removes a ledger file.

## Distributed Queue

To destroy a distributed queue transport, call `destroyEx()`. With the ordinary `destroy` call, the distributed queue can lose reliable (non-certified) task messages before they are processed. In contrast, `destroyEx()` blocks until previously assigned tasks are complete; then it destroys the transport and returns. The distributed queue needs the listeners, queues and dispatchers (associated with the transport) to remain operational—programs must wait until after the transport has been completely destroyed before destroying these associated objects.

## Restrictions

`destroyEx()` is available in both the JNI preferred implementation and the JNI backward compatibility implementation. It is not available in the pure Java implementation.

## See Also

[TibrvCmTransport\(\)](#)

# TibrvCmTransport.disallowListener()

*Method*

## Declaration

```
void disallowListener(  
    java.lang.String cmName)  
    throws TibrvException
```

## Purpose

Cancel certified delivery to all listeners at a specific correspondent. Deny subsequent certified delivery registration requests from those listeners.

## Remarks

Disallowed listeners still receive subsequent messages from this sender, but delivery is not certified. In other words:

- The first labeled message causes the listener to initiate registration. Registration fails, and the listener discards that labeled message.
- The listener receives a [REGISTRATION.NOT\\_CERTIFIED](#) advisory, informing it that the sender has canceled certified delivery of all subjects.
- If the sender's ledger contains messages sent to the disallowed listener (for which this listener has not confirmed delivery), then Rendezvous software removes those ledger items, and does not attempt to redeliver those messages.
- Rendezvous software presents subsequent messages (from the canceling sender) to the listener without a sequence number, to indicate that delivery is not certified.

Senders can promptly revoke the acceptance of certified delivery by calling [TibrvCmTransport.disallowListener\(\)](#) within the callback method that processes the REGISTRATION.REQUEST advisory.

This method disallows a correspondent by name. If the correspondent terminates, and another process instance (with the same reusable name) takes its place, the new process is still disallowed by this sender.

To supersede the effect of [TibrvCmTransport.disallowListener\(\)](#), call [TibrvCmTransport.allowListener\(\)](#).

Parameter	Description
cmName	Cancel certified delivery to listeners of the transport with this name.

## See Also

[Name](#)

[TibrvCmTransport.allowListener\(\)](#)

Disallowing Certified Delivery in TIBCO Rendezvous Concepts



# TibrvCmTransport.expireMessages()

*Method*

## Declaration

```
void expireMessages(  
    String subject,  
    long  sequenceNumber)  
    throws TibrvException;
```

## Purpose

Mark specified outbound CM messages as expired.

## Remarks

This call checks the ledger for messages that match *both* the subject and sequence number criteria, and *immediately* marks them as expired.

Once a message has expired, the CM transport no longer attempts to redeliver it to registered listeners.

Rendezvous software presents each expired message to the sender in a DELIVERY.FAILED advisory. Each advisory includes all the fields of an expired message. (This call can cause many messages to expire simultaneously.)



### Warning

Use with extreme caution. This call exempts the expired messages from certified delivery semantics. It is appropriate only in very few situations.

For example, consider an application program in which an improperly formed CM message causes registered listeners to exit unexpectedly. When the listeners restart, the sender attempts to redeliver the offending message, which again causes the listeners to exit. To break this cycle, the sender can expire the offending message (along with all prior messages bearing the same subject).

Parameter	Description
subject	Mark messages with this subject.  Wild cards subjects are permitted, but must exactly reflect the send subject of the message. For example, if the program sends to A.* then you may expire messages with subject A.* (however, A.> does not resolve to match A.*).
sequenceNumber	Mark messages with sequence numbers <i>less than or equal</i> to this value.

## See Also

DELIVERY.FAILED in TIBCO Rendezvous Concepts

# TibrvCmTransport.getDefaultTimeLimit()

*Method*

## Declaration

```
double getDefaultTimeLimit()
```

## Purpose

Get the default message time limit for all outbound certified messages from a transport.

## Remarks

Every labeled message has a time limit, after which the sender no longer certifies delivery.

Sending programs can explicitly set the time limit on a message (see [TibrvCmMsg.setTimeLimit\(\)](#)). If a time limit is not already set for the outbound message, the transport sets it to the transport's default time limit (extractable with this method); if this default is not set for the transport (nor for the message), the default time limit is zero (no time limit).

Time limits represent the minimum time that certified delivery is in effect.

## See Also

[TibrvCmTransport.setDefaultTimeLimit\(\)](#)

[TibrvCmMsg.setTimeLimit\(\)](#)

# TibrvCmTransport.getLedgerName()

*Method*

## Declaration

```
java.lang.String getLedgerName()
```

## Purpose

Extract the ledger name of a certified delivery transport.

## Errors

The error code [TibrvStatus.ARG\\_CONFLICT](#) can indicate that the transport does not have a ledger file.

## See Also

[Ledger File](#)

[TibrvCmTransport\(\)](#)

# TibrvCmTransport.getName()

*Method*

## Declaration

```
java.lang.String getName()
```

## Purpose

Extract the correspondent name of a certified delivery transport or distributed queue member.

## See Also

[Name](#)

[TibrvCmTransport\(\)](#)

[TibrvCmQueueTransport\(\)](#)

# TibrvCmTransport.getRequestOld()

*Method*

## Declaration

```
boolean getRequestOld()
```

## Purpose

Extract the request old messages flag of a certified delivery transport.

## See Also

[requestOld](#)

[TibrvCmTransport\(\)](#)

# TibrvCmTransport.getSyncLedger()

*Method*

## Declaration

```
boolean getSyncLedger()
```

## Purpose

Extract the sync ledger flag of a certified delivery transport.

## Errors

The error code [TibrvStatus.ARG\\_CONFLICT](#) can indicate that the transport does not have a ledger file.

## See Also

[Ledger File](#)

[TibrvCmTransport\(\)](#)

# TibrvCmTransport.getTransport()

*Method*

## Declaration

```
TibrvRvdTransport getTransport()
```

## Purpose

Extract the transport employed by a certified delivery transport or a distributed queue member.

## See Also

[TibrvTransport](#)

[TibrvRvdTransport](#)

[TibrvCmTransport\(\)](#)

[TibrvCmQueueTransport\(\)](#)



# TibrvCmTransport.removeListener()

*Method*

## Declaration

```
void removeListener(  
    java.lang.String  cmName,  
    java.lang.String  subject)  
throws TibrvException
```

## Purpose

Unregister a specific listener at a specific correspondent, and free associated storage in the sender's ledger.

## Remarks

This method cancels certified delivery of the specific subject to the correspondent with this name. The listening correspondent may subsequently re-register for certified delivery of the subject. (In contrast, [TibrvCmTransport.disallowListener\(\)](#) cancels certified delivery of *all* subjects to the correspondent, *and* prohibits re-registration.)

Senders can call this method when the ledger item for a listening correspondent has grown very large. Such growth indicates that the listener is not confirming delivery, and may have terminated. Removing the listener reduces the ledger size by deleting messages stored for the listener.

When a sending program calls this method, certified delivery software in the sender behaves as if the listener had closed the endpoint for the subject. The sending program deletes from its ledger all information about delivery of the subject to the correspondent with this cmName. The sending program receives a REGISTRATION.CLOSED advisory, to trigger any operations in the callback method for the advisory.

If the listening correspondent is available (running and reachable), it receives a REGISTRATION.NOT\_CERTIFIED advisory, informing it that the sender no longer certifies delivery of the subject.

If the correspondent with this name does not receive certified delivery of the subject from this sender [TibrvCmTransport](#), then [TibrvCmTransport.removeListener\(\)](#) throws an exception with the status code [TibrvStatus.INVALID\\_ARG](#).

Parameter	Description
cmName	Cancel certified delivery of the subject to listeners of this correspondent.
subject	Cancel certified delivery of this subject to the named listener. Wildcard subjects are illegal.

## See Also

[Name](#)

[TibrvCmTransport.addListener\(\)](#)

[TibrvCmTransport.disallowListener\(\)](#)

Canceling Certified Delivery in TIBCO Rendezvous Concepts

# TibrvCmTransport.removeSendState()

*Method*

## Declaration

```
void removeSendState(  
    java.lang.String subject)  
    throws TibrvException
```

## Purpose

Reclaim ledger space from obsolete subjects.

## Background

In some programs subject names are useful only for a limited time; after that time, they are never used again. For example, consider a server program that sends certified reply messages to client inbox names; it only sends one reply message to each inbox, and after delivery is confirmed and complete, that inbox name is obsolete. Nonetheless, a record for that inbox name remains in the server's ledger.

As such obsolete records accumulate, the ledger size grows. To counteract this growth, programs can use this method to discard obsolete subject records from the ledger.

The DELIVERY.COMPLETE advisory is a good opportunity to clear the send state of an obsolete subject. Another strategy is to review the ledger periodically, sweeping to detect and remove all obsolete subjects.



### Warning

Do not use this method to clear subjects that are still in use.

Parameter	Description
subject	Remove send state for this obsolete subject.

## Remarks

As a side-effect, this method resets the sequence numbering for the subject, so the next message sent on the subject would be number 1. In proper usage, this side-effect is never detected, since obsolete subjects are truly obsolete.

## See Also

[TibrvCmTransport.reviewLedger\(\)](#)

[TibrvCmTransport.send\(\)](#)

DELIVERY.COMPLETE in TIBCO Rendezvous Concepts

# TibrvCmTransport.reviewLedger()

*Method*

## Declaration

```
void reviewLedger(  
    TibrvCmReviewCallback callback,  
    java.lang.String      subject,  
    java.lang.Object      closure)  
throws TibrvException
```

## Purpose

Query the ledger for stored items related to a subject name.

## Remarks

The callback method receives one message for each matching subject of outbound messages stored in the ledger. For example, when FOO.\* is the subject, [TibrvCmTransport.reviewLedger\(\)](#) calls its callback method separately for each matching subject—once for FOO.BAR, once for FOO.BAZ, and once for FOO.BOX.

However, if the callback method returns non-null, then [TibrvCmTransport.reviewLedger\(\)](#) returns immediately.

If the ledger does not contain any matching items, [TibrvCmTransport.reviewLedger\(\)](#) returns normally without calling the callback method.

For information about the content and format of the callback messages, see [TibrvCmReviewCallback.onLedgerMsg\(\)](#).

Parameter	Description
callback	This object receives the review messages.
subject	Query for items related to this subject name.

Parameter	Description
	If this subject contains wildcard characters (* or >), then review all items with matching subject names. The callback method receives a separate message for each matching subject in the ledger.
closure	Pass this closure data to the review callback method.

## See Also

[TibrvCmReviewCallback.onLedgerMsg\(\)](#)

# TibrvCmTransport.send()

*Method*

## Declaration

```
void send(  
    TibrvMsg msg)  
    throws TibrvException
```

## Purpose

Send a labeled message.

## Remarks

This method sends the message, along with its certified delivery protocol information: the correspondent name of the [TibrvCmTransport](#), a sequence number, and a time limit. The protocol information remains on the message within the sending program, and also travels with the message to all receiving programs.

Programs can explicitly set the message time limit; see [TibrvCmMsg.setTimeLimit\(\)](#). If a time limit is not already set for the outbound message, this method sets it to the transport's default time limit (see [TibrvCmTransport.setDefaultTimeLimit\(\)](#)); if that default is not set for the transport, the default time limit is zero (no time limit).

Parameter	Description
msg	Send this message.  Wildcard subjects are illegal.

## See Also

[TibrvCmTransport.sendReply\(\)](#)

[TibrvCmTransport.sendRequest\(\)](#)

[TibrvCmTransport.setDefaultTimeLimit\(\)](#)

[TibrvCmMsg.setTimeLimit\(\)](#)



# TibrvCmTransport.sendReply()

*Method*

## Declaration

```
void sendReply(  
    TibrvMsg replyMsg,  
    TibrvMsg requestMsg)  
throws TibrvException
```

## Purpose

Send a labeled reply message.

## Remarks

This convenience call extracts the reply subject of an inbound request message, and sends a labeled outbound reply message to that subject. In addition to the convenience, this call is marginally faster than using separate calls to extract the subject and send the reply.

This method can send a labeled reply to an ordinary message.

This method automatically registers the requesting CM transport, so the reply message is certified.

Parameter	Description
replyMsg	Send this <i>outbound</i> reply message.
requestMsg	<p>Send a reply to this <i>inbound</i> request message; extract its reply subject to use as the subject of the outbound reply message.</p> <p>If this message has a wild card reply subject, the method produces an error.</p>

**Warning**

Give special attention to the order of the arguments to this method. Reversing the inbound and outbound messages can cause an infinite loop, in which the program repeatedly resends the inbound message to itself (and all other recipients).

**See Also**

[TibrvCmTransport.send\(\)](#)

[TibrvCmTransport.sendRequest\(\)](#)

# TibrvCmTransport.sendRequest()

*Method*

## Declaration

```
TibrvMsg sendRequest(  
    TibrvMsg msg,  
    double timeout)  
throws TibrvException
```

## Purpose

Send a labeled request message and wait for a reply.

## Blocking can Stall Event Dispatch



### Warning

This call blocks all other activity on its program thread. If appropriate, programmers must ensure that other threads continue dispatching events on its queues.

Parameter	Description
msg	Send this request message.  Wildcard subjects are illegal.
timeout	Maximum time (in seconds) that this call can block while waiting for a reply.

## Remarks

Programs that receive and process the request message cannot determine that the sender has blocked until a reply arrives.

The sender and receiver must already have a certified delivery agreement, otherwise the request is not certified.

The request message must have a valid destination subject; see [TibrvMsg.setSendSubject\(\)](#).

A certified request does not necessarily imply a certified reply; the replying program determines the type of reply message that it sends.

## Operation

This method operates in several synchronous steps:

### Procedure

1. Create a [TibrvCmListener](#) that listens for messages on the reply subject of msg.
2. Label and send the outbound message.
3. Block until the listener receives a reply; if the time limit expires before a reply arrives, then return null. (The reply event uses a private queue that is not accessible to the program.)
4. Return the reply message as the value of the method call.

## See Also

[TibrvCmTransport.send\(\)](#)

[TibrvCmTransport.sendReply\(\)](#)

# TibrvCmTransport.setDefaultTimeLimit()

*Method*

## Declaration

```
void setDefaultTimeLimit(  
    double timeLimit)  
    throws TibrvException
```

## Purpose

Set the default message time limit for all outbound certified messages from a transport.

## Remarks

Every labeled message has a time limit, after which the sender no longer certifies delivery.

Sending programs can explicitly set the time limit on a message (see [TibrvCmMsg.setTimeLimit\(\)](#)). If a time limit is not already set for the outbound message, the transport sets it to the transport's default time limit (set with this method); if this default is not set for the transport, the default time limit is zero (no time limit).

Time limits represent the minimum time that certified delivery is in effect.

Parameter	Description
timeLimit	Use this time limit (in whole seconds). The time limit must be non-negative.

## See Also

[TibrvCmTransport.getDefaultTimeLimit\(\)](#)

[TibrvCmMsg.setTimeLimit\(\)](#)

# TibrvCmTransport.setPublisherInactivityDiscardInterval()

*Method*

## Declaration

```
void setPublisherInactivityDiscardInterval(  
    int timeout);  
throws TibrvException
```

## Purpose

Set a time limit after which a listening CM transport can discard state for inactive CM senders.

## Remarks

The timeout value limits the time that can elapse during which such a sender does not send a message. When the elapsed time exceeds this limit, the listening transport declares the sender inactive, and discards internal state corresponding to the sender.



### Warning

We discourage programmers from using this call except to solve a very specific problem, in which a long-running CM listener program accumulates state for a large number of obsolete CM senders with non-reusable names.

Before using this call, review every subject for which the CM transport has a listener; ensure that only CM senders with non-reusable names send to those subjects. (If senders with reusable names send messages to such subjects, the listening transport can discard their state, and incorrect behavior can result.)

Parameter	Description
timeout	Use this time limit (in whole seconds). The time limit must be non-negative.

# TibrvCmTransport.syncLedger()

*Method*

## Declaration

```
void syncLedger()  
    throws TibrvException
```

## Purpose

Synchronize the ledger to its storage medium.

## Remarks

When this method returns, the transport's current state is safely stored in the ledger file.

Transports that use synchronous ledger files need not call this method, since the current state is automatically written to the storage medium before returning. Transports that use process-based ledger storage need not call this method, since they have no ledger file.

## Errors

The error code [TibrvStatus.INVALID\\_ARG](#) can indicate that the transport does not have a ledger file.

## See Also

[Ledger File](#)

[TibrvCmTransport\(\)](#)

[TibrvCmTransport.getSyncLedger\(\)](#)



# TibrvCmReviewCallback

*Interface*

## Declaration

```
interface com.tibco.tibrv.TibrvCmReviewCallback
```

## Purpose

Process ledger review messages.

## Remarks

Implement this interface to process ledger review messages.

Method	Description
<a href="#">TibrvCmReviewCallback.onLedgerMsg()</a>	Programs define this method to process ledger review messages.

## See Also

[TibrvCmTransport.reviewLedger\(\)](#)

# TibrvCmReviewCallback.onLedgerMsg()

*Method*

## Declaration

```
boolean onLedgerMsg(  
    TibrvCmTransport cmTransport,  
    java.lang.String subject,  
    TibrvMsg msg,  
    java.lang.Object closure)
```

## Purpose

Programs define this method to process ledger review messages.

## Remarks

[TibrvCmTransport.reviewLedger\(\)](#) calls this callback method once for each matching subject stored in the ledger.

To continue reviewing the ledger, return false from this callback method. To stop reviewing the ledger, return true from this callback method; [TibrvCmTransport.reviewLedger\(\)](#) cancels the review and returns immediately.

Parameter	Description
cmTransport	This parameter receives the transport.
subject	This parameter receives the subject for this ledger item.
msg	This parameter receives a summary message describing the delivery status of messages in the ledger. The table below describes the fields of the summary message.
closure	This parameter receives closure data that the program supplied to <a href="#">TibrvCmTransport.reviewLedger()</a> .

Field Name	Description
subject	<p>The subject that this message summarizes.</p> <p>This field has datatype <a href="#">TibrvMsg.STRING</a>.</p>
seqno_last_sent	<p>The sequence number of the most recent message sent with this subject name.</p> <p>This field has datatype <a href="#">TibrvMsg.U64</a>.</p>
total_msgs	<p>The total number of messages stored at this subject name.</p> <p>This field has datatype <a href="#">TibrvMsg.U32</a>.</p>
total_size	<p>The total storage (in bytes) occupied by all messages with this subject name.</p> <p>If the ledger contains several messages with this subject name, then this field sums the storage space over all of them.</p> <p>This field has datatype <a href="#">TibrvMsg.I64</a>.</p>
listener	<p>Each summary message can contain one or more fields named listener. Each listener field contains a nested submessage with details about a single registered listener.</p> <p>This field has datatype <a href="#">TibrvMsg.MSG</a>.</p>
listener.name	<p>Within each listener submessage, the name field contains the name of the listener transport.</p> <p>This field has datatype <a href="#">TibrvMsg.STRING</a>.</p>
listener.last_confirmed	<p>Within each listener submessage, the last_confirmed field contains the sequence number of the last message for which the listener confirmed delivery.</p> <p>This field has datatype <a href="#">TibrvMsg.U64</a>.</p>

## See Also

[TibrvCmTransport.reviewLedger\(\)](#)

# TibrvCmMsg

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvCmMsg  
    extends java.lang.Object
```

## Purpose

Define methods to manipulate labeled messages.

## Remarks

Programs do not create instances of [TibrvCmMsg](#). Instead, programs use its static methods to get and set certified delivery information of [TibrvMsg](#) objects.

Method	Description
<a href="#">TibrvCmMsg.getSender()</a>	Extract the correspondent name of the sender from a certified message.
<a href="#">TibrvCmMsg.getSequence()</a>	Extract the sequence number from a certified message.
<a href="#">TibrvCmMsg.getTimeLimit()</a>	Extract the message time limit from a certified message.
<a href="#">TibrvCmMsg.setTimeLimit()</a>	Set the message time limit of a certified message.

### Inherited Methods

[java.lang.Object.equals](#)

[java.lang.Object.getClass](#)

## Inherited Methods

---

`java.lang.Object.hashCode`

`java.lang.Object.notify`

`java.lang.Object.notifyAll`

`java.lang.Object.toString`

`java.lang.Object.wait`

---

## See Also

[TibrvMsg](#)

# TibrvCmMsg.getSender()

*Method*

## Declaration

```
static final java.lang.String getSender(  
    TibrvMsg msg)  
    throws TibrvException
```

## Purpose

Extract the correspondent name of the sender from a certified message.

## Remarks

If the message is from a CM sender, then [TibrvCmMsg.getSender\(\)](#) yields a valid CM correspondent name. If the message is *not* from a CM sender, then [TibrvCmMsg.getSender\(\)](#) returns null.

Parameter	Description
<code>msg</code>	Extract the sender name from this message.

## See Also

[TibrvCmTransport\(\)](#)

[TibrvCmTransport.getName\(\)](#)

# TibrvCmMsg.getSequence()

*Method*

## Declaration

```
static final long getSequence(  
    TibrvMsg msg)  
    throws TibrvException
```

## Purpose

Extract the sequence number from a certified message.

## Remarks

Rendezvous certified delivery sending methods automatically generate positive sequence numbers for outbound labeled messages.

In receiving programs, zero is a special value, indicating that an inbound message is not certified.

Parameter	Description
msg	Extract the sequence number from this message.

## Exception

This method throws an exception to discriminate between certified messages (included in a certified delivery agreement) and other messages.

- If the message is from a CM sender, *and* the CM listener is registered for certified delivery with that sender, then `TibrvCmMsg.getSequence()` yields a valid sequence number.
- If the message is from a CM sender, but the listener is *not* registered for certified delivery, then `TibrvCmMsg.getSequence()` *in the context of a TibrvCmListener's callback method* throws an exception with the status code `TibrvStatus.NOT_FOUND`.

(In any other context, it returns the actual sequence number stored on the message.)

Notice that the first labeled message that a program receives on a subject might not be certified; that is, the sender has not registered a certified delivery agreement with the listener. If appropriate, the certified delivery library automatically requests that the sender register the listener for certified delivery. (See *Discovery and Registration for Certified Delivery in TIBCO Rendezvous Concepts*.)

A labeled but uncertified message can also result when the sender explicitly disallows or removes the listener.

- If the message is *not* from a CM sender, then [TibrvCmMsg.getSequence\(\)](#) (in any context) returns zero.

## Release 5 Interaction

In release 6 (and later) the sequence number is a 64-bit unsigned integer, while in older releases (5 and earlier) it is a 32-bit unsigned integer.

When 32-bit senders overflow the sequence number, behavior is undefined.

When 64-bit senders send sequence numbers greater than 32 bits, 32-bit receivers detect malformed label information, and process the message as an ordinary reliable message (uncertified and unlabeled).

## See Also

[TibrvCmTransport.send\(\)](#)



# TibrvCmMsg.getTimeLimit()

*Method*

## Declaration

```
static final double getTimeLimit(  
    TibrvMsg msg)  
    throws TibrvException
```

## Purpose

Extract the message time limit from a certified message.

## Remarks

Programs can explicitly set the message time limit (see [TibrvCmMsg.setTimeLimit\(\)](#)).

Zero is a special value, indicating no time limit.

If a time limit is not set for a message, this method returns zero. This situation can occur only for unsent outbound messages, and for inbound unlabeled messages.

Time limits represent the minimum time that certified delivery is in effect.

This value represents the total time limit of the message, *not* the time remaining.

Parameter	Description
<code>msg</code>	Extract the time limit from this message.

## See Also

[TibrvCmTransport.send\(\)](#)

[TibrvCmMsg.setTimeLimit\(\)](#)

# TibrvCmMsg.setTimeLimit()

*Method*

## Declaration

```
static final void setTimeLimit(  
    TibrvMsg msg,  
    double timeLimit)  
    throws TibrvException
```

## Purpose

Set the message time limit of a certified message.

## Remarks

Every labeled message has a time limit, after which the sender no longer certifies delivery.

Sending programs can explicitly set the message time limit using this method. If a time limit is not already set for the outbound message, [TibrvCmTransport.send\(\)](#) sets it to the transport's default time limit (see [TibrvCmTransport.setDefaultTimeLimit\(\)](#)); if that default is not set for the transport, the default time limit is zero (no time limit).

Time limits represent the minimum time that certified delivery is in effect.

It is meaningless for receiving programs to call this method.

Parameter	Description
msg	Set the time limit of this message.
timeLimit	Use this time limit (in whole seconds) for the message. The time limit must be non-negative.

## See Also

[TibrvCmTransport.getDefaultTimeLimit\(\)](#)

[TibrvCmTransport.setDefaultTimeLimit\(\)](#)

[TibrvCmMsg.getTimeLimit\(\)](#)

# Distributed Queue

---

Programs can use distributed queues for *one of  $n$*  certified delivery to a group of worker processes.

A distributed queue is a group of [TibrvCmQueueTransport](#) objects, each in a separate process. From the outside, a distributed queue appears as though a single transport object; inside, the group members act in concert to process inbound task messages. Ordinary senders and CM senders can send task messages to the group. Notice that the senders are not group members, and do not do anything special to send messages to a group; rather, they send messages to ordinary subject names. Inside the group, the member acting as scheduler assigns each task message to exactly one of the other members (which act as workers); only that worker processes the task message. Each member uses CM listener objects to receive task messages.

Distributed queues depend upon the certified delivery methods and the fault tolerance methods.



## Note

We do not recommend sending messages across network boundaries to a distributed queue, nor distributing queue members across network boundaries. However, when crossing network boundaries in either of these ways, you must configure the Rendezvous routing daemons to exchange `_RVCM` and `_RVCMQ` administrative messages. For details, see *Distributed Queues in TIBCO Rendezvous Administration*.

## See Also

Distributed Queue in *TIBCO Rendezvous Concepts*

# TibrvCmQueueTransport

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvCmQueueTransport  
    extends TibrvCmTransport
```

## Purpose

Coordinate a distributed queue for *one-of-n* delivery.

## Remarks

Each [TibrvCmQueueTransport](#) object employs a [TibrvTransport](#) for network communications. The [TibrvCmQueueTransport](#) adds the accounting and coordination mechanisms needed for one-of-n delivery.

Several [TibrvCmQueueTransport](#) objects can employ one [TibrvTransport](#), which also remains available for its own ordinary listeners and for sending ordinary messages.

Programs must explicitly destroy each [TibrvCmQueueTransport](#) object. Destroying a [TibrvCmQueueTransport](#) invalidates any certified listeners using that transport (while preserving their certified delivery agreements).

Whether explicitly or implicitly, programs must destroy instances of this class. Rendezvous software keeps internal references to these objects, so the Java garbage collector does not delete them automatically.

All members of a distributed queue must listen to exactly the same set of subjects. See Enforcing Identical Subscriptions in TIBCO Rendezvous Concepts.

Scheduler recovery and task rescheduling are available only when the task message is a certified message (that is, a certified delivery agreement is in effect between the task sender and the distributed queue transport scheduler).

## Disabled Methods

Although [TibrvCmQueueTransport](#) is a subclass of [TibrvCmTransport](#), all methods related to sending messages are disabled in [TibrvCmQueueTransport](#). These disabled methods throw an `IllegalStateException`; for a list, see [Disabled Methods](#). See also Certified Delivery Behavior in Queue Members in TIBCO Rendezvous Concepts.

Method	Description
<a href="#">TibrvCmQueueTransport()</a>	Create a transport as a distributed queue member.
<a href="#">TibrvCmQueueTransport.destroy()</a>	Destroy a distributed queue member object.
<a href="#">TibrvCmQueueTransport.getCompleteTime()</a>	Extract the worker complete time limit of a distributed queue member.
<a href="#">TibrvCmQueueTransport.getUnassignedMessageCount()</a>	Extract the number of unassigned task messages from a distributed queue transport.
<a href="#">TibrvCmQueueTransport.getWorkerWeight()</a>	Extract the worker weight of a distributed queue member.
<a href="#">TibrvCmQueueTransport.getWorkerTasks()</a>	Extract the worker task capacity of a distributed queue member.
<a href="#">TibrvCmQueueTransport.setCompleteTime()</a>	Set the worker complete time limit of a distributed queue member.

Method	Description
<a href="#">TibrvCmQueueTransport.setTaskBacklogLimit...()</a>	Set the scheduler task queue limits of a distributed queue transport.
<a href="#">TibrvCmQueueTransport.setWorkerWeight()</a>	Set the worker weight of a distributed queue member.
<a href="#">TibrvCmQueueTransport.setWorkerTasks()</a>	Set the worker task capacity of a distributed queue member.
<b>Inherited Methods</b>	
Legal Methods	<a href="#">TibrvCmTransport.getName()</a>
	<a href="#">TibrvCmTransport.getTransport()</a>
	<a href="#">TibrvTransport.isValid()</a> (override)
	<a href="#">java.lang.Object.equals</a>
	<a href="#">java.lang.Object.getClass</a>
	<a href="#">java.lang.Object.hashCode</a>
	<a href="#">java.lang.Object.notify</a>
	<a href="#">java.lang.Object.notifyAll</a>
	<a href="#">java.lang.Object.toString</a>
	<a href="#">java.lang.Object.wait</a>
Disabled Methods	<a href="#">TibrvCmTransport.addListener()</a>
	<a href="#">TibrvCmTransport.allowListener()</a>
	<a href="#">TibrvCmTransport.disallowListener()</a>
	<a href="#">TibrvCmTransport.getDefaultTimeLimit()</a>
	<a href="#">TibrvCmTransport.getLedgerName()</a>

## Inherited Methods

[TibrvCmTransport.getRequestOld\(\)](#)  
[TibrvCmTransport.getSyncLedger\(\)](#)  
[TibrvCmTransport.removeListener\(\)](#)  
[TibrvCmTransport.removeSendState\(\)](#)  
[TibrvCmTransport.reviewLedger\(\)](#)  
[TibrvCmTransport.send\(\)](#)  
[TibrvCmTransport.sendReply\(\)](#)  
[TibrvCmTransport.sendRequest\(\)](#)  
[TibrvCmTransport.setDefaultTimeLimit\(\)](#)  
[TibrvCmTransport.syncLedger\(\)](#)  
[TibrvTransport.createInbox\(\)](#)  
[TibrvTransport.send\(\)](#)  
[TibrvTransport.sendReply\(\)](#)  
[TibrvTransport.sendRequest\(\)](#)

Constant	Description
<code>TibrvCmQueueTransport.DEFAULT_COMPLETE_TIME</code>	static final double 0
<code>TibrvCmQueueTransport.DEFAULT_WORKER_WEIGHT</code>	static final int 1
<code>TibrvCmQueueTransport.DEFAULT_WORKER_TASKS</code>	static final int 1
<code>TibrvCmQueueTransport.DEFAULT_SCHEDULER_WEIGHT</code>	static final int 1
<code>TibrvCmQueueTransport.DEFAULT_SCHEDULER_HEARTBEAT</code>	static final double 1.0
<code>TibrvCmQueueTransport.DEFAULT_SCHEDULER_ACTIVATION</code>	static final double 3.5



## Related Classes

[TibrvTransport](#)

[TibrvRvdTransport](#)

[TibrvCmTransport](#)

# TibrvCmQueueTransport()

*Constructor*

## Declaration

```
TibrvCmQueueTransport(  
    TibrvRvdTransport transport,  
    java.lang.String  cmName)  
    throws TibrvException  
TibrvCmQueueTransport(  
    TibrvRvdTransport transport,  
    java.lang.String  cmName,  
    int               workerWeight,  
    int               workerTasks,  
    int               schedulerWeight,  
    double            schedulerHeartbeat,  
    double            schedulerActivation)  
    throws TibrvException
```

## Purpose


Create a transport as a distributed queue member.

## Remarks

The new [TibrvCmQueueTransport](#) must employ a valid [TibrvRvdTransport](#) for network communications.

Parameter	Description
transport	<p>The new <a href="#">TibrvCmQueueTransport</a> employs this <a href="#">TibrvRvdTransport</a> object for network communications.</p> <p>Destroying the <a href="#">TibrvCmQueueTransport</a> does not affect this transport.</p>
cmName	Bind this reusable name to the new transport object, which

Parameter	Description
	<p>becomes a member of the distributed queue with this name.</p> <p>The name must be non-null, and conform to the syntax rules for Rendezvous subject names. It cannot begin with reserved tokens. It cannot be a non-reusable name generated by a call to <a href="#">TibrvCmTransport()</a>. It cannot be the empty string.</p> <p>For more information, see <a href="#">Reusable Names</a> in TIBCO Rendezvous Concepts.</p>
workerWeight	<p>When the scheduler receives a task, it assigns the task to the available worker with the greatest worker weight.</p> <p>A worker is considered available unless either of these conditions are true:</p> <ul style="list-style-type: none"> <li>• The pending tasks assigned to the worker member exceed its task capacity.</li> <li>• The worker is also the scheduler. (The scheduler assigns tasks to its own worker role only when no other workers are available.)</li> </ul> <p>When omitted, the default value is 1.</p>
workerTasks	<p>Task capacity is the maximum number of tasks that a worker can accept. When the number of accepted tasks reaches this maximum, the worker cannot accept additional tasks until it completes one or more of them.</p> <p>When the scheduler receives a task, it assigns the task to the worker with the greatest worker weight—unless the pending tasks assigned to that worker exceed its task capacity. When the preferred worker has too many tasks, the scheduler assigns the new inbound task to the worker with the next greatest worker weight.</p> <p>The value must be a non-negative integer. When omitted, the default value is 1.</p> <p>Zero is a special value, indicating that this distributed queue</p>

Parameter	Description
	<p>member is a dedicated scheduler (that is, it never accepts tasks).</p> <p> <b>Warning</b></p> <p>Tuning task capacity to compensate for communication time lag is more complicated than it might seem. Before setting this value to anything other than 1, see <a href="#">Task Capacity</a> in TIBCO Rendezvous Concepts.</p>
schedulerWeight	<p>Weight represents the ability of this member to fulfill the role of scheduler, relative to other members with the same name. Cooperating members use relative scheduler weight values to elect one member as the scheduler; members with higher scheduler weight take precedence.</p> <p>When omitted, the default value is 1.</p> <p>Acceptable values range from 0 to 65535. Zero is a special value, indicating that the member can never be the scheduler. For more information, see <a href="#">Rank and Weight</a> in TIBCO Rendezvous Concepts.</p>
schedulerHeartbeat	<p>The scheduler sends heartbeat messages at this interval (in seconds).</p> <p>All <a href="#">TibrvCmQueueTransport</a> objects with the same name must specify the same value for this parameter. The value must be strictly positive. To determine the correct value, see <a href="#">Step 4: Choose the Intervals</a> in TIBCO Rendezvous Concepts.</p> <p>When omitted, the default value is 1.0.</p>
schedulerActivation	<p>When the heartbeat signal from the scheduler has been silent for this interval (in seconds), the cooperating member with the greatest scheduler weight takes its place as the new scheduler.</p> <p>All <a href="#">TibrvCmQueueTransport</a> objects with the same name must specify the same value for this parameter. The value must be strictly positive. To determine the correct value, see <a href="#">Step 4: Choose the Intervals</a> in TIBCO Rendezvous Concepts.</p>

Parameter	Description
	When omitted, the default value is 3.5.

## See Also

[TibrvCmQueueTransport.destroy\(\)](#)

[Distributed Queue](#), in TIBCO Rendezvous Concepts

# TibrvCmQueueTransport.destroy()

*Method*

## Declaration

```
void destroy()
```

## Purpose

Destroy a distributed queue member object.

## Remarks

Destroying a [TibrvCmQueueTransport](#) object removes the program from the distributed queue group.

## See Also

[TibrvCmQueueTransport\(\)](#)

# TibrvCmQueueTransport.getCompleteTime()

*Method*

## Declaration

```
double getCompleteTime()
```

## Purpose

Extract the worker complete time limit of a distributed queue member.

## See Also

Distributed Queue, in TIBCO Rendezvous Concepts

[TibrvCmQueueTransport.setCompleteTime\(\)](#)

# TibrvCmQueueTransport.getUnassignedMessageCount()

*Method*

## Declaration

```
int getUnassignedMessageCount()  
throws TibrvException
```

## Purpose

Extract the number of unassigned task messages from a distributed queue transport.

## Remarks

An unassigned task message is a message received by the scheduler, but not yet assigned to any worker in the distributed queue.

This call produces a valid count only within a scheduler process. Within a worker process, this call always produces zero.



# TibrvCmQueueTransport.getWorkerWeight()

*Method*

## Declaration

```
int getWorkerWeight()
```

## Purpose

Extract the worker weight of a distributed queue member.

## See Also

Distributed Queue, in TIBCO Rendezvous Concepts

[TibrvCmQueueTransport\(\)](#)

[TibrvCmQueueTransport.setWorkerWeight\(\)](#)

# TibrvCmQueueTransport.getWorkerTasks()

*Method*

## Declaration

```
int getWorkerTasks()
```

## Purpose

Extract the worker task capacity of a distributed queue member.

## See Also

Distributed Queue, in TIBCO Rendezvous Concepts

[TibrvCmQueueTransport\(\)](#)

[TibrvCmQueueTransport.setWorkerTasks\(\)](#)

# TibrvCmQueueTransport.setCompleteTime()

*Method*

## Declaration

```
void setCompleteTime(  
    double completeTime)  
    throws TibrvException
```

## Purpose

Set the worker complete time limit of a distributed queue member.

## Remarks

If the complete time is non-zero, the scheduler waits for a worker member to complete an assigned task. If the complete time elapses before the scheduler receives completion from the worker member, the scheduler reassigns the task to another worker member.

Zero is a special value, which specifies no limit on the completion time—that is, the scheduler does not set a timer, and does not reassign tasks when task completion is lacking. All members implicitly begin with a default complete time value of zero; programs can change this parameter using this method.

Parameter	Description
completeTime	Use this complete time (in seconds). The time must be non-negative.

## See Also

Distributed Queue, in TIBCO Rendezvous Concepts

[TibrvCmQueueTransport.getCompleteTime\(\)](#)

# TibrvCmQueueTransport.setTaskBacklogLimit...()

*Method*

## Declaration

```
void setTaskBacklogLimitInBytes(  
    int byteLimit)  
    throws TibrvException  
void setTaskBacklogLimitInMessages(  
    int msgLimit)  
    throws TibrvException
```

## Purpose

Set the scheduler task queue limits of a distributed queue transport.

## Remarks

The scheduler stores tasks in a queue. These properties limit the maximum size of that queue—by number of bytes or number of messages (or both). When no value is set for these properties, the default is no limit.

When the task messages in the queue exceed either of these limits, Rendezvous software deletes new inbound task messages.

Programs may call each of these methods at most once. The calls must occur before the transport assumes the scheduler role; after a transport acts as a scheduler, these values are fixed, and subsequent attempts to change them throw an exception with status code [TibrvStatus.NOT\\_PERMITTED](#).

Parameter	Description
byteLimit	Use this size limit (in bytes).  Zero is a special value, indicating no size limit.

Parameter	Description
msgLimit	Use this message limit (number of messages).  Zero is a special value, indicating no limit on the number of messages.

## See Also

Distributed Queue, in TIBCO Rendezvous Concepts

# TibrvCmQueueTransport.setWorkerWeight()

*Method*

## Declaration

```
void setWorkerWeight(  
    int workerWeight)  
    throws TibrvException
```

## Purpose

Set the worker weight of a distributed queue member.

## Remarks

Relative worker weights assist the scheduler in assigning tasks. When the scheduler receives a task, it assigns the task to the available worker with the greatest worker weight.

The default worker weight is 1; programs can set this parameter at creation using [TibrvCmQueueTransport\(\)](#), or change it dynamically using this method.

Parameter	Description
workerWeight	Use this worker weight.

## See Also

Distributed Queue in TIBCO Rendezvous Concepts

[TibrvCmQueueTransport\(\)](#)

[TibrvCmQueueTransport.getWorkerWeight\(\)](#)

# TibrvCmQueueTransport.setWorkerTasks()

*Method*

## Declaration

```
void setWorkerTasks(  
    int workerTasks)  
    throws TibrvException
```

## Purpose

Set the worker task capacity of a distributed queue member.

## Remarks

Task capacity is the maximum number of tasks that a worker can accept. When the number of accepted tasks reaches this maximum, the worker cannot accept additional tasks until it completes one or more of them.

When the scheduler receives a task, it assigns the task to the worker with the greatest worker weight—unless the pending tasks assigned to that worker exceed its task capacity. When the preferred worker has too many tasks, the scheduler assigns the new inbound task to the worker with the next greatest worker weight.

The default worker task capacity is 1.

Zero is a special value, indicating that this distributed queue member is a dedicated scheduler (that is, it never accepts tasks).



### Warning

Tuning task capacity to compensate for communication time lag is more complicated than it might seem. Before setting this value to anything other than 1, see Task Capacity in TIBCO Rendezvous Concepts.

Parameter	Description
workerTasks	Use this task capacity. The value must be a non-negative integer.

## See Also

Distributed Queue, in TIBCO Rendezvous Concepts

[TibrvCmQueueTransport\(\)](#)

[TibrvCmQueueTransport.getWorkerTasks\(\)](#)



# Exceptions and Errors

---

# TibrvException

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvException  
    extends java.lang.Exception
```

## Purpose

Rendezvous software throws exceptions of this class.

## Remarks

Rendezvous software also throws exceptions defined as part of the Java language.

Field	Description
error	An error or status code, indicating the reason for the exception; see <a href="#">TibrvStatus</a> .
internal	In some cases this field contains a Java exception, which can yield additional information. When no further information is available, this field is null.

Method	Description
<a href="#">TibrvException.printStackTrace()</a>	Print stack trace of this exception, and of the internal exception (if it is set).

### Inherited Methods

[java.lang.Throwable.fillInStackTrace](#)

## Inherited Methods

`java.lang.Throwable.getMessage`

`java.lang.Throwable.getMessage`

`java.lang.Throwable.printStackTrace (override)`

`java.lang.Throwable.toString (override)`

`java.lang.Object.equals`

`java.lang.Object.getClass`

`java.lang.Object.hashCode`

`java.lang.Object.notify`

`java.lang.Object.notifyAll`

`java.lang.Object.wait`

## See Also

[TibrvStatus](#)

# Throwable.printStackTrace()

*Method*

## Declaration

```
void printStackTrace()  
void printStackTrace(java.io.PrintWriter s)  
void printStackTrace(java.io.PrintStream s)
```

## Purpose

Print stack trace of this exception, and of the internal exception (if it is set).

## Overrides

java.lang.Throwable.printStackTrace

Parameter	Description
s	Print the stack trace to this object. When absent, print to the standard error stream.

# TibrvStatus

*Class*

## Declaration

```
class com.tibco.tibrv.TibrvStatus
```

## Purpose

Define status codes.

Status	Description
TibrvStatus.INIT_FAILURE	Cannot create the network transport.
TibrvStatus.INVALID_TRANSPORT	The transport has been destroyed, or is otherwise unusable.
TibrvStatus.INVALID_ARG	An argument is invalid. Check arguments other than messages, subject names, transports, events, queues and queue groups (which have separate status codes).
TibrvStatus.NOT_INITIALIZED	The method cannot run because the Rendezvous environment is not initialized (open).
TibrvStatus.ARG_CONFLICT	Two arguments that require a specific relation are in conflict. For example, the upper end of a numeric range is less than the lower end.
TibrvStatus.SERVICE_NOT_FOUND	Transport creation failed; cannot match the service name using <code>getservbyname()</code> .
TibrvStatus.NETWORK_NOT_FOUND	Transport creation failed; cannot match the network name using <code>getnetbyname()</code> .

Status	Description
TibrvStatus.DAEMON_NOT_FOUND	Transport creation failed; cannot match the daemon port number.
TibrvStatus.NO_MEMORY	The method could not allocate dynamic storage.
TibrvStatus.INVALID_SUBJECT	The method received a subject name with incorrect syntax.
TibrvStatus.DAEMON_NOT_CONNECTED	The Rendezvous daemon process (rvd) exited, or was never started. This status indicates that the program cannot start the daemon and connect to it.
TibrvStatus.VERSION_MISMATCH	The library, header files and Rendezvous daemon are incompatible.
TibrvStatus.SUBJECT_COLLISION	It is illegal to create two certified worker events on the same CM transport with overlapping subjects.
TibrvStatus.TIBRV_VC_NOT_CONNECTED	A virtual circuit terminal was once complete, but is now irreparably broken.
TibrvStatus.NOT_PERMITTED	<ol style="list-style-type: none"> <li>1. The program attempted an illegal operation.</li> <li>2. Cannot create ledger file.</li> </ol>
TibrvStatus.INVALID_NAME	The field name is too long; see <a href="#">Field Name Length</a> .
TibrvStatus.INVALID_TYPE	<ol style="list-style-type: none"> <li>1. The field type is not registered.</li> <li>2. Cannot update field to a type that differs from the existing field's type.</li> </ol>
TibrvStatus.INVALID_SIZE	The explicit size in the field does not match its explicit type.

Status	Description
TibrvStatus.INVALID_COUNT	The explicit field count does not match its explicit type.
TibrvStatus.INVALID_DATA	The program attempted to add data to a message field, but the datatype is not supported.
TibrvStatus.NOT_FOUND	Could not find the specified field in the message.
TibrvStatus.ID_IN_USE	Cannot add this field because its identifier is already present in the message; identifiers must be unique.
TibrvStatus.ID_CONFLICT	After field search by identifier fails, search by name succeeds, but the actual identifier in the field is non-null (so it does not match the identifier supplied).
TibrvStatus.CONVERSION_FAILED	Found the specified field, but could not convert it to the desired datatype.
TibrvStatus.RESERVED_HANDLER	The datatype handler number is reserved for Rendezvous internal datatype handlers.
TibrvStatus.ENCODER_FAILED	The program's datatype encoder failed.
TibrvStatus.DECODER_FAILED	The program's datatype decoder failed.
TibrvStatus.INVALID_MSG	The method received a message argument that is not a well-formed message.
TibrvStatus.INVALID_FIELD	The program supplied an invalid field as an argument.
TibrvStatus.INVALID_INSTANCE	The program supplied zero as the field instance number (the first instance is number 1).
TibrvStatus.CORRUPT_MSG	The method detected a corrupt message

Status	Description
	argument.
TibrvStatus.TIMEOUT	<p>A timed dispatch call returned without dispatching an event.</p> <p>A send request call returned without receiving a reply message.</p> <p>A virtual circuit terminal is not yet ready for use.</p>
TibrvStatus.INTR	Interrupted operation.
TibrvStatus.INVALID_DISPATCHABLE	The method received an event queue or queue group that has been destroyed, or is otherwise unusable.
TibrvStatus.INVALID_DISPATCHER	The dispatcher thread is invalid or has been destroyed.
TibrvStatus.INVALID_EVENT	The method received an event that has been destroyed, or is otherwise unusable.
TibrvStatus.INVALID_CALLBACK	The method received NULL instead of a callback method.
TibrvStatus.INVALID_QUEUE	The method received a queue that has been destroyed, or is otherwise unusable.
TibrvStatus.INVALID_QUEUE_GROUP	The method received a queue group that has been destroyed, or is otherwise unusable.
TibrvStatus.INVALID_TIME_INTERVAL	The method received a negative timer interval.
TibrvStatus.SOCKET_LIMIT	The operation failed because of an operating system socket limitation.
TibrvStatus.OS_ERROR	<a href="#">Tibrv.open()</a> encountered an operating system error.



Status	Description
<code>TibrvStatus.EOF</code>	End of file.
<code>TibrvStatus.INVALID_FILE</code>	<p>1. A certificate file or a ledger file is not recognizable as such.</p> <p>2. <code>TibrvSdContext.setUserCertWithKey()</code> or <code>TibrvSdContext.setUserCertWithKeyBin()</code> could not complete a certificate file operation; this status code can indicate either disk I/O failure, or invalid certificate data, or an incorrect password.</p>
<code>TibrvStatus.FILE_NOT_FOUND</code>	Rendezvous software could not find the specified file.
<code>TibrvStatus.IO_FAILED</code>	Cannot write to ledger file.
<code>TibrvStatus.NOT_FILE_OWNER</code>	<p>The program cannot open the specified file because another program owns it.</p> <p>For example, ledger files are associated with correspondent names.</p>
<code>TibrvStatus.TIBRV_IPM_ONLY</code>	The call is not available because the IPM library is not linked (that is, the call is available only when the IPM library is linked).
<b>Java-Specific Status Codes</b>	
<code>TibrvStatus.ERROR</code>	Default error when the error cannot be specified more precisely.
<code>TibrvStatus.LIBRARY_NOT_FOUND</code>	The JNI library is not present.
<code>TibrvStatus.LIBRARY_NOT_LOADED</code>	<code>SecurityException</code> while opening TIBCO Rendezvous machinery; the JNI library is required, but not properly loaded.
<code>TibrvStatus.WRONG_JAVA_ARCHIVE</code>	Attempted to open the native (JNI)

Status	Description
	implementation, but failed; opened the Java implementation instead.
TibrvStatus.INVALID_ENCODING	<a href="#">TibrvMsg.setStringEncoding()</a> received an invalid encoding name.

## See Also

[TibrvException](#)

# TibrvErrorCallback

*Interface*

## Declaration

```
interface com.tibco.tibrv.TibrvErrorCallback
```

## Purpose

Process asynchronous errors.

## Remarks

Programs can implement this interface (optional) to process asynchronous errors.

Method	Description
<a href="#">TibrvErrorCallback.onError()</a>	<a href="#">Process asynchronous errors.</a>

## See Also

[Tibrv.getErrorCallback\(\)](#)

[Tibrv.setErrorCallback\(\)](#)

# TibrvErrorCallback.onError()

*Method*

## Declaration

```
void onError(  
    java.lang.Object tibrvObject,  
    int errorCode,  
    java.lang.String message,  
    java.lang.Throwable internal)
```

## Purpose

Process asynchronous errors.

## Remarks

Rendezvous software calls this method in an asynchronous error situation: A subscription (listener) request failed. In this situation, the `tibrvObject` parameter receives the [TibrvListener](#) object representing the failed subscription. This situation is rare.

Programs need not respond to these errors. Programs that *do* respond to these errors usually inform the user of the problem.

Parameter	Description
<code>tibrvObject</code>	This parameter receives the object that is the locus of the error—a <a href="#">TibrvListener</a> .
<code>errorCode</code>	This parameter receives a status code indicating the error. See <a href="#">TibrvStatus</a> .
<code>message</code>	This parameter receives a printable string describing the error. In some cases, this message yields more information than the error code alone.

Parameter	Description
internal	In some cases this parameter receives a Java exception, which can yield additional information about the cause of the error. When no further information is available, this parameter receives null.

# Custom Datatypes

---

Programs can define custom datatypes by implementing interfaces that encode and decode the data. Decoder methods of [TibrvMsg](#) decode data from Rendezvous wire format representation into Java objects (for example, while extracting data from a message field). Encoder methods of [TibrvMsg](#) encode data into Rendezvous wire format representation from Java objects (for example, when sending a message).

This section describes these interfaces, and the methods that bind them to [TibrvMsg](#).

# TibrvMsg.getDecoder()

*Method*

## Declaration

```
static TibrvMsgDecoder getDecoder(  
    short userType)
```

## Purpose

Extract the decoder interface for a custom datatype.

## Remarks

If no decoder is registered for the datatype, return null.

Parameter	Description
userType	This type designator must be in the inclusive range [ <a href="#">TibrvMsg.USER_FIRST</a> , <a href="#">TibrvMsg.USER_LAST</a> ].

## See Also

[TibrvMsg.setHandlers\(\)](#)

[TibrvMsgDecoder](#)

# TibrvMsg.getEncoder()

*Method*

## Declaration

```
static TibrvMsgEncoder getEncoder(  
    short userType)
```

## Purpose

Extract the encoder interface for a custom datatype.

## Remarks

If no encoder is registered for the datatype, return null.

Parameter	Description
userType	This type designator must be in the inclusive range [ <a href="#">TibrvMsg.USER_FIRST</a> , <a href="#">TibrvMsg.USER_LAST</a> ].

## See Also

[TibrvMsg.setHandlers\(\)](#)

[TibrvMsgEncoder](#)



# TibrvMsg.setHandlers()

*Method*

## Declaration

```
static void setHandlers(  
    short      userType,  
    TibrvMsgEncoder encoder,  
    TibrvMsgDecoder decoder)
```

## Purpose

Define a custom datatype by registering its encoder and decoder interfaces.

## Remarks

The encoder and decoder must implement inverse operators. That is, when the encoder encodes a Java object as a byte array, and the decoder must decode the byte array to an identical Java object. Conversely, when the decoder decodes the byte array to a Java object, the encoder must encode the Java object as an identical byte array.

This method sets (or replaces) both interfaces. Supplying null for either the encoder or the decoder, removes that interface. Supplying null for both interfaces disables the custom type (by removing both interfaces).

Parameter	Description
userType	Define a custom datatype with this numeric designator.  This type designator must be in the inclusive range <a href="#">[TibrvMsg.USER_FIRST,TibrvMsg.USER_LAST]</a> .
encoder	Register this encoder for the type.
decoder	Register this decoder for the type.

## See Also

[TibrvMsgDecoder](#)

[TibrvMsgEncoder](#)

# TibrvMsgDecoder

*Interface*

## Declaration

```
interface com.tibco.tibrv.TibrvMsgDecoder
```

## Purpose

Decode custom datatypes from wire format into Java objects.

## Remarks

To define this interface, programs must implement its method.

Method	Description
<a href="#">TibrvMsgDecoder.decode()</a>	Decode data (of a custom datatype) from wire format into a Java object.

## See Also

[TibrvMsg.setHandlers\(\)](#).

[TibrvMsgEncoder](#)

# TibrvMsgDecoder.decode()

*Method*

## Declaration

```
java.lang.Object decode(  
    short type,  
    byte[] bytes)
```

## Purpose

Decode data (of a custom datatype) from wire format into a Java object.

## Remarks

When this method successfully decodes the data, it must return the decoding as a Java object. When this method cannot decode the data, it must return null.

Parameter	Description
type	Decode this custom datatype.
bytes	Decode the data contained in this byte array.  This argument cannot be null. However, it can be a byte array with length zero.

## See Also

[TibrvMsgEncoder.encode\(\)](#)

# TibrvMsgEncoder

*Interface*

## Declaration

```
interface com.tibco.tibrv.TibrvMsgEncoder
```

## Purpose

Encode Java objects as wire format custom datatypes.

## Remarks

To define this interface, programs must implement both of its methods.

Method	Description
<a href="#">TibrvMsgEncoder.canEncode()</a>	Test whether this encoder can encode the data as a particular wire format custom datatype.
<a href="#">TibrvMsgEncoder.encode()</a>	Encode a Java object as a wire format custom datatype.

## See Also

[TibrvMsg.setHandlers\(\)](#).

[TibrvMsgDecoder](#)

# TibrvMsgEncoder.canEncode()

*Method*

## Declaration

```
boolean canEncode(  
    short type,  
    java.lang.Object data)
```

## Purpose

Test whether this encoder can encode the data as a particular wire format custom datatype.

## Remarks

Before calling [TibrvMsgEncoder.encode\(\)](#), [TibrvMsg](#) first checks its applicability by calling this method. Whenever this method indicates that encoding is viable, [TibrvMsgEncoder.encode\(\)](#) must correctly encode the object.

This method must return true if the encoder can encode the data into the specified custom datatype; otherwise it must return false.

Parameter	Description
type	Test viability of encoding the data as an instance of this custom datatype.
data	Test viability of encoding this data.

## See Also

[TibrvMsgEncoder.encode\(\)](#)

# TibrvMsgEncoder.encode()

*Method*

## Declaration

```
byte[] encode(  
    short type,  
    java.lang.Object data)
```

## Purpose

Encode a Java object as a wire format custom datatype.

## Remarks

Before calling this method, [TibrvMsg](#) first checks its applicability by calling [TibrvMsgEncoder.canEncode\(\)](#). Whenever [TibrvMsgEncoder.canEncode\(\)](#) indicates that encoding is viable, this method must correctly encode the object.

When this method successfully encodes the data, it must return the encoding as a byte array; the byte array value can have length zero. When this method fails, it must return null.

Methods that call [TibrvMsgEncoder.encode\(\)](#) incorporate its byte array value directly into a [TibrvMsg](#) object.

Parameter	Description
type	Encode the data as an instance of this custom datatype.
data	Encode this data.

## See Also

[TibrvMsgDecoder.decode\(\)](#)

[TibrvMsgEncoder.canEncode\(\)](#)

# TIBCO Documentation and Support Services

---

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

## How to Access TIBCO Documentation

Documentation for TIBCO products is available on the [Product Documentation website](#), mainly in HTML and PDF formats.

The [Product Documentation website](#) is updated frequently and is more current than any other documentation included with the product.

## Product-Specific Documentation

The following documentation for this product is available on the [TIBCO Rendezvous® Product Documentation](#) page:

- *TIBCO Rendezvous® Concepts* - Read this book first. It contains basic information about Rendezvous components, principles of operation, programming constructs and techniques, advisory messages, and a glossary. All other books in the documentation set refer to concepts explained in this book.
- *TIBCO Rendezvous® Administration* - Begins with a checklist of action items for system and network administrators. This book describes the mechanics of TIBCO Rendezvous® licensing, network details, plus a chapter for each component of the TIBCO Rendezvous® software suite. Readers should have TIBCO Rendezvous Concepts at hand for reference.
- *TIBCO Rendezvous® Installation* - Includes step-by-step instructions for installing TIBCO Rendezvous® software on various operating system platforms.
- *TIBCO Rendezvous® C Reference* - Detailed descriptions of each data type and function in the TIBCO Rendezvous® C API. Readers should already be familiar with the C programming language, as well as the material in TIBCO Rendezvous Concepts.
- *TIBCO Rendezvous® C++ Reference* - Detailed descriptions of each class and method in the TIBCO Rendezvous® C++ API. The C++ API uses some data types and functions from the C API, so we recommend the TIBCO Rendezvous C Reference as an



additional resource. Readers should already be familiar with the C++ programming language, as well as the material in TIBCO Rendezvous Concepts.

- *TIBCO Rendezvous® .NET Reference* - Detailed descriptions of each class and method in the TIBCO Rendezvous® .NET interface. Readers should already be familiar with either C# or Visual Basic .NET, as well as the material in TIBCO Rendezvous Concepts.
- *TIBCO Rendezvous® Java Reference* - Detailed descriptions of each class and method in the TIBCO Rendezvous® Java language interface. Readers should already be familiar with the Java programming language, as well as the material in TIBCO Rendezvous Concepts.
- *TIBCO Rendezvous® Configuration Tools* - Detailed descriptions of each Java class and method in the TIBCO Rendezvous® configuration API, plus a command line tool that can generate and apply XML documents representing component configurations. Readers should already be familiar with the Java programming language, as well as the material in TIBCO Rendezvous Administration.
- *TIBCO Rendezvous® z/OS Installation and Configuration* - Information about TIBCO Rendezvous® for IBM z/OS systems regarding installation and maintenance. Some information may be also useful for application programmers.
- *TIBCO Rendezvous® Release Notes* - Lists new features, changes in functionality, deprecated features, migration and compatibility information, closed issues and known issues.

To directly access documentation for this product, double-click the following file:

`TIBCO_HOME/release_notes/TIB_rv_8.7.0_docinfo.html`

where `TIBCO_HOME` is the top-level directory in which TIBCO products are installed.

- On Windows, the default `TIBCO_HOME` is `C:\tibco`.
- On UNIX systems, the default `TIBCO_HOME` is `/opt/tibco`.

## How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our [product Support website](#).
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the our [product Support website](#). If you do not have a username, you can

request one by clicking **Register** on the website.

## How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the [TIBCO Ideas Portal](#). For a free registration, go to [TIBCO Community](#).

# Legal and Third-Party Notices

---

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of Cloud Software Group, Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, TIB, Information Bus, FTL, eFTL, Rendezvous, and LogLogic are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (<https://www.cloud.com/legal>) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: <https://scripts.sil.org/OFL>

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file

for the availability of a specific version of Cloud SG software on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at <https://www.tibco.com/patents>.

Copyright © 1997-2023. Cloud Software Group, Inc. All Rights Reserved.