TIBCO Rendezvous®

COM Reference

Software Release 8.4
February 2012

two-second advantage™

TIBCO®
The Power of Now®

Important Information

THIS 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 THIS SOFTWARE VERSION 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. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

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 "READ ME" FILES.

Contents

Figures

Tables

New or Modified Sections

Preface

TIBCO Rendezvous® is a messaging infrastructure product.

TIBCO is proud to announce the latest release of TIBCO Rendezvous®. This release is the latest in a long history of TIBCO products that leverage the power of the Information Bus® to enable truly event-driven IT environments. To find out more about how TIBCO Rendezvous and other TIBCO products are powered by TIB® technology, please visit us at www.tibco.com.

This manual describes the TIBCO Rendezvous API for COM and programming systems that use COM. It is part of the documentation set for Rendezvous Software Release 8.4.0.

Topics

Manual Organization

The organization of this book mirrors the underlying object structure of the Rendezvous COM component. Each chapter describes a group of closely related objects and the methods that pertain to them.

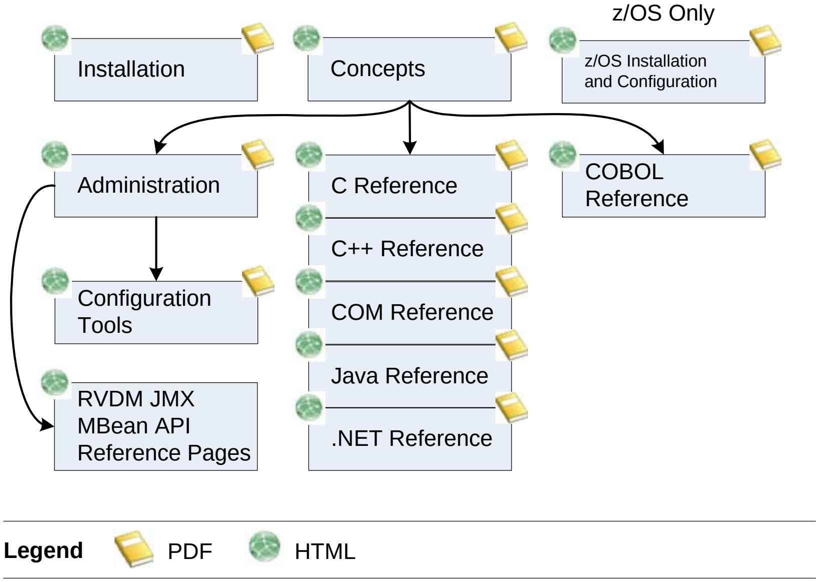
Within each object, methods appear in alphabetical order.

Related Documentation

This section lists documentation resources you may find useful.

TIBCO Rendezvous Documentation

The documentation road map shows the relationships between the books and online references in this product's documentation set.



The following documents form the Rendezvous documentation set:

- *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 C Reference*

  Detailed descriptions of each datatype and function in the 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 Rendezvous C++ API. The C++ API uses some datatypes 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 Java Reference*

  Detailed descriptions of each class and method in the 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 .NET Reference*

  Detailed descriptions of each class and method in the 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 COM Reference*

  Detailed descriptions of each class and method in the Rendezvous COM component. Readers should already be familiar with the programming environment that uses COM and OLE automation interfaces, as well as the material in *TIBCO Rendezvous Concepts*.

- *TIBCO Rendezvous Administration*

  Begins with a checklist of action items for system and network administrators. This book describes the mechanics of Rendezvous licensing, network details, plus a chapter for each component of the Rendezvous software suite. Readers should have *TIBCO Rendezvous Concepts* at hand for reference.

- *TIBCO Rendezvous Configuration Tools*

  Detailed descriptions of each Java class and method in the 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 Installation*

  Includes step-by-step instructions for installing Rendezvous software on various operating system platforms.

- *TIBCO Rendezvous Release Notes*

  Lists new features, changes in functionality, deprecated features, migration and compatibility information, closed issues and known issues.

Typographical Conventions

The following typographical conventions are used in this manual.

Table 1 General Typographical Conventions

Convention	Use
TIBCO_HOME *ENV_HOME* *TIBRV_HOME*	Many TIBCO products must be installed within the same home directory. This directory is referenced in documentation as *TIBCO_HOME*. The value of *TIBCO_HOME* depends on the operating system. For example, on Windows systems, the default value is `C:\tibco`. Other TIBCO products are installed into an *installation environment*. Incompatible products and multiple instances of the same product are installed into different installation environments. An environment home directory is referenced in documentation as *ENV_HOME*. The default value of *ENV_HOME* depends on the operating system. For example, on Windows systems the default value is `C:\tibco`. TIBCO Rendezvous installs into a version-specific directory inside *TIBCO_HOME*. This directory is referenced in documentation as *TIBRV_HOME*. The value of *TIBRV_HOME* depends on the operating system. For example on Windows systems, the default value is `C:\tibco\rv\8.4`.
`code font`	Code font identifies commands, code examples, filenames, pathnames, and output displayed in a command window. For example: Use `MyCommand` to start the foo process.
`bold code font`	Bold code font is used in the following ways: • In procedures, to indicate what a user types. For example: Type **`admin`**. • In large code samples, to indicate the parts of the sample that are of particular interest. • In command syntax, to indicate the default parameter for a command. For example, if no parameter is specified, `MyCommand` is enabled: `MyCommand [`**`enable`**` | disable]`

Table 1 General Typographical Conventions (Cont'd)

Convention	Use
italic font	Italic font is used in the following ways: • To indicate a document title. For example: See *TIBCO FTL Concepts*. • To introduce new terms For example: A portal page may contain several portlets. *Portlets* are mini-applications that run in a portal. • To indicate a variable in a command or code syntax that you must replace. For example: `MyCommand` *PathName*
Key combinations	Key name separated by a plus sign indicate keys pressed simultaneously. For example: Ctrl+C. Key names separated by a comma and space indicate keys pressed one after the other. For example: Esc, Ctrl+Q.
📝	The note icon indicates information that is of special interest or importance, for example, an additional action required only in certain circumstances.
💡	The tip icon indicates an idea that could be useful, for example, a way to apply the information provided in the current section to achieve a specific result.
⚠	The warning icon indicates the potential for a damaging situation, for example, data loss or corruption if certain steps are taken or not taken.

Table 2 Syntax Typographical Conventions

Convention	Use
`[ ]`	An optional item in a command or code syntax. For example: `MyCommand [optional_parameter] required_parameter`
`|`	A logical OR that separates multiple items of which only one may be chosen. For example, you can select only one of the following parameters: `MyCommand para1 | param2 | param3`

Table 2 Syntax Typographical Conventions

Convention	Use

`{ }`	A logical group of items in a command. Other syntax notations may appear within each logical group.
	For example, the following command requires two parameters, which can be either the pair `param1` and `param2`, or the pair `param3` and `param4`.
	`MyCommand {param1 param2} | {param3 param4}`
	In the next example, the command requires two parameters. The first parameter can be either `param1` or `param2` and the second can be either `param3` or `param4`:
	`MyCommand {param1 | param2} {param3 | param4}`
	In the next example, the command can accept either two or three parameters. The first parameter must be `param1`. You can optionally include `param2` as the second parameter. And the last parameter is either `param3` or `param4`.
	`MyCommand param1 [param2] {param3 | param4}`

Connecting with TIBCO Resources

How to Join TIBCOmmunity

TIBCOmmunity is an online destination for TIBCO customers, partners, and resident experts. It is a place to share and access the collective experience of the TIBCO community. TIBCOmmunity offers forums, blogs, and access to a variety of resources. To register, go to http://www.tibcommunity.com.

How to Access All TIBCO Documentation

You can access TIBCO documentation here:

http://docs.tibco.com

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, contact TIBCO Support as follows:

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:

  http://www.tibco.com/services/support

- If you already have a valid maintenance or support contract, visit this site:

  https://support.tibco.com

  Entry to this site requires a user name and password. If you do not have a user name, you can request one.

Chapter 1 Concepts

This chapter presents concepts specific to the TIBCO Rendezvous® COM component. For concepts that pertain to Rendezvous software in general, see the book *TIBCO Rendezvous Concepts*.

Topics

General Description

The Rendezvous COM component is a set of COM automation objects that allow a variety of programming language environments to access Rendezvous message middleware. The Rendezvous COM component supports several qualities of Rendezvous service—ordinary messages, fault tolerance, certified message delivery, and distributed queues.

The range of programming environments includes Visual Basic, Delphi, Visual C++, J++, Microsoft Office products, as well as scripts in Internet Explorer and Active Server Pages.

The automation objects feature dual interfaces—a custom vtable interface (for most programming languages, including Visual Basic) and a dispatch table interface (for scripting environments).

TIBCO Rendezvous no longer adds new features to the Microsoft COM interface.

However, we continue to maintain a COM interface to support legacy applications.

Programming Environments

This document is oriented toward programmers using the Visual Basic (VB) programming environment. This book presents method names using VB dot notation, and code fragments using VB syntax. It is also possible to use Rendezvous COM in other environments that accept COM components.

The release contains example programs for several environments—such as VB, Excel (VBA), VBScript, and JavaScript—however, the examples are not exhaustive.

Constants

The Rendezvous component defines all constants using enumerators in the type library. These constants are available in all programming environments.

To view these constants in VB, use the object browser.

Constants include message datatypes, queue limit policies, queue default priority, fault tolerance actions, distributed queue constants, queue dispatch constants, and error status codes.

Object Safety in Scripting Environments

Scripts can use the Rendezvous COM component without explicit confirmation in the object safety dialog box.

Certified delivery transports can write data to their ledger files. This is the only situation in which the Rendezvous component makes permanent changes to the host system.

Events in Excel

Excel drops events during certain operations (for example, while calculating a spreadsheet or editing a sheet). This behavior can result in missed Rendezvous messages and timer events.

Error Handling

See Error Overview on page 290.

Objects

This COM component implements Rendezvous objects in two layers. Each COM object is an outer shell, with a corresponding internal implementation object.

Programs create the outer object using the techniques of the programming environment, such as the `New` or `CreateObject` methods. Programs initialize the internal implementation using the `Create` method of the object class.

Once an object's internal structure is initialized, we say that the outer COM object is *valid*.

Programs can delete the internal implementation (which stops its activity) by using the `Destroy` method of an outer COM object. However, destroying the object in this way affects only the inner structure (in this situation we say that the remaining outer object is *invalid*). In contrast, VB programs delete an outer object by severing *all* references to it; to sever a variable reference, either set the object variable to `Nothing`, or exit the scope of the variable. (Other programming languages use similar constructs.) As a side effect of deleting the outer object, the inner object is also tacitly destroyed.

Creation

Object creation involves three steps—declare a variable, create the outer COM object, create the inner structure. To illustrate, these examples create message objects (the sequence is analogous for other objects).

To create an empty message object in a **VB** program, use this sequence:

```
    Dim myMsg as TibrvMsg         'Declare the variable
    set myMsg = new TibrvMsg      'Create the COM shell object
    myMsg.create                  'Init the internal message
structure
```

To create an empty message object in **VBScript**, use this sequence:

```
    Dim myMsg as Object           'Declare the variable
    set myMsg = CreateObject("TIBRVCOM.TibrvMsg")   'Create the
COM obj
    myMsg.create                  'Init the internal message
structure
```

To create an empty message object in **JavaScript**, use this sequence:

```
    var msg                       //Declare the variable
    msg = new ActiveXObject("TIBRVCOM.TibRvMsg"); //Create the
JS/COM obj
    msg.create();                 //Init the internal message
structure
```

Deletion

VB program code can delete an object by severing *all* variable references to it. Sever each variable reference in one of two ways:

- Explicit—set the variable to `Nothing`.
- Implicit—exit the scope of the variable.

Calling methods of a variable that is set to `Nothing` can yield error `91`.

Destroy Method

The `destroy` method invalidates the internal implementation of an object. The outer object remains as an empty shell.

A program may call the `destroy` method of an object more than once without error.

Validity

The `isValid` method tests the validity of an object. If its internal implementation is intact, then it is valid. (In contrast, checking that the value of a variable is not `Nothing` merely tests that the variable refers to an outer object.)

Calling any *other* methods of an object that is invalid yields the error `TIBRVCOM_NOT_INITIALIZED`. Conversely, calling the create method on a valid object yields the error `TIBRVCOM_ALREADY_INITIALIZED`.

Callback Methods and Event Handlers

Event processing in COM is more complicated than in other supported languages in the Rendezvous software family.

In the other languages, dispatching an event object triggers Rendezvous software to call back into your program code—invoking a *callback method* (or *callback function*) to process the event.

In COM, dispatching an event object triggers a longer sequence. That is, dispatch triggers a callback in the `tibrvcom` DLL, which fires a COM event, which in turn invokes the program's *event handler method* (which is associated with the event object). For consistency with other languages, this manual uses the term *callback method* to refer to that event handler method—even though (strictly speaking) COM cannot directly call back into the program.

Limitations

In the single-threaded apartment model (STA), the Rendezvous component does not trigger dispatch while a program callback method is already running. Visual Basic is subject to this limitation.

This restriction does not apply in the multi-threaded apartment model (MTA).

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)

These strings exist in two representations:

- VB and COM `BSTR`s (binary strings) use the Unicode 2-byte character set (also called *wide characters)*.
- Rendezvous wire format (internal implementation) uses the character encoding appropriate to the locale (that is, the system default ANSI code page associated with the default language installed on the computer).

  For example, the United States is locale `English_USA.1252`, and uses the Latin-1 character encoding (also called ISO 8859-1); Japan is locale `Japanese_Japan.932`, and uses the Shift-JIS character encoding.

  These strings can use either 1-byte or 2-byte character sets.

  Rendezvous strings must not contain any null characters. (Rendezvous software could truncate strings with embedded null characters. COM software stops processing `BSTR`s at the first null character.)

When two programs exchange messages within the same locale, the automatic translation between these two representations is correct. However, when a message sender and receiver use different locales, the receiving program must retranslate between code pages as needed.

Numeric Datatypes and Arrays

In many cases, Rendezvous methods present numeric data using a larger datatype. For example, field identifiers are limited to the range of 16-bit unsigned integers, but COM methods present them as 32-bit signed integers (with zeroes padding the high 16 bits). For details, see Decoding and Type Conversion on page 56.

Some methods use 64-bit integer data (for example, `TibrvMsg.getProperty()` can extract a CM sequence number). The Rendezvous COM component represents such values using the object `TibrvInt64`.

The Rendezvous COM component does *not* support arrays in scripting languages, such as VBScript and JavaScript.

Date and Time

Programs represent date and time values as a VB `Date`, or as a `Object` that contains a `Date`.

Rendezvous does not do time zone conversions, which are the responsibility of sending and receiving programs.

When printing time values, `TibrvMsg.toString()` treats all date and time values as UTC time; for more information, see Converting Dates to Strings on page 88.

The range of the VB type `Date` is from January 1, 100 to December 31, 9999. Notice that this range is narrower than the range for the corresponding Rendezvous type `TIBRVCOM_MSG_DATETIME`.

See Also DateTime Format, page 77 in *TIBCO Rendezvous Concepts*

Closure Data

Several object creation calls require a closure argument. These methods store the closure data on the object, and pass it to the callback method.

The closure data is an `Object`.

Multi-Threading

Table 3 compares the two models for multi-thread behavior in COM. Rendezvous 7.0 (and later) supports *both* models. Rendezvous objects are thread-safe in either model.

Table 3 Thread Models

Apartment Model	Free Thread Model
Single-threaded apartments (STA)	Multi-threaded apartments (MTA)
Restrictions	
Programs using this model require the Windows message queue. Windows GUI programs automatically administer the Windows message queue. Use this model in the GUI thread. Such programs can use the Rendezvous automatic dispatch queue group.	Programs must *not* use a GUI in threads that use this model. Such programs can *not* use the Rendezvous automatic dispatch queue group; instead, they must *explicitly* dispatch Rendezvous events.

See Also Tibrv.getAutoDispatchQueueGroup() on page 20

Chapter 2 Programmer’s Checklist

Topics

Checklist

Developers of Rendezvous programs can use this checklist during the three phases of the development cycle: installing Rendezvous software, coding your program, and running your program.

Install

- Install the Rendezvous software release.
- Register (only one) Rendezvous component in the Windows registry, using either these commands:

  ```
  regsvr32 path\bin\tibrvcom.dll
  ```

  ```
  regsvr32 path\bin\tibrvsdcom.dll
  ```

Code

- The project must set a reference to the Rendezvous component; see Referencing the Rendezvous COM Component on page 15.
- Excel programs (and other applications that use VBA) must enable macros in order to execute code using the Rendezvous component.
- The type library contains constant definitions that are available in all programming environments. To view these constants in VB, use the object browser. Select the appropriate constant library for your program (one includes secure daemon features, the other does not).

Run

- `rvd` must be in one of the path directories.
- `rvd` requires valid licensing.

  See Licensing Information on page 11 in *TIBCO Rendezvous Administration*.

Referencing the Rendezvous COM Component

This Rendezvous COM component is *not* an ActiveX control (in contrast, in release 5 and earlier, it *was* packaged as a control). As a result, programmers no longer create control objects in forms using drag-and-drop gestures from the control tool in the programming environment. Instead, programmers set the project to reference this component.

Select the Appropriate Component

Reference *only one* of these two components:

- `TIBCO Rendezvous TibrvCOM` *m.n* `Type Library`—for programs that connect to ordinary daemons (rvd, `rvrd`).
- `TIBCO Rendezvous Secure Daemon TibrvsdCOM` *m.n* `Type Library`—for programs that connect to secure daemons (rvsd, `rvsrd`) using SSL. For further details, see TibrvSdContext on page 26.

VB

For example, in VB, select **Project>References...** In the resulting dialog box, check either the item `TibrvCOM` *or* the item `TibrvsdCOM`—*but not both simultaneously*. Visual Basic reads the type library to define all the objects, constants and capabilities of the Rendezvous component.

Excel

For example, in Excel, select **Tools>References...** In the resulting dialog box, check either the item `TibrvCOM` *or* the item `TibrvsdCOM`—*but not both simultaneously*. Excel reads the type library to define all the objects, constants and capabilities of the Rendezvous component.

Shared Library Files

VB Programs must be able to access Rendezvous shared library files (C libraries). Table 4 details the environment variables that direct VB programs to the Rendezvous installation directory. The installation directory must contain the required shared library files.

Table 4 Environment Variables for Shared Library Files

Platform	Environment Variable
Windows	`PATH` must include `\`*install_dir*`\bin` The installation procedure sets this variable automatically.

Chapter 3 Rendezvous Environment

This chapter describes the methods related to the internal machinery upon which Rendezvous software depends.

Topics

Tibrv

Class

Purpose The Rendezvous environment.

Remarks Each program creates only one instance of `Tibrv`, and uses its methods to open and close the Rendezvous environment, refer to Rendezvous resources, and extract version information.

All other objects require an open `Tibrv` environment, and are inoperative when the environment is not open. The only exceptions to this rule are the objects defined in Chapter 4, Data, on page 33 (namely, `TibrvMsg`, `TibrvMsgField`, and `TibrvInt64`).

Method	Description	Page
Open and Close		
`Tibrv.open`	Start Rendezvous internal machinery.	25
`Tibrv.close`	Stop and destroy Rendezvous internal machinery.	19
`Tibrv.isOpen()`	Determine whether the Rendezvous machinery is open.	24
Accessors		
`Tibrv.getAutoDispatchQueueGroup()`	Return a reference to the automatic dispatch queue group object.	20
`Tibrv.getDefaultQueue()`	Return a reference to the default queue object.	21
`Tibrv.getProcessTransport()`	Return a reference to the intra-process transport object.	22

`Tibrv.getVersion()`
`Tibrv.getCmVersion()`
`Tibrv.getFtVersion()`
	Return the version string.	23

Tibrv.close

Method

Declaration `Tibrv.close`

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; all calls to any methods of these objects will fail.

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.isOpen() on page 24.

See Also Tibrv.open on page 25

Tibrv.getAutoDispatchQueueGroup()

Method

Declaration

```
set autoDispatchGroup = Tibrv.getAutoDispatchQueueGroup()
```

Purpose Return a reference to the automatic dispatch queue group object.

Remarks This method returns a special queue group, which detects the presence of an event in any of its queues, and automatically dispatches the queues (according to their priority) until no more events are waiting.

To automatically dispatch a queue, add it to this queue group.

To disable automatic dispatch of queue, remove it from this group, and arrange explicit dispatch calls instead. (It is crucial to dispatch all queues promptly—otherwise queues can overflow or discard events. A queue without a discard limit can cause the program to exceed available process storage.)

The automatic dispatch queue group is a convenience for programmers developing GUI applications. GUI programs operate best with automatic event dispatch. We do not recommend using explicit dispatch in GUI application programs; see also Multi-Threading on page 11.

Each process can have at most one automatic dispatch queue group. A program can use this queue group in at most one (STA) thread. Programs cannot destroy this queue group.

If Rendezvous is not open, this method throws the error `TIBRVCOM_TIBRV_NOT_OPEN`.

Parameter	Description
`autoDispatchGroup`	`TibrvQueueGroup` Return the automatic dispatch queue group.

⚠ Automatic dispatch continues until all the queues in this group are empty. During that time, COM does not service GUI events.

VB GUI programs can ameliorate that difficulty by including `DoEvents` statements in callback methods—but this solution requires attention to reentrance issues.

See Also TibrvQueue on page 133
TibrvQueueGroup on page 151
TibrvQueueGroup.add on page 153
TibrvQueueGroup.remove on page 159

Tibrv.getDefaultQueue()

Method

Declaration `set defaultQueue = Tibrv.getDefaultQueue()`

Purpose Return a reference to the default queue object.

Remarks If Rendezvous is not open, this method reports the error `TIBRVCOM_TIBRV_NOT_OPEN`, and returns a null object.

Each process has exactly one default queue. `Tibrv.open` automatically creates the default queue. Programs cannot destroy the default queue.

Parameter	Description
`defaultQueue`	`TibrvQueue` Return the default queue.

See Also TibrvQueue on page 133

Tibrv.getProcessTransport()

Method

Declaration `set processTransport = Tibrv.getProcessTransport()`

Purpose Return a reference to the intra-process transport object.

Remarks A program can use the intra-process transport to send messages to itself.

If Rendezvous is not open, this method reports the error `TIBRVCOM_TIBRV_NOT_OPEN`, and returns a null object.

Each process has exactly one intra-process transport. `Tibrv.open` automatically creates it. Programs cannot destroy the intra-process transport.

Parameter	Description
`processTransport`	`TibrvTransport` Return the intra-process transport.

See Also TibrvTransport on page 164

Tibrv.getVersion()

Method

Declaration

```
version = Tibrv.getVersion()
version = Tibrv.getCmVersion()
version = Tibrv.getFtVersion()
```

Purpose

Return the version string.

Remarks

`getVersion()` returns the version numbers of the Rendezvous COM component and the underlying C implementation of the core library.

`getCmVersion()` returns the version number of the underlying C implementation of the certified delivery library.

`getFtVersion()` returns the version number of the underlying C implementation of the fault tolerance library.

Version strings use this template:

`TibrvCOM Component Version` *a.b.c* `using C API version` *x.y.z*

It is important that *x.y* represent a DLL at least as recent as the component *a.b*. If not, you must reinstall a later version of Rendezvous software.

Parameter	Description
`version`	String Return a string describing the software version numbers.

Tibrv.isOpen()

Method

Declaration `open = Tibrv.isOpen()`

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`.

Parameter	Description
`open`	Boolean Return a state indicator.

See Also Tibrv.open on page 25
Tibrv.close on page 19

Tibrv.open

Method

Declaration `Tibrv.open`

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.

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 Tibrv.close on page 19

TibrvSdContext

Class

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

Remarks Each program creates only one instance of `TibrvSdContext`, and uses its static methods to configure user names, passwords and certificates, and to register trust in daemon certificates.

Before using any methods of this class, ensure that your programming environment reference the *secure daemon* `TibrvsdCOM` component; see Referencing the Rendezvous COM Component on page 15.

Programs that use methods of this class must also use the type library `Secure Daemon TIBRVSDCOMLib` (rather than `TIBRVCOMLib`) when defining callback methods.

Method	Description	Page
`TibrvSdContext.setDaemonCert()`	Register trust in a secure daemon.	27
`TibrvSdContext.setUserCertWithKey()`	Register a (PEM) certificate with private key for identification to secure daemons.	29
`TibrvSdContext.setUserCertWithKeyBin()`	Register a (PKCS #12) certificate with private key for identification to secure daemons.	30
`TibrvSdContext.setUserNameWithPassword()`	Register a user name with password for identification to secure daemons.	32

TibrvSdContext.setDaemonCert()

Method

Declaration `TibrvSdContext.`**`setDaemonCert`**` daemonName, daemonCert`

Purpose Register trust in a secure daemon.

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

Parameter	Description
`daemonName`	`String` Register a certificate for a secure daemon with this name. For the syntax and semantics of this parameter, see Daemon Name, below.
`daemonCert`	`String` Register this public certificate. The text of this certificate must be in PEM encoding. See also Certificate on page 28.

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 TibrvTransport.create on page 166.

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 SSL 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 null string (`vbNullString`). 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 on page 177 in *TIBCO Rendezvous Administration*.

In place of an actual certificate, you can also supply the null string (`vbNullString`). 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 supplying the null string (`vbNullString`) as either argument eliminates one of the two security checks before transmitting sensitive identification data to a secure daemon. We strongly discourage using the null string for both parameters simultaneously, because that would eliminate all security checks, leaving the program vulnerable to unauthorized daemons.

See Also TibrvSdContext on page 26

TibrvSdContext.setUserCertWithKey()

Method

Declaration

```
TibrvSdContext.setUserCertWithKey userCertWithKey, password
```

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 SSL 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`	`String` Register this user certificate with private key. The text of this certificate must be in PEM encoding.
`password`	`String` Use this password to decrypt the private key.

For important information about password security, see Security Factors on page 177 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 on page 177 in *TIBCO Rendezvous Administration*.

Exceptions An exception that reports status `TIBRVCOM_INVALID_FILE` can indicate either disk I/O failure, or invalid certificate data, or an incorrect password.

See Also TibrvSdContext on page 26
TibrvSdContext.setUserCertWithKeyBin() on page 30

TibrvSdContext.setUserCertWithKeyBin()

Method

Declaration

```
TibrvSdContext.setUserCertWithKeyBin
    userCertWithKey,
    userCertWithKey_size,
    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 SSL 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`	`Object` Register this user certificate with private key. The binary data of this certificate must be in PKCS #12 format.
`userCertWithKey_size`	`Long` Length of the certificate data (in bytes).
`password`	`String` Use this password to decrypt the private key.

For important information about password security, see Security Factors on page 177 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 on page 177 in *TIBCO Rendezvous Administration*.

Exceptions

An exception that reports status `TIBRVCOM_INVALID_FILE` can indicate either disk I/O failure, or invalid certificate data, or an incorrect password.

See Also

TibrvSdContext on page 26
TibrvSdContext.setUserCertWithKey() on page 29
www.rsasecurity.com/rsalabs/pkcs

TibrvSdContext.setUserNameWithPassword()

Method

Declaration `TibrvSdContext.setUserNameWithPassword userName, password`

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

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

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

For important information about password security, see Security Factors on page 177 in *TIBCO Rendezvous Administration*.

See Also TibrvSdContext on page 26

Chapter 4 Data

This chapter describes messages and the data they contain.

Topics

See Also

Strings and Character Encodings, page 7

Message Ownership and Control

The *internal structure* of each Rendezvous message belongs either to Rendezvous software or to your program. These three situations determine message ownership:

- When a program creates the internal structure of a message object (by calling `TibrvMsg.create`, `TibrvMsg.createCopy()` or `TibrvMsg.createFromByteArray`), the *program* owns that message. The program is responsible for destroying that message (by deleting all variable references to the outer message object).
- When a program receives a reply message as the return value of `TibrvTransport.sendRequest()`, the *program* owns that message. The program is responsible for destroying that message.
- When a program receives a message in a callback method, *Rendezvous software* owns that message. Rendezvous software destroys such messages when the callback method returns (unless the program explicitly detaches the message and assigns it to a global variable).
- When a program's callback method detaches an inbound message and sets a persistent reference to it, then the *program* owns the message, and must explicitly delete it.
- When a program receives a parent message, *a*, and extracts a submessage, *b*, from a field of *a*, then the program owns *b*. (That is, extracting *b* automatically detaches it.) When *a* is destroyed, *b* remains, and the program must explicitly delete it.

Rendezvous software *controls* the storage in which all messages reside. Programs can change the contents of a message only by using Rendezvous methods that add, remove or update fields.

See Also

TibrvMsg.destroy on page 53
TibrvMsg.detach on page 54

Field Objects

When a program must extract information about a message field—such as its field name, field identifier, or datatype—it can extract the field as a `TibrvMsgField` object. A `TibrvMsgField` object contains complete detail about the field.

The methods `TibrvMsg.getField()`, `TibrvMsg.getFieldByIndex()`, and `TibrvMsg.getFieldInstance()` return `TibrvMsgField` objects.

All `TibrvMsgField` objects *always* belongs to your program, so the program is always responsible for deleting the field object.

Reference Datatypes

Five datatypes are called *reference datatypes*: array, message, opaque, XML, and string.

When a message field contains a reference datatype, and a program extracts it as a field object, then the data in the field object depends on a reference to a snapshot in the parent message. For details, see Snapshot Reference on page 36.

See Also

Validity of Data Extracted from Message Fields, page 36
TibrvMsgField on page 93
TibrvMsg.getField() on page 63

Validity of Data Extracted from Message Fields

To extract data values from the fields of a message, programs use the *get* methods. All of these methods extract a *snapshot* of the message data—that is, the data value as it exists at a particular time. If the program later modifies the message by removing or updating the field, the snapshot remains unchanged.

Rendezvous messages implement snapshot semantics using two separate strategies—snapshot copy and snapshot reference.

Snapshot Copy

A snapshot *copy* is independent of the original message. The program can modify this snapshot at any time without affecting the original message. The program can update or delete the original field form the message at any time without affecting the snapshot copy.

These methods always return an independent snapshot copy of the data:

- `TibrvMsg.get()`
- `TibrvMsg.getByIndex()`
- `TibrvMsg.getInstance()`
- `TibrvMsgField.getData()`

Snapshot Reference

A snapshot *reference* is a reference to data that still resides within the original message. The validity of the reference depends on the continued validity of the original message.

These are the only methods that can produce a snapshot reference; moreover, these methods only produce a snapshot reference when the datatype of the field is array, message, opaque, XML, or string:

- `TibrvMsg.getField()`
- `TibrvMsg.getFieldByIndex()` (see TibrvMsg.getByIndex() on page 60)
- `TibrvMsg.getFieldInstance()` (see TibrvMsg.getInstance() on page 68)

Extracting the data from the resulting field object with `TibrvMsgField.getData()` *uses* the reference to yield an independent snapshot copy of the data.

⚠ When a field object contains a reference datatype (array, message, opaque, XML, or string), the data in the field object depends on a reference into the parent message. If the parent message is deleted, the program can no longer extract these datatypes from the field object; attempting to extract them references freed storage, which is illegal, and can cause serious consequences.

Deleting Snapshot References

Ordinarily, snapshot references remain part of the message until the program destroys the message. However, repeated `getField` calls can cause snapshots to accumulate within a message, causing unbounded memory growth.

For example, consider the result of a program that calls a method repeatedly on the same message, where each call creates a new snapshot within the storage associated with that message. Message storage grows, and destroying the message is the only way to free that storage.

A pair of methods give programs explicit control over snapshot references, so you can avoid such situations:

- `TibrvMsg.markReferences`
- `TibrvMsg.clearReferences`

When a program repeatedly extracts snapshot references data *and* does not destroy the parent messages, consider using these methods to control the proliferation of references.

See Also TibrvMsg.markReferences on page 77

Multiple Subscription Snapshots

Rendezvous software also protects the integrity of messages distributed to multiple subscriptions. When a callback method modifies an inbound message (whether detached or not), Rendezvous software still presents the original message content to subsequent callback methods.

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.

  An identifier has the range of a 16-bit unsigned integer, [0, 65535]. We represent identifiers with `Long` values (32-bit signed integers) in this range.

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 on page 38.

Rules and Restrictions

It is illegal for a field to have *both* a null field name (`vbNullString`) *and* a non-zero identifier.

Adding a New Field

When a program adds a new field to a message, it can specify 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 and reports the error `TIBRVCOM_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.getInstance() on page 68
- TibrvMsg.removeInstance on page 81

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 one-dimensional array datatypes within message fields. The method `TibrvMsg.getInstance()` 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.

TibrvMsg

Class

Purpose Represent Rendezvous messages.

(Sheet 1 of 3)

Method	Description	Page
Life Cycle		
`TibrvMsg.create`	Initialize the message object.	50
`TibrvMsg.createCopy()`	Copy a message object.	51
`TibrvMsg.destroy`	Destroy a message object.	53
`TibrvMsg.detach`	Detach an inbound message from Rendezvous storage; the program assumes responsibility for destroying the message.	54
`TibrvMsg.isDetached()`	Test the ownership of a message object.	75
`TibrvMsg.isValid()`	Test the validity of a message object.	76
`TibrvMsg.reset`	Clear a message, preparing it for re-use.	82
Fields		
`TibrvMsg.add`	Add a field to a message.	45
`TibrvMsg.addField`	Add a field object to a message.	48
`TibrvMsg.get()`	Get the value of a specified field from a message.	55
`TibrvMsg.getByIndex()`	Get the value of a field from a message by an index.	60
`TibrvMsg.getField()`	Get a specified field from a message.	63
`TibrvMsg.getFieldByIndex()`	Get a field from a message by an index.	65
`TibrvMsg.getFieldInstance()`	Get the value of a specific field instance from a message.	66

(Sheet 2 of 3)

Method	Description	Page
`TibrvMsg.getInstance()`	Get the value of a specific field instance from a message.	68
`TibrvMsg.remove`	Remove a field from a message.	79
`TibrvMsg.removeInstance`	Remove a specified instance of a field from a message.	81
`TibrvMsg.update`	Update a field within a message.	89
`TibrvMsg.updateField`	Update a field within a message.	92
Address Information and Properties		
`TibrvMsg.getProperty()`	Extract a property from a message.	72
`TibrvMsg.getReplySubject()`	Extract the reply subject from a message.	73
`TibrvMsg.getSendSubject()`	Extract the subject from a message.	74
`TibrvMsg.setProperty`	Set a property on a message.	85
`TibrvMsg.setReplySubject`	Set the reply subject for a message.	86
`TibrvMsg.setSendSubject`	Set the subject for a message.	87
Archive		
`TibrvMsg.createFromByteArray`	Initialize a message with data from a byte sequence.	52
`TibrvMsg.getAsByteArray()`	Convert the data from a message to a byte sequence.	59
Accessors		
`TibrvMsg.getByteSize()`	Get the size of the message.	61
`TibrvMsg.getNumFields()`	Extract the number of fields in a message.	70
String		
`TibrvMsg.toString()`	Format a message as a string.	88

(Sheet 3 of 3)

Method	Description	Page
References		
`TibrvMsg.markReferences`	Mark references in this message.	77
`TibrvMsg.clearReferences`	Clear references in this message.	49
Encoding		
`TibrvMsg.getDefaultInboundEncoding()`	Get the global default encoding for inbound messages.	62
`TibrvMsg.getMsgEncoding()`	Get the character encoding of a message.	69
`TibrvMsg.getOutboundEncoding()`	Get the global encoding for outbound messages.	71
`TibrvMsg.setDefaultInboundEncoding`	Set the global default encoding for inbound messages.	83
`TibrvMsg.setOutboundEncoding`	Set the global encoding for outbound messages.	84

Datatype Constants

The type library defines these constants, which denote wire format datatypes.

Table 5 Datatype Constants (Sheet 1 of 2)

Constant	Comments
`TIBRVCOM_MSG_NO_TAG`	
`TIBRVCOM_MSG_MSG`	
`TIBRVCOM_MSG_DATETIME`	
`TIBRVCOM_MSG_OPAQUE`	
`TIBRVCOM_MSG_STRING`	
`TIBRVCOM_MSG_XML`	Byte-array, compressed for network transmission.
`TIBRVCOM_MSG_BOOL`	
`TIBRVCOM_MSG_I8`	

Table 5 Datatype Constants (Sheet 2 of 2)

Constant	Comments
TIBRVCOM_MSG_U8	
TIBRVCOM_MSG_I16	
TIBRVCOM_MSG_U16	
TIBRVCOM_MSG_I32	
TIBRVCOM_MSG_U32	
TIBRVCOM_MSG_I64	
TIBRVCOM_MSG_U64	
TIBRVCOM_MSG_F32	
TIBRVCOM_MSG_F64	
TIBRVCOM_MSG_IPPORT16	
TIBRVCOM_MSG_IPADDR32	
TIBRVCOM_MSG_I8ARRAY	
TIBRVCOM_MSG_U8ARRAY	
TIBRVCOM_MSG_I16ARRAY	
TIBRVCOM_MSG_U16ARRAY	
TIBRVCOM_MSG_I32ARRAY	
TIBRVCOM_MSG_U32ARRAY	
TIBRVCOM_MSG_I64ARRAY	
TIBRVCOM_MSG_U64ARRAY	
TIBRVCOM_MSG_F32ARRAY	
TIBRVCOM_MSG_F64ARRAY	
TIBRVCOM_MSG_USER_FIRST	
TIBRVCOM_MSG_USER_LAST	

See Also

Strings and Character Encodings, page 7
TibrvMsgField on page 93

TibrvMsg.add

Method

Declaration `TibrvMsg.add fieldName, data, tibrvType, fieldId`

Purpose Add a field to a message.

Remarks This method copies the information into the new message field. Subsequent changes to the arguments do not affect the field in the message object.

Parameter	Description
`fieldName`	`String` Add a field with this name. `vbNullString` is a legal name. However, if `fieldId` is non-zero, then `fieldName` must be non-null.
`data`	`Object` Add a field with this data value. Pass the data as any VB type or Rendezvous object type. When this parameter is a `TibrvMsgField` object, this method cannot convert it to another type. The `tibrvType` parameter must be zero (or omitted).
`tibrvType`	`Byte` optional When absent or zero, determine the field's type from the type of the object `data`. In Figure 1 on page 47, filled dots indicate default encodings between homologous types. When present, explicitly convert the data to this type. For a list of types, see Datatype Constants on page 42. Figure 1 on page 47 indicates permissible conversions.
`fieldId`	`Long` optional Add a field with this identifier. All field identifiers must be unique within each message. An identifier has the numeric range of a 16-bit unsigned integer, [0, 65535]. When absent or zero, add a field without an identifier. 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.

Encoding and Type Conversion

This method automatically converts VB data to its corresponding Rendezvous wire format type. Figure 1 on page 47 specifies default encodings with filled circles; you can override the default encoding by supplying an explicit `tibrvType` argument (numeric and supported conversions). This feature lets you add fields with datatypes that the programming language does not directly support (for example, a signed byte or an unsigned integer.

When adding a numeric value as a `TIBRVMSG_BOOL`, zero maps to `False`, and all non-zero values map to `True`.

When adding a boolean value as a numeric scalar type, `False` maps to zero, and `True` maps to 1.

When adding an unsigned value as a signed type, this method interprets the high bit as a sign bit. When adding a numeric value as a type with lesser precision, this method discards bits of precision without warning. When adding a numeric value as a type with smaller range, this method checks the range and reports an error if the target type cannot accommodate the value.

Encoding XML

As a convenience, Visual Basic lets you add a string value as an XML field; the resulting XML is encoded as UTF-16LE (16-bit Unicode, little-endian byte order). However, other language APIs might not support this encoding. For this reason, we discourage programmers from directly adding string values as XML fields. Instead, we recommend converting the XML document string to a byte sequence, using the encoding that corresponds to your locale. This practice ensures that all receivers can easily parse the resulting XML document. For more information, see Decoding XML on page 57.

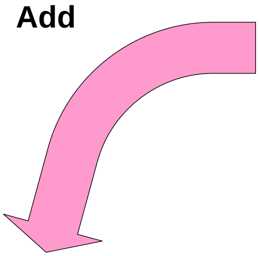
Nested Message

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.

Field Name Length

The the longest possible field name is 127 bytes.

Figure 1 VB to Wire Format Datatype Conversion Matrix



		VB Native Type															
		Boolean	Single	Double	Byte	Integer	Long	TibrvInt64	Single()	Double()	Byte()	Integer()	Long()	TibrvMsgInt64()	Date	TibrvMsg	String
TibrvMsg Destination Type	TIBRVCOM_MSG_BOOL	●	S	S	S	S	S										P
	TIBRVCOM_MSG_F32	S	●	N	S	S	N										P
	TIBRVCOM_MSG_F64	S	S	●	S	S	N										P
	TIBRVCOM_MSG_I8	S	N	N	+	N	N										P
	TIBRVCOM_MSG_I16	S	N	N	S	●	S										P
	TIBRVCOM_MSG_I32	S	N	N	S	S	●										P
	TIBRVCOM_MSG_I64	S	N	N	S	S	S	●									P
	TIBRVCOM_MSG_U8	S	N	N	●	N	N										P
	TIBRVCOM_MSG_U16	S	N	N	S	+	N										P
	TIBRVCOM_MSG_U32	S	N	N	S	S	+										P
	TIBRVCOM_MSG_U64	S	N	N	S	S	S	+									P
	TIBRVCOM_MSG_IPADDR32		N	N	S	S	S										P
	TIBRVCOM_MSG_IPPORT16		N	N	S	S	N										P
	TIBRVCOM_MSG_DATETIME														●		
	TIBRVCOM_MSG_F32ARRAY								●								
	TIBRVCOM_MSG_F64ARRAY									●							
	TIBRVCOM_MSG_I8ARRAY										+						
	TIBRVCOM_MSG_I16ARRAY											●					
	TIBRVCOM_MSG_I32ARRAY												●				
	TIBRVCOM_MSG_I64ARRAY													●			
	TIBRVCOM_MSG_U8ARRAY										S						
	TIBRVCOM_MSG_U16ARRAY											+					
	TIBRVCOM_MSG_U32ARRAY												+				
	TIBRVCOM_MSG_U64ARRAY													+			
	TIBRVCOM_MSG_MSG															●	
	TIBRVCOM_MSG_OPAQUE										●						S
	TIBRVCOM_MSG_STRING																●
	TIBRVCOM_MSG_XML										S						S

Key	
●	Homologous types; conversion always supported; no loss of information
S	Supported conversion; always supported
N	Numeric conversion; out of range is an error; loss of sign or precision is OK
+	Signed interpreted as unsigned integer, or unsigned interpreted as signed
P	Parsed conversion; supported when string has appropriate format
	Unsupported conversion

See Also TibrvMsg.addField on page 48

TibrvMsg.addField

Method

Declaration `TibrvMsg.`**`addField`**` field`

Purpose Add a field object to a message.

Remarks This method copies the information into a new message field, without type conversion.

It is illegal to add a field that has both a null field name (`vbNullString`), and a non-zero field identifier.

Parameter	Description
`field`	`TibrvMsgField` Add this field to the message.

See Also TibrvMsg.add on page 45

TibrvMsg.clearReferences

Method

Declaration `TibrvMsg.`**`clearReferences`**

Purpose Clear references in this message.

Remarks This method clears references back to the most recent mark.

For a description and example, see TibrvMsg.markReferences on page 77.

See Also Validity of Data Extracted from Message Fields, page 36
Deleting Snapshot References, page 37
TibrvMsg.markReferences on page 77

TibrvMsg.create

Method

Declaration `TibrvMsg.`**`create`**` initialSize`

Purpose Initialize the message object.

Remarks This method initializes the internal structure of the message object, allocating storage for it.

Calling this method on a valid object yields the error `TIBRVCOM_ALREADY_INITIALIZED`.

Parameter	Description
`initialSize`	`Long`; optional Allocate a message of this non-negative size (in bytes). (Messages can grow dynamically.) When absent or zero, Rendezvous allocates initial storage automatically. We recommend omitting this argument.

TibrvMsg.createCopy()

Method

Declaration

```
set msgCopy = TibrvMsg.createCopy()
```

Purpose

Copy a message object.

Remarks

Create an independent copy of the message, copying all its fields (that is, field values are also independent copies).

This method does not copy subject information, certified delivery information, nor other supplementary information or properties.

This method does not copy snapshot references into the new copy message.

Parameter	Description
`msgCopy`	`TibrvMsg` Return an independent copy of the message.

TibrvMsg.createFromByteArray

Method

Declaration `TibrvMsg.`**`createFromByteArray`**` byteArray`

Purpose Initialize a message with data from a byte sequence.

Remarks This method is an alternative to `TibrvMsg.create`. It initializes the message with data from the byte sequence.

To extract a byte sequence from a message, see TibrvMsg.getAsByteArray() on page 59.

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.

Calling this method on a valid object yields the error `TIBRVCOM_ALREADY_INITIALIZED`.

Parameter	Description
`byteArray`	`Object` Initialize the message with data from this one-dimensional byte array.

See Also TibrvMsg.create on page 50
TibrvMsg.getAsByteArray() on page 59

TibrvMsg.destroy

Method

Declaration `TibrvMsg.`**`destroy`**

Purpose Destroy a message object.

Remarks This method destroys only the internal structure of the message object. Rendezvous software automatically destroys a message when program execution leaves the scope of the object, or deletes all references to the object.

See Also Message Ownership and Control, page 34

TibrvMsg.detach

Method

Declaration `TibrvMsg.detach`

Purpose Detach an inbound message from Rendezvous storage; the program assumes responsibility for destroying the message.

Remarks When Rendezvous software creates a message, it owns that message. This situation occurs only in the case of inbound messages presented to a data callback method; Rendezvous software destroys such messages when the callback method returns, unless the program explicitly detaches the message. After a detach operation, the program owns the message, and must explicitly destroy it to reclaim the storage.

Example In this example, `msgref` is accessible after the callback method returns.

```
Dim msgref as TibrvMsg

Private Sub mylis_OnMsg( _
      ByVal lisn as TIBRVCOMLib.ITibrvListener, _
      ByVal message as TIBRVCOMLib.ITibrvMsg)
  set msgref = message
  msgref.detach
End Sub
```

See Also Message Ownership and Control on page 34
TibrvMsg.destroy on page 53
TibrvMsg.isDetached() on page 75
TibrvListener_onMsg on page 120

TibrvMsg.get()

Method

Declaration

```
data = TibrvMsg.get(fieldName, tibrvType, fieldId)
```

Purpose

Get the value of a specified field from a message.

Remarks

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

This method returns an independent snapshot copy of the field value. For details, see Snapshot Copy on page 36.

Programs can use a related method to loop through all the fields of a message; to retrieve each field of a message, see TibrvMsg.getByIndex() on page 60.

Parameter	Description
`fieldName`	`String` Get a field with this name. See Field Search Algorithm. `vbNullString` is a legal field name.
`tibrvType`	`Byte` optional Convert the data value to the object datatype corresponding to this type, if possible. For a list of types, see Datatype Constants on page 42. When absent or zero, automatically convert the data to its homologous type. Figure 2 on page 58 indicates default decodings between homologous types.
`fieldId`	`Long` optional Get the field with this identifier. See Field Search Algorithm. An identifier has the range of a 16-bit unsigned integer, [0, 65535].
`data`	`Object` Return the data value of the field.

Field Search Algorithm

`TibrvMsg.get()`, `TibrvMsg.getField()`, `TibrvMsg.getInstance()` and `TibrvMsg.getFieldInstance()` use this algorithm to find a field within a message, as specified by a field identifier and a field name.

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 zero, return the field.

   If the name search succeeds, but the actual identifier in the field is non-zero (so it does not match the identifier supplied) then report the error `TIBRVCOM_ID_CONFLICT`.

   On failure, or if the program supplied `vbNullString` as the field name, report the error `TIBRVCOM_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 `vbNullString`.

   If the search succeeds, return the field.

   On failure, report the error `TIBRVCOM_NOT_FOUND`.

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

When an error occurs, the method returns an empty object.

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.

Decoding and Type Conversion

This method automatically decodes the extracted field data from its Rendezvous wire format type to a corresponding VB type. In Figure 2 on page 58, filled circles (as well as + and - symbols) specify default decodings between homologous types; you can override the default decoding by supplying an explicit `tibrvType` argument (numeric and supported conversions).

The default conversion for extracting a TIBRVCOM_MSG_I8 yields an Integer, with zeroes padding the high 8 bits. (Similarly for TIBRVCOM_MSG_I8ARRAY.)

The default conversion for extracting a TIBRVCOM_MSG_U16 yields a Long, with zeroes padding the high 16 bits. (Similarly for TIBRVCOM_MSG_U16ARRAY.)

The default conversion for extracting a TIBRVCOM_MSG_U32 yields a Long, interpreting the high bit as a sign bit. (Similarly for TIBRVCOM_MSG_U32ARRAY.)

When extracting a numeric value as a boolean, zero maps to boolean False. All non-zero numeric values map to boolean True.

When extracting a TIBRVMSG_BOOL as a numeric value, False maps to zero, and True maps to 1.

When extracting a TIBRVMSG_STRING as a boolean, if the first character is either t or T, then it maps to boolean True. All other strings map to boolean False.

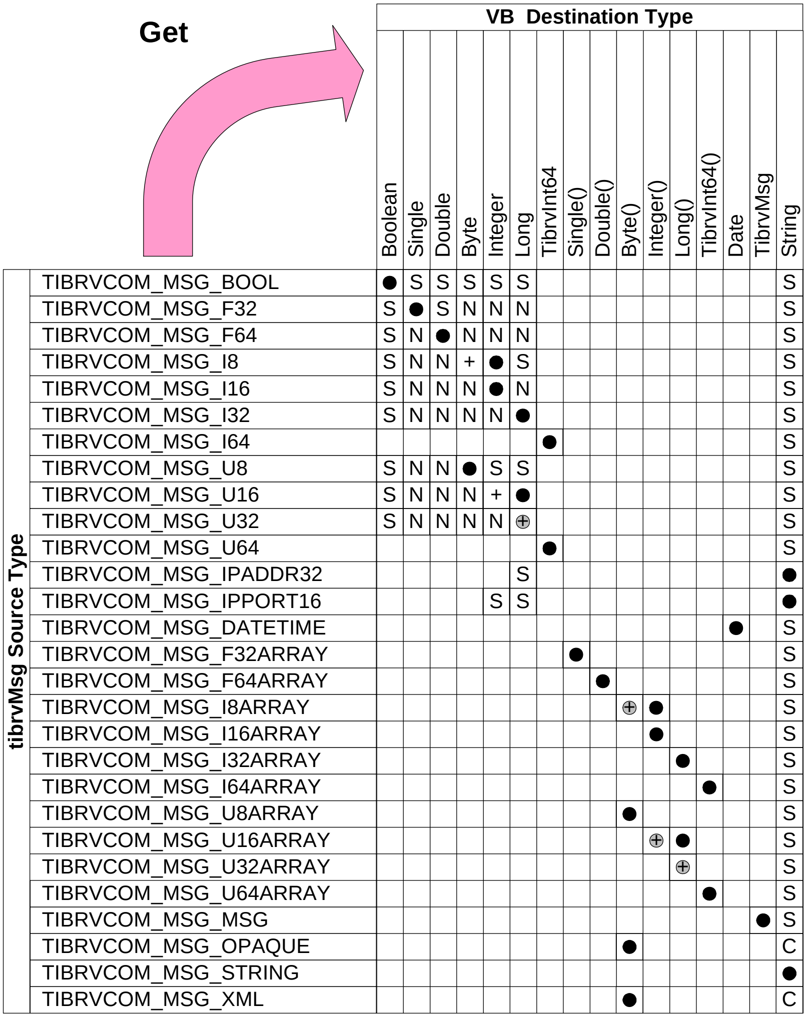
Extracting a TIBRVCOM_MSG_IPADDR32 as a string yields the standard dot notation (a.b.c.d).

To determine the datatype of the original message field, use TibrvMsg.getField() and TibrvMsgField.getType().

Decoding XML

As a convenience, Visual Basic lets you extract an XML field as a string value; however, the resulting string is only valid if the XML field uses UTF-16LE encoding. Other language APIs might not support this encoding. For this reason, we discourage programmers from directly extracting an XML field as a string value. Instead, we recommend extracting the XML document as a byte array, and then explicitly converting it to a string using the encoding that corresponds to your locale. For more information, see Encoding XML on page 46.

Figure 2 Wire Format to VB Datatype Conversion Matrix



tibrvMsg Source Type	VB Destination Type															
	Boolean	Single	Double	Byte	Integer	Long	TibrvInt64	Single()	Double()	Byte()	Integer()	Long()	TibrvInt64()	Date	TibrvMsg	String
TIBRVCOM_MSG_BOOL	●	S	S	S	S	S										S
TIBRVCOM_MSG_F32	S	●	S	N	N	N										S
TIBRVCOM_MSG_F64	S	N	●	N	N	N										S
TIBRVCOM_MSG_I8	S	N	N	+	●	S										S
TIBRVCOM_MSG_I16	S	N	N	N	●	N										S
TIBRVCOM_MSG_I32	S	N	N	N	N	●										S
TIBRVCOM_MSG_I64							●									S
TIBRVCOM_MSG_U8	S	N	N	●	S	S										S
TIBRVCOM_MSG_U16	S	N	N	N	+	●										S
TIBRVCOM_MSG_U32	S	N	N	N	N	⊕										S
TIBRVCOM_MSG_U64							●									S
TIBRVCOM_MSG_IPADDR32						S										●
TIBRVCOM_MSG_IPPORT16					S	S										●
TIBRVCOM_MSG_DATETIME														●		S
TIBRVCOM_MSG_F32ARRAY								●								S
TIBRVCOM_MSG_F64ARRAY									●							S
TIBRVCOM_MSG_I8ARRAY										⊕	●					S
TIBRVCOM_MSG_I16ARRAY											●					S
TIBRVCOM_MSG_I32ARRAY												●				S
TIBRVCOM_MSG_I64ARRAY													●			S
TIBRVCOM_MSG_U8ARRAY										●						S
TIBRVCOM_MSG_U16ARRAY											⊕	●				S
TIBRVCOM_MSG_U32ARRAY												⊕				S
TIBRVCOM_MSG_U64ARRAY													●			S
TIBRVCOM_MSG_MSG															●	S
TIBRVCOM_MSG_OPAQUE										●						C
TIBRVCOM_MSG_STRING																●
TIBRVCOM_MSG_XML										●						C

Key	
●	Homologous types; conversion always supported
S	Supported conversion; always supported
N	Numeric conversion; loss of sign or precision OK; out of range truncated (no error)
+	Signed interpreted as unsigned integer, or unsigned interpreted as signed
⊕	Homologous types; conversion always supported, but sign bit interpretation as with +
C	Caution; supported, but results sometimes can be misleading
	Unsupported conversion

See Also TibrvMsg.getByIndex() on page 60
TibrvMsg.getInstance() on page 68
TibrvMsg.getField() on page 63

TibrvMsg.getAsByteArray()

Method

Declaration `byteArray = TibrvMsg.`**`getAsByteArray`**`()`

Purpose Convert the data from a message to a byte sequence.

Remarks Return a copy of the data as a byte sequence, suitable for archiving in a file. To reconstruct the message from bytes, see TibrvMsg.createFromByteArray on page 52.

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 zero bytes.

Parameter	Description
`byteArray`	`Object` Return the message data as a one-dimensional byte array.

See Also TibrvMsg.createFromByteArray on page 52
TibrvMsg.getByteSize() on page 61

TibrvMsg.getByIndex()

Method

Declaration

```
data   = TibrvMsg.getByIndex(index)
```

Purpose Get the value of 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.

This method returns an independent snapshot copy of the field value. For details, see Snapshot Copy on page 36.

These two related methods differ only in the form of their return values. The first returns the field's data value; the second returns a `TibrvMsgField` object.

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

Parameter	Description
`index`	`Long` Get the field with this index. Zero specifies the first field.
`data`	`Object` Return the data value of the field.

TibrvMsg.getByteSize()

Method

Declaration `byteSize = TibrvMsg.`**`getByteSize`**`()`

Purpose Get the size of the message.

Remarks This method returns the size of the message. Programs can use this size in conjunction with `TibrvMsg.getAsByteArray()`; for example, when writing the message byte array to a file, and reading it in again (using binary file I/O).

Parameter	Description
`byteSize`	`Long` Return the size of the wire-format message (in bytes).

See Also TibrvMsg.createFromByteArray on page 52
TibrvMsg.getAsByteArray() on page 59

TibrvMsg.getDefaultInboundEncoding()

Method

Declaration `encoding = TibrvMsg.getDefaultInboundEncoding()`

Purpose Get the global default encoding for inbound messages.

Remarks When an inbound message arrives without an explicit encoding tag, `TibrvMsg.getMsgEncoding()` in the receiving program infers this default encoding.

If no global default is set, this call yields `vbNullString` and returns normally.

Parameter	Description
`encoding`	`String` The program supplies a location in this parameter; the function stores the name of the encoding (as a string) in that location.

See Also TibrvMsg.getMsgEncoding() on page 69
TibrvMsg.setDefaultInboundEncoding on page 83

TibrvMsg.getField()

Method

Declaration `set field = TibrvMsg.`**`getField`**`(fieldName, fieldId)`

Purpose Get a specified field from a message.

Remarks When a program must extract information about a message field—such as its field name, field identifier, or datatype—it can extract the field as a `TibrvMsgField` object. A `TibrvMsgField` object contains complete detail about the field.

All `TibrvMsgField` objects *always* belongs to your program, so the program is always responsible for deleting the field object.

Programs specify the field to retrieve using the `fieldName` and `fieldId` parameters. The method returns a `TibrvMsgField` object. Unlike `TibrvMsg.get()`, this method does not attempt any datatype conversions.

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() on page 65.

Parameter	Description
`fieldName`	`String` Get a field with this name. See Field Search Algorithm on page 55. `vbNullString` is a legal field name.
`fieldId`	`Long` optional Get the field with this identifier. See Field Search Algorithm on page 55. An identifier has the range of a 16-bit unsigned integer, [0, 65535].
`field`	`TibrvMsgField` Return the field object.

⚠ When a field object contains a reference datatype (array, message, opaque, or string), the data in the field object depends on a reference into the parent message. If the parent message is deleted, the program can no longer extract these datatypes from the field object; attempting to extract them references freed storage, which is illegal, and can cause serious consequences.

See Also

Field Objects, page 35
TibrvMsg.get() on page 55
TibrvMsgField on page 93

TibrvMsg.getFieldByIndex()

Method

Declaration

```
set field = TibrvMsg.getFieldByIndex(index)
```

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.

When a program must extract information about a message field—such as its field name, field identifier, or datatype—it can extract the field as a `TibrvMsgField` object. A `TibrvMsgField` object contains complete detail about the field.

All `TibrvMsgField` objects *always* belongs to your program, so the program is always responsible for deleting the field object.

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

Parameter	Description
`index`	`Long` Get the field with this index. Zero specifies the first field.
`field`	`TibrvMsgField` Return the field object.

⚠ When a field object contains a reference datatype (array, message, opaque, XML, or string), the data in the field object depends on a reference into the parent message. If the parent message is deleted, the program can no longer extract these datatypes from the field object; attempting to extract them references freed storage, which is illegal, and can cause serious consequences.

See Also TibrvMsg.getByIndex() on page 60
TibrvMsgField on page 93

TibrvMsg.getFieldInstance()

Method

Declaration

```
set field = TibrvMsg.getFieldInstance(fieldName, instance)
```

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 i*th* field named foo). Programs can use this method in a loop that examines every field with a specified name.

The first instance of the named field is instance 1.

When a program must extract information about a message field—such as its field name, field identifier, or datatype—it can extract the field as a TibrvMsgField object. A TibrvMsgField object contains complete detail about the field.

All TibrvMsgField objects *always* belongs to your program, so the program is always responsible for deleting the field object.

When the instance argument is greater than the actual number of instances of the field in the message, this method reports the error TIBRVCOM_NOT_FOUND.

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 one-dimensional array datatypes within message fields. The method TibrvMsg.getInstance() 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.

(Sheet 1 of 2)

Parameter	Description
fieldName	String Get an instance of a field with this name. vbNullString is a legal field name.
instance	Long Get this instance of the specified field name. The argument 1 denotes the first instance of the named field.

(Sheet 2 of 2)

Parameter	Description
`field`	`TibrvMsgField` Return the field object.

⚠ When a field object contains a reference datatype (array, message, opaque, XML, or string), the data in the field object depends on a reference into the parent message. If the parent message is deleted, the program can no longer extract these datatypes from the field object; attempting to extract them references freed storage, which is illegal, and can cause serious consequences.

See Also TibrvMsg.getInstance() on page 68
TibrvMsgField on page 93

TibrvMsg.getInstance()

Method

Declaration `data = TibrvMsg.getInstance(fieldName, instance)`

Purpose Get the value of a specific field instance 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 `i`*th* field named `foo`). Programs can use this method in a loop that examines every field with a specified name.

The first instance of the named field is instance `1`.

This method returns an independent snapshot copy of the field value. For details, see Snapshot Copy on page 36.

When the `instance` argument is greater than the actual number of instances of the field in the message, this method reports the error `TIBRVCOM_NOT_FOUND`.

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 one-dimensional array datatypes within message fields. The method `TibrvMsg.getInstance()` 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`	`String` Get an instance of a field with this name. `vbNullString` is a legal field name.
`instance`	`Long` Get this instance of the specified field name. The argument `1` denotes the first instance of the named field.
`data`	`Object` Return the data value of the field.

See Also TibrvMsg.getFieldInstance() on page 66
TibrvMsgField on page 93

TibrvMsg.getMsgEncoding()

Method

Declaration `encoding = TibrvMsg.`**`getMsgEncoding`**`()`

Purpose Get the character encoding of a message.

Remarks This call yields either the actual encoding stored with a message, or (if none is stored) a reasonable estimate:

- When a message is created locally, and its send subject is set, then its encoding is also set, and this call gets the actual encoding from the message. (This value is identical to the program's global outbound encoding.)
- When the message is created locally and its send subject is *not* set, then its encoding is also not set, and this call gets the program's global outbound encoding. If no global value is set, this call yields the string `UNKNOWN`.
- If the message is inbound, and bears an explicit encoding, then this call gets that encoding.
- If the message is inbound, and does *not* bear an explicit encoding, then this call gets the program's global default inbound encoding. If no global default is set, this call yields the string `UNKNOWN`.

Parameter	Description
`encoding`	`String` The program supplies a location in this parameter; the function stores the name of the encoding (as a string) in that location.

See Also TibrvMsg.getDefaultInboundEncoding() on page 62
TibrvMsg.getOutboundEncoding() on page 71
TibrvMsg.setDefaultInboundEncoding on page 83
TibrvMsg.setOutboundEncoding on page 84
TibrvMsg.setSendSubject on page 87

TibrvMsg.getNumFields()

Method

Declaration `numFields = TibrvMsg.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.

Parameter	Description
`numFields`	`Long` Return the number of fields in the message.

TibrvMsg.getOutboundEncoding()

Method

Declaration `encoding = TibrvMsg.`**`getOutboundEncoding`**`()`

Purpose Get the global encoding for outbound messages.

Remarks If no global value is set, this call yields `vbNullString` and returns normally.

Parameter	Description
`encoding`	`String` The program supplies a location in this parameter; the function stores the name of the encoding (as a string) in that location.

See Also TibrvMsg.getMsgEncoding() on page 69
TibrvMsg.setOutboundEncoding on page 84

TibrvMsg.getProperty()

Method

Declaration `value = TibrvMsg.getProperty(name)`

Purpose Extract a property from a message.

Remarks Properties are supplementary information—they are *not* part of the message itself.

Parameter	Description
`name`	`String` Extract the property with this name.
`value`	`Object` Return this property value. If the property is not set, this method reports an error, and the object is empty.

Property	Description
`cmSender`	`String` The correspondent name of the CM transport that sent a labeled message. When the message is not labeled with this CM property, the method reports the error `TIBRVCOM_NOT_FOUND`.

`cmSequenceNumber`
`cmSequence`
(either name extracts the same property)
	`TibrvInt64`
	The sequence number of labeled message.
	When the message is not labeled with this CM property, the method reports the error `TIBRVCOM_NOT_FOUND`, and the `TibrvInt64` contains zero.

`cmTimeLimit`	`Double` The CM time limit of a labeled message. If the message is not labeled, the value is zero. This value is always a positive integer.

See Also TibrvMsg.setProperty on page 85
CM Label Information, page 235
Supplementary Information for Messages on page 41 in *TIBCO Rendezvous Concepts*

TibrvMsg.getReplySubject()

Method

Declaration `replySubject = TibrvMsg.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.

Parameter	Description
`replySubject`	`String` Return the reply subject of the message. The method makes a snapshot copy of the reply subject string, and supplies a pointer to that snapshot within message storage. The pointer remains valid as long as the message itself remains valid in the same location. The reply subject pointer becomes *invalid* if the program destroys the message, or returns from the data callback method that determines the scope of an inbound message. For more information, see Snapshot Reference on page 36, and Deleting Snapshot References on page 37. If the reply subject is not set, this method reports the error `TIBRVCOM_NOT_FOUND`, and the string is `vbNullString`.

See Also TibrvMsg.setReplySubject on page 86
Supplementary Information for Messages on page 41 in *TIBCO Rendezvous Concepts*

TibrvMsg.getSendSubject()

Method

Declaration

```
sendSubject = TibrvMsg.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.

Parameter	Description
`sendSubject`	`String` Return the subject of the message. The method makes a snapshot copy of the subject string, and supplies a pointer to that snapshot within message storage. The pointer remains valid as long as the message itself remains valid in the same location. The subject pointer becomes *invalid* if the program destroys the message, or returns from the data callback method that determines the scope of an inbound message. For more information, see Snapshot Reference on page 36, and Deleting Snapshot References on page 37. If the destination subject is not set, this method reports the error `TIBRVCOM_NOT_FOUND`, and the string is `vbNullString`.

See Also TibrvMsg.setSendSubject on page 87
Supplementary Information for Messages on page 41 in *TIBCO Rendezvous Concepts*

TibrvMsg.isDetached()

Method

Declaration

```
isDetached = TibrvMsg.isDetached()
```

Purpose Test the ownership of a message object.

Parameter	Description
`isDetached`	`Boolean` Return `True` if the message is detached (the program owns the object, and must destroy it). Return `False` if the message is *not* detached (Rendezvous software owns the object, and will destroy it).

See Also TibrvMsg.destroy on page 53
TibrvMsg.detach on page 54

TibrvMsg.isValid()

Method

Declaration `isValid = TibrvMsg.isValid()`

Purpose Test the validity of a message object.

Parameter	Description
`isValid`	`Boolean` Return `True` if the message is valid. Return `False` if the internal structure of the message has been destroyed.

TibrvMsg.markReferences

Method

Declaration `TibrvMsg.`**`markReferences`**

Purpose Mark references in this message.

Remarks Extracting a field object from a message creates a snapshot of that data. When the field contains a reference datatype, the snapshot is a reference that depends on the parent message. A reference snapshot remains associated with the message until the program destroys the message. Repeatedly extracting field objects can cause reference snapshots to accumulate within a program, causing unbounded memory growth. This method gives programs explicit control over snapshot references; by clearing references, the program declares that it no longer needs the references that arise as side effects of calls that extract a message field object.

For example, consider a program fragment that repeatedly sends a template message, extracting field objects from the message before each send call. Each call to extract a field produces a snapshot reference. By surrounding the *getField* operation with a *mark* and *clear* pair (with the *clear* call occurring at any time after the *getField* call), the program releases the reference, which helps control memory usage.

Every call to `TibrvMsg.markReferences` must be paired with a call to `TibrvMsg.clearReferences`. It is legal to mark references several times, as long as the program eventually clears all the marks. To understand this idea, it is helpful to think of get and mark as *pushdown* operations, and clear as a *pop* operation. Figure 3 on page 78 illustrates that each clear call deletes snapshots back to the most recent mark.

Unless a program explicitly marks and clears references, references persist until the message is destroyed or reset.

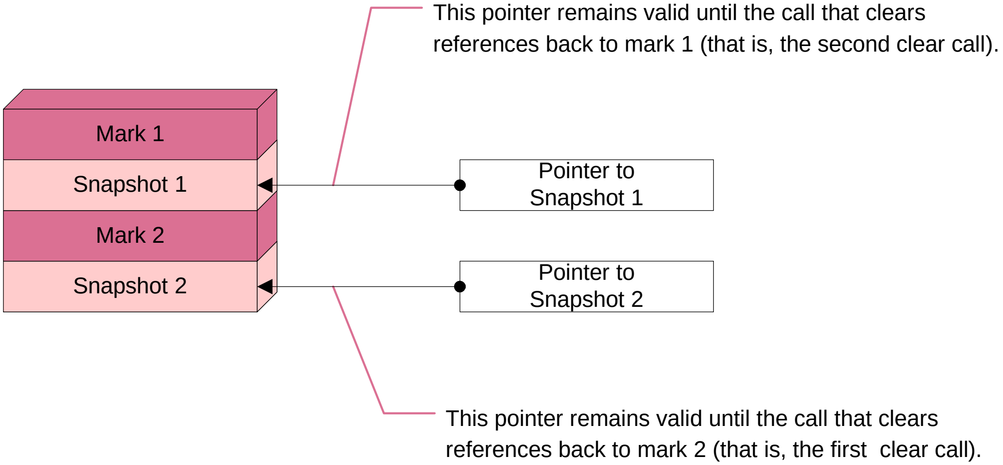
Example This first code fragment exits the loop with 10000 unneeded references to the field named `myFld`:

```
For i = 1 to 10000
 set myFld = msg.getField("fld_name")     'Creates a snapshot refr
 ...myFld.getData()...                    'Uses the reference
 next i
```

This second code fragment clears the reference at the end of each iteration:

```
For i = 1 to 10000
 msg.markReferences
 set myFld = msg.getField("fld_name")     'Creates a snapshot refr
 ...myFld.getData()...                    'Uses the reference
 msg.clearReferences
 next i
```

Figure 3 Mark and Clear References



This pointer remains valid until the call that clears references back to mark 1 (that is, the second clear call).

This pointer remains valid until the call that clears references back to mark 2 (that is, the first clear call).

See Also Reference Datatypes on page 35
Validity of Data Extracted from Message Fields, page 36
Deleting Snapshot References, page 37
TibrvMsg.getField() on page 63
TibrvMsg.clearReferences on page 49

TibrvMsg.remove

Method

Declaration

```
TibrvMsg.remove fieldName, fieldId
```

Purpose

Remove a field from a message.

Parameter	Description
`fieldName`	`String` Remove a field with this name. See Field Search Algorithm. `vbNullString` is a legal field name.
`fieldId`	`Long` optional Remove the field with this identifier. See Field Search Algorithm. An identifier has the range of a 16-bit unsigned integer, [0, 65535].

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.

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.

   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 `vbNullString` as the field name, report the error `TIBRVCOM_NOT_FOUND`.

   If the name search succeeds, but the actual identifier in the field is non-zero (so it does not match the identifier supplied) then report the error `TIBRVCOM_ID_CONFLICT`.

   If the search succeeds, remove the field.

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 `vbNullString`.

If the search succeeds, remove the field.

If the search does not find a field, report the error `TIBRVCOM_NOT_FOUND`.

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

TibrvMsg.removeInstance

Method

Declaration `TibrvMsg.`**`remove`**` fieldName, instance`

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 `i`*th* 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 reports the error `TIBRVCOM_NOT_FOUND`.

Parameter	Description
`fieldName`	`String` Remove a field with this name. `vbNullString` is a legal field name.
`instance`	`Long` Remove this instance of the field. The argument `1` specifies the first instance of the named field.

TibrvMsg.reset

Method

Declaration `TibrvMsg.reset`

Purpose Clear a message, preparing it for re-use.

Remarks This method is the equivalent of destroying the message object and creating a new, except that the new object re-uses the same storage for the outer message object.

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

Snapshot references of the old message are freed. Field objects previously extracted from the old message may contain references into the freed storage.

TibrvMsg.setDefaultInboundEncoding

Method

Declaration `TibrvMsg.setDefaultInboundEncoding encoding`

Purpose Set the global default encoding for inbound messages.

Remarks When an inbound message arrives without an explicit encoding tag, `TibrvMsg.getMsgEncoding()` in the receiving program infers this default encoding.

If no global default is set, `TibrvMsg.getMsgEncoding()` infers an `UNKNOWN` encoding.

Parameter	Description
`encoding`	`String` Set the global default inbound encoding to this value.

See Also TibrvMsg.getMsgEncoding() on page 69
TibrvMsg.getDefaultInboundEncoding() on page 62

TibrvMsg.setOutboundEncoding

Method

Declaration `TibrvMsg.`**`setOutboundEncoding`**` encoding`

Purpose Set the global encoding for outbound messages.

Remarks Outbound messages bear this tag, indicating the encoding of all strings within the message.

When no global encoding is set, outbound messages do not bear any encoding tag.

Parameter	Description
`encoding`	`String` Set the global encoding to this value.

See Also TibrvMsg.getMsgEncoding() on page 69
TibrvMsg.getOutboundEncoding() on page 71

TibrvMsg.setProperty

Method

Declaration `TibrvMsg.`**`setProperty`**` name, value`

Purpose Set a property on a message.

Remarks Properties are supplementary information—they are *not* part of the message itself.

Parameter	Description
`name`	`String` Set the property with this name.
`value`	`Object` Set this property value.

Property	Description
`cmTimeLimit`	`Double` The CM time limit (in whole seconds) of a message. Setting this property overrides the default time limit of the sending CM transport. Fractional values are truncated.

See Also TibrvMsg.getProperty() on page 72
Supplementary Information for Messages on page 41 in *TIBCO Rendezvous Concepts*

TibrvMsg.setReplySubject

Method

Declaration

```
TibrvMsg.setReplySubject replySubject
```

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`	`String` Use this string as the new reply subject, replacing any existing reply subject. The reply subject `vbNullString` removes the previous reply subject. The empty string is *not* a legal subject name.

See Also TibrvMsg.getReplySubject() on page 73
Supplementary Information for Messages on page 41 in *TIBCO Rendezvous Concepts*
Names and Subject-Based Addressing on page 43 in *TIBCO Rendezvous Concepts*

TibrvMsg.setSendSubject

Method

Declaration `TibrvMsg.`**`setSendSubject`**` sendSubject`

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
`sendSubject`	`String` Use this string as the new subject, replacing any existing subject. The subject `vbNullString` removes the previous subject, leaving the message unsendable. The empty string is *not* a legal subject name.

See Also TibrvMsg.getSendSubject() on page 74
Supplementary Information for Messages on page 41 in *TIBCO Rendezvous Concepts*
Names and Subject-Based Addressing on page 43 in *TIBCO Rendezvous Concepts*

TibrvMsg.toString()

Method

Declaration `formattedString = tibrvMsg.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 each field to the output string; however, these two types are exceptions:

TIBRVCOM_MSG_OPAQUE	This method abbreviates the value of an opaque field; for example, [472 opaque bytes].
TIBRVCOM_MSG_XML	This method abbreviates the value of an XML field; for example, [XML document: 472 bytes]. The size measures *un*compressed data.

This method formats TIBRVCOM_MSG_IPADDR32 fields as four dot-separated decimal integers.

This method formats TIBRVCOM_MSG_IPPORT16 fields as one decimal integer.

Parameter	Description
formattedString	String Return a string denoting the message.

Converting Dates to Strings TibrvMsg.toString() prints times in UTC format (also known as Zulu time or GMT). The ISO-8601 standard requires appending a Z character in this notation.

TibrvMsg.toString() uses Common Era numbering for years. This system does not include a year zero. So for example, the time 1 second before 0001-01-01 00:00:00Z would print as -0001-12-31 23:59:59Z.

TibrvMsg.toString() uses a proleptic Gregorian calendar. That is, when formatting dates earlier than the adoption of the Gregorian calendar, it projects the Gregorian calendar backward beyond its actual invention and adoption.

TibrvMsg.update

Method

Declaration `TibrvMsg.`**`update`**` fieldName, data, tibrvType, fieldId`

Purpose Update a field within a message.

Remarks This method copies the new data into the message field.

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 the program may not supply a different field name, field identifier, or datatype.)

If no existing field matches the specifications in the `fieldName` and `fieldId` arguments, then this method adds a new field to the message.

The type of the existing message field and the updating `tibrvType` argument must be identical; otherwise, the method reports the error `TIBRVCOM_INVALID_TYPE`.

When updating, a new array may have a different count (number of elements); a new string, opaque or XML may have different length.

(Sheet 1 of 2)

Parameter	Description
`fieldName`	`String` Update a field with this name. See Field Search Algorithm on page 90. `vbNullString` is a legal name. However, if `fieldId` is non-zero, then `fieldName` must be non-null.
`data`	`Object` Update the field to this data value. Pass the data as any VB type or Rendezvous object type. When this parameter is a `TibrvMsgField` object, this method cannot convert it to another type. To remove a field, use TibrvMsg.remove on page 79.

(Sheet 2 of 2)

Parameter	Description
`tibrvType`	`Byte` optional Update a field with this type. When absent or zero, determine the field’s type from the type of the data. Default encodings and possible conversions are identical to methods that add fields; see Figure 1 on page 47.
`fieldId`	`Long` optional Update the field with this identifier. All field identifiers must be unique within each message. See Field Search Algorithm on page 90. An identifier has the range of a 16-bit unsigned integer, [0, 65535]. Zero is a special default 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

The methods `TibrvMsg.update` and `TibrvMsg.updateField` use this algorithm to find and update a field within a message, as specified by a field identifier and a field name.

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-zero (so it does not match the identifier supplied) then report the error `TIBRVCOM_ID_CONFLICT`.

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

   However, if the program supplied `vbNullString` as the field name, then do not search for the field name; instead, report the error `TIBRVCOM_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 `vbNullString`.

   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 subject address information or certified delivery information.

TibrvMsg.updateField

Method

Declaration `TibrvMsg.`**`updateField`**` field`

Purpose Update a field within a message.

Remarks This method copies the new data into the existing message field.

This method locates a field within the message by matching the name and identifier of the field argument. Then it updates the message field using the `data` of the field object. (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 object's name and identifier, then this method adds a new field to the message.

The type of the existing message field and the type of the updating `field` argument must be identical; otherwise, the method reports the error `TIBRVCOM_INVALID_TYPE`.

When updating, a new array may have a different count (number of elements); a new string, opaque or XML may have different length.

Parameter	Description
`field`	`TibrvMsgField` Update the existing message field using this field.

TibrvMsgField

Class

Purpose Represent a message field.

Remarks Programs use field objects in two situations:

- When a program must repeatedly add similar information to several messages, it may be convenient to initialize a field object with that information, and supply that object to `TibrvMsg.addField` or `TibrvMsg.updateField`.
- When a program must extract information about a message field—such as its field name, field identifier, or datatype—it can extract the field as a `TibrvMsgField` object. A `TibrvMsgField` object contains complete detail about the field.

All `TibrvMsgField` objects *always* belong to your program, so the program is always responsible for deleting the field object.

The validity of the data in a field object depends on its origin:

- When a program creates and initializes a field (by calling `New TibrvMsgField`, and then `TibrvMsgField.initialize`), the field object's data remains valid indefinitely.
- When a program extracts a field object from a message, and the field contains a reference datatype (array, message, opaque, XML, or string), then the field object's data is a reference into the message—so the data remains valid only until the message is destroyed or deleted (or `TibrvMsg.clearReferences` explicitly clears the reference).

  For example, consider a listener callback method that extracts a field from the inbound message. That is, it calls `TibrvMsg.getField()`, and receives a field object as the return value. When the callback method exits, the inbound message is destroyed, and the field object's data reference snapshot becomes invalid.

(Sheet 1 of 2)

Method	Description	Page
`TibrvMsgField.initialize`	Set the contents of a message field object.	99
`TibrvMsgField.getData()`	Extract the data portion of a field object.	95
`TibrvMsgField.getId()`	Extract the field identifier of a field object.	96

(Sheet 2 of 2)

Method	Description	Page
`TibrvMsgField.getName()`	Extract the field name of a field object.	97
`TibrvMsgField.getType()`	Extract the type of a field object.	98

See Also Field Objects, page 35
TibrvMsg.addField on page 48
TibrvMsg.getField() on page 63
TibrvMsg.updateField on page 92

TibrvMsgField.getData()

Method

Declaration

```
data = TibrvMsgField.getData()         'For non-object data.

Set data = TibrvMsgField.getData()     'For object data
```

Purpose Extract the data portion of a field object.

Remarks This method *always* yields an independent snapshot copy of the field data—as does `TibrvMsg.get()`.

Field	Description
`data`	`Object` Return the data content of the field.

⚠ When a field object contains a reference datatype (array, message, opaque, XML, or string), the data in the field object depends on a reference into the parent message. If the parent message is deleted, the program can no longer extract these datatypes from the field object; attempting to extract them references freed storage, which is illegal, and can cause serious consequences.

TibrvMsgField.getId()

Method

Declaration

```
fieldId = TibrvMsgField.getId()
```

Purpose Extract the field identifier of a field object.

Field	Description
`fieldId`	`Long` Return the identifier of the field. An identifier has the range of a 16-bit unsigned integer, [0, 65535]. Zero is a special default value, indicating no identifier.

TibrvMsgField.getName()

Method

Declaration `fieldName = TibrvMsgField.getName()`

Purpose Extract the field name of a field object.

Field	Description
`fieldName`	`String` Return the field name of the field. Names must be strings; `vbNullString` is a special value that indicates no name.

TibrvMsgField.getType()

Method

Declaration `tibrvType = TibrvMsgField.getType()`

Purpose Extract the type of a field object.

Field	Description
`tibrvType`	`Byte` Return the Rendezvous type designator of the field. For a list of types, see Datatype Constants on page 42.

TibrvMsgField.initialize

Method

Declaration `TibrvMsgField.`**`initialize`**` fieldName, data, tibrvType, fieldId`

Purpose Set the contents of a message field object.

Remarks This method 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`).

When a program creates a field object (using the `New` operator), the program owns the object, and can initialize it using this method—even repeatedly. This method discards any old field content, and replaces it with the new content from the method's arguments.

(Sheet 1 of 2)

Parameter	Description
`fieldName`	`String` Store this name in the field object. `vbNullString` is a legal name. However, if `fieldId` is non-zero, then `fieldName` must be non-null.
`data`	`Object` Store this data value in the field object. Pass the data as any VB type or Rendezvous object type (except for `TibrvMsgField`).
`tibrvType`	`Byte` optional Store this explicit type in the field object, and convert the `data` to this type. For a list of types, see Datatype Constants on page 42. When absent or zero, determine the field's type from the type of the object `data`. In Figure 1 on page 47, filled dots indicate default encodings between homologous types.

(Sheet 2 of 2)

Parameter	Description
`fieldId`	`Long` optional Store this identifier in the field object. All field identifiers must be unique within each message. An identifier has the range of a 16-bit unsigned integer, [0, 65535]. When absent or zero, add a field without an identifier. 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.

See Also TibrvMsg.addField on page 48
TibrvMsg.updateField on page 92

TibrvInt64

Class

Purpose Represent a 64-bit integer.

Remarks Inbound messages can contain 64-bit integers, which COM and VB do not directly support. Programs can store signed or unsigned 64-bit values in this object; it is the responsibility of the program to use the high-order bit consistently (either as a sign bit or a value bit).

Certified delivery software attaches 64-bit sequence numbers to messages. `TIBRVCOM` includes this object to represent those sequence numbers.

Rendezvous software does not support arithmetic on this object. However, VB programmers can use the VB currency type (scaled 64-bit integer) to facilitate 64-bit arithmetic; programs must handle scale factor appropriately.

Method	Description	Page
`TibrvInt64.getAsHexString()`	Get the value of a 64-bit integer as a hexidecimal string.	102
`TibrvInt64.getAsString()`	Get the value of a 64-bit integer as a string, interpreting the high order bit as a sign bit.	103
`TibrvInt64.getAsUnsignedString()`	Get the value of a 64-bit integer as a string, interpreting the high order bit as a value bit (rather than a sign bit).	104
`TibrvInt64.getLowOrderPart()`	Get the low-order 32 bits of a 64-bit integer.	105
`TibrvInt64.getHighOrderPart()`	Get the high-order 32 bits of a 64-bit integer.	106
`TibrvInt64.setFromString`	Set the value of a 64-bit integer.	107

See Also TibrvMsg.get() on page 55

TibrvInt64.getAsHexString()

Method

Declaration

```
stringValue = TibrvInt64.getAsHexString()
```

Purpose Get the value of a 64-bit integer as a hexidecimal string.

Remarks Notice that hexidecimal strings avoid confusion regarding the sign bit.

Parameter	Description
`stringValue`	`String` Return the value as a hexidecimal string.

TibrvInt64.getAsString()

Method

Declaration

```
stringValue = TibrvInt64.getAsString()
```

Purpose

Get the value of a 64-bit integer as a string, interpreting the high order bit as a sign bit.

Parameter	Description
`stringValue`	`String` Return the value as a string.

TibrvInt64.getAsUnsignedString()

Method

Declaration

```
stringValue = TibrvInt64.getAsUnsignedString()
```

Purpose Get the value of a 64-bit integer as a string, interpreting the high order bit as a value bit (rather than a sign bit).

Parameter	Description
`stringValue`	`String` Return the value as a string.

TibrvInt64.getLowOrderPart()

Method

Declaration `lowOrderPart = TibrvInt64.`**`getLowOrderPart()`**

Purpose Get the low-order 32 bits of a 64-bit integer.

Remarks If the high order bit of this part is set, the value appears as a negative integer.

Parameter	Description
`lowOrderPart`	`Long` Return the 32 low order bits.

TibrvInt64.getHighOrderPart()

Method

Declaration `highOrderPart = TibrvInt64.`**`getHighOrderPart()`**

Purpose Get the high-order 32 bits of a 64-bit integer.

Remarks If the high order bit of this part is set, the value appears as a negative integer.

Parameter	Description
`lowOrderPart`	`Long` Return the 32 high order bits.

TibrvInt64.setFromString

Method

Declaration `TibrvInt64.setFromString stringValue`

Purpose Set the value of a 64-bit integer.

Remarks This method parses a string of decimal digits, truncating any fractional part.

String values can be in the range [-2^{63}, 2^{64}-1], that is, [-9223372036854775808, +18446744073709551615]. Notice that this range overloads the high-order bit, using it as a sign bit for string values less than zero, and as a value bit for string values greater than 2^{63}-1. It is the responsibility of the program to use the high-order bit consistently (either as a sign bit or a value bit).

Values outside the range [-2^{63}, 2^{64}-1] yield incorrect results, but do not produce errors.

Parameter	Description
`stringValue`	`String` Set the value by parsing this string.

Recommendations

For the type `TIBRVCOM_MSG_I64`, limit values to the range [-9223372036854775808, +9223372036854775807]. Some operating system platforms cannot correctly construct -9223372036854775808; instead, they limit the minimum to -9223372036854775807.

For the type `TIBRVCOM_MSG_U64`, limit values to the range [0, +18446744073709551615].

(Note that extracting this object from a message field does not retain the original type designator.)

Chapter 5 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. A special queue group, called the automatic dispatch queue group, automatically dispatches its queues; we recommend this mechanism, especially for GUI applications.

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

Topics

See Also

Objects, page 4
Callback Methods and Event Handlers, page 6
Tibrv.getAutoDispatchQueueGroup() on page 20

TibrvListener

Class

Purpose Listen for inbound messages.

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

Destroying the queue or transport of a listener automatically destroys the listener as well, but does not delete the COM listener object.

Method	Description	Page
Life Cycle		
`TibrvListener.create`	Create a listener object to listen for inbound messages.	112
`TibrvListener.destroy`	Destroy the internal structure of a listener, canceling interest in its inbound messages.	114
`TibrvListener.isValid()`	Test the validity of a listener.	119
Callback Method		
`TibrvListener_onMsg`	Process inbound messages (listener events).	120
Properties		
`TibrvListener.getClosure()`	Extract the closure from a listener object.	115
`TibrvListener.getQueue()`	Extract the queue from a listener object.	116
`TibrvListener.getSubject()`	Extract the subject from a listener object.	117
`TibrvListener.getTransport()`	Extract the transport from a listener object.	118

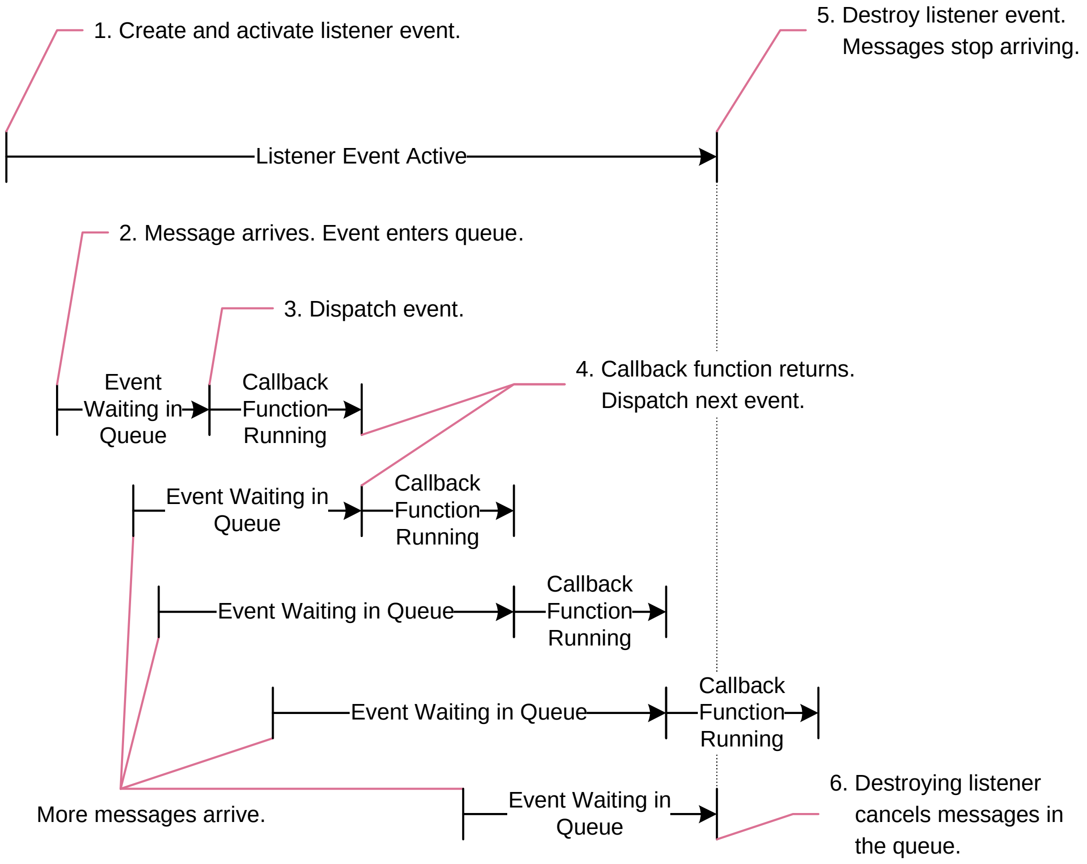
Activation and Dispatch

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

`TibrvListener.create` creates the internal structure of a listener object, and *activates* event processing—that is, it begins listening for all inbound messages with matching subjects. When a message arrives, Rendezvous software places the listener object and message on its queue. Dispatch removes the listener object from the queue, and runs the callback method to process the message. (To stop receiving inbound messages on the subject, destroy or delete the listener object; this action cancels all messages already queued for the listener; see also TibrvListener.destroy on page 114.)

Figure 4 illustrates that messages can continue to accumulate in the queue, even while the callback method is processing.

Figure 4 Listener Activation and Dispatch



TibrvListener.create

Method

Declaration `TibrvListener.create queue, transport, subject, closure`

Purpose Create a listener object to listen for inbound messages.

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

Starting a listener requires three steps:

1. Declare a variable for the listener.

   ```
   Dim WithEvents myListener as TibrvListener
   ```

   This declaration triggers VB to automatically define a private subroutine callback stub named `myListener_OnMsg`. The program must complete this method stub for each listener object.

2. Create the VB object, and assign it to that variable.

   ```
   Set myListener = New TibrvListener
   ```

3. Create the internal implementation object to begin listening.

   ```
   myListener.create myQueue, myTransport, mySubject, 0
   ```

Calling this method on a valid object yields the error `TIBRVCOM_ALREADY_INITIALIZED`.

Parameter	Description
`queue`	`TibrvQueue` For each inbound message, place the listener object on this queue.
`transport`	`TibrvTransport` Listen for inbound messages on this transport.
`subject`	`String` Listen for inbound messages with subjects that match this specification. Wildcard subjects are permitted. The empty string and `vbNullString` are *not* legal subject names.
`closure`	`Object` Store this closure data in the listener 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.create` 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

TibrvListener.destroy on page 114
TibrvListener_onMsg on page 120

TibrvListener.destroy

Method

Declaration `TibrvListener.`**`destroy`**

Purpose Destroy the internal structure of a listener, canceling interest in its inbound messages.

Remarks This method destroys only the internal structure of the listener object. Rendezvous software automatically destroys a listener (using this method) when program execution leaves the scope of the object, or sets the object to `Nothing`.

Destroying a listener object cancels interest in it. Upon return from `TibrvListener.destroy`, the destroyed listener’s message events are no longer dispatched. However, an active callback method of this listener continues to run and return normally, even though the listener is invalid.

It is legal for a listener callback method to destroy its own listener object.

See Also TibrvListener.isValid() on page 119

TibrvListener.getClosure()

Method

Declaration `closure = TibrvListener.getClosure()`

Purpose Extract the closure from a listener object.

Parameter	Description
`closure`	`Object` Return the closure data of the listener object.

TibrvListener.getQueue()

Method

Declaration

```
set queue = TibrvListener.getQueue()
```

Purpose Extract the queue from a listener object.

Parameter	Description
`queue`	`TibrvQueue` Return the queue of the listener object.

TibrvListener.getSubject()

Method

Declaration `subject = TibrvListener.getSubject()`

Purpose Extract the subject from a listener object.

Parameter	Description
`subject`	`String` Return the subject name of the listener object.

TibrvListener.getTransport()

Method

Declaration

```
set transport = TibrvListener.getTransport()
```

Purpose Extract the transport from a listener object.

Parameter	Description
`transport`	`TibrvTransport` Return the transport of the listener object.

TibrvListener.isValid()

Method

Declaration

```
isValid = TibrvListener.isValid()
```

Purpose Test the validity of a listener.

Remarks Notice that `TibrvListener.destroy` invalidates the listener immediately, even though active callback methods may continue to run.

Parameter	Description
`isValid`	`Boolean` Return `True` if the listener is valid. Return `False` if the listener has been destroyed.

See Also TibrvListener.destroy on page 114

TibrvListener_onMsg

Method

Declaration

```
Private Sub myListener_onMsg (
    ByVal listener As TIBRVCOMLib.ITibrvListener,
    ByVal message As TIBRVCOMLib.ITibrvMsg)
```

Purpose Process inbound messages (listener events).

Remarks Implement this method to process inbound messages.

This method receives its parameters by value, so it cannot delete them.

To stop listening to the message subject, destroy the *original* listener object associated with this callback method.

Declaring a variable of class `TibrvListener` with events triggers VB to automatically define a COM event callback, and a private subroutine callback stub named *var*`_OnMsg`. The program must complete this method for each listener object.

Parameter	Description
`listener`	`TibrvListener` This parameter receives the listener object.
`message`	`TibrvMsg` This parameter receives the inbound message. To preserve this message beyond exit from this callback method, the program must set a persistent reference to it, and detach it.

CM Label Information The callback method for certified delivery messages can use the certified delivery label property `cmSender` to discriminate these situations:

- If extracting the `cmSender` property yields the error `TIBRVCOM_NOT_FOUND`, then the message uses the reliable protocol (that is, it was sent from an ordinary transport).
- If the `cmSender` property is a valid sender name, then the message uses the certified delivery protocol (that is, it is a labeled message, sent from a certified delivery transport).

See Also TibrvMsg.detach on page 54
TibrvCmListener.create on page 227
TibrvMsg.getProperty() on page 72

TibrvTimer

Class

Purpose Timer event.

Remarks All timers are repeating timers. To simulate a once-only timer, code the callback method to destroy the timer. To stop the timer, destroy or delete it.

Destroying the queue of a timer automatically destroys the timer as well. However, the program must also delete the timer object to reclaim resources.

Method	Description	Page
Life Cycle		
`TibrvTimer.create`	Start a timer.	124
`TibrvTimer.destroy`	Destroy a timer event.	126
`TibrvTimer.isValid()`	Test the validity of a timer.	130
Callback Method		
`TibrvTimer_onTimer`	Process timer events.	131
Properties		
`TibrvTimer.getClosure()`	Extract the closure from a timer object.	127
`TibrvTimer.getInterval()`	Extract the interval from a timer object.	128
`TibrvTimer.getQueue()`	Extract the queue from a timer object.	129
`TibrvTimer.resetInterval`	Reset the interval of a timer object.	132

Activation and Dispatch

`TibrvTimer.create` creates the internal structure of a timer 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 TibrvTimer.destroy on page 126.)

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.

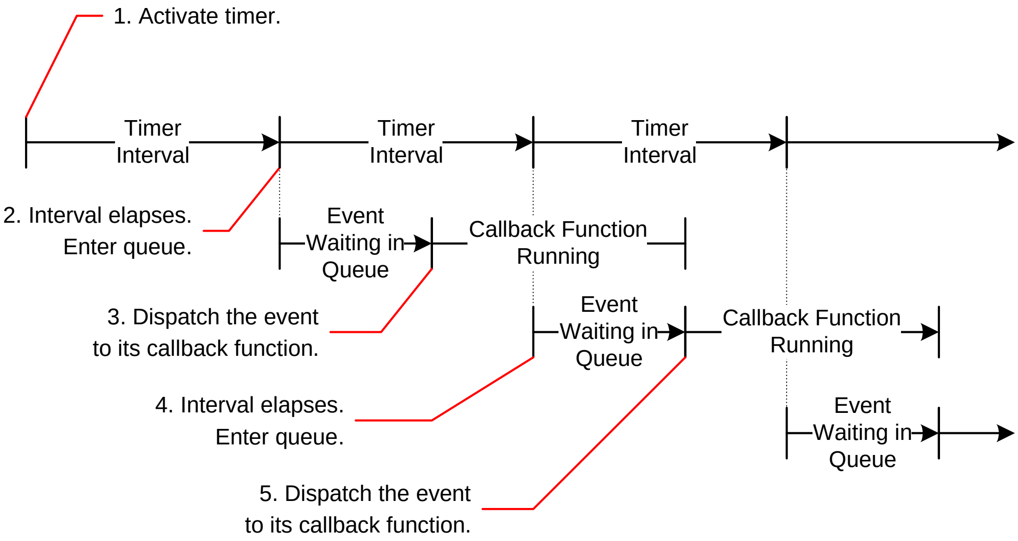
Figure 5 illustrates a sequence of timer intervals. The number of elapsed timer intervals directly determines the number of event callbacks.

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 5 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 VB, hardware and operating system constraints.

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 on page 114 in *TIBCO Rendezvous Concepts*.

TibrvTimer.create

Method

Declaration

```
TibrvTimer.create queue, interval, closure
```

Purpose Start a timer.

Remarks All timers are repeating timers. To simulate a once-only timer, code the callback method to destroy the timer. To stop a timer, destroy or delete it.

Starting a timer requires three steps:

1. Declare a variable for the timer.

   ```
   Dim WithEvents myTimer as TibrvTimer
   ```

   This declaration triggers VB to automatically define a private subroutine callback stub named `myTimer_OnTimer`. The program must complete this method for each timer object.

2. Create the VB object, and assign it to that variable.

   ```
   Set myTimer = New TibrvTimer
   ```

3. Create the internal implementation object to begin counting.

   ```
   myTimer.create myQueue, myInterval, 0
   ```

Calling this method on a valid object yields the error `TIBRVCOM_ALREADY_INITIALIZED`.

Parameter	Description
`queue`	`TibrvQueue` At each time interval, place the timer object on this queue.
`interval`	`Double` The timer triggers its callback method at this repeating interval (in seconds).
`closure`	`Object` Store this closure data in the timer 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 VB, hardware and operating system constraints.

See Also TibrvTimer.destroy on page 126
TibrvTimer_onTimer on page 131

TibrvTimer.destroy

Method

Declaration `TibrvTimer.destroy`

Purpose Destroy a timer event.

Remarks This method destroys only the internal structure of the timer object. Rendezvous software automatically destroys a timer (using this method) when program execution leaves the scope of the object, or deletes the object.

Destroying a timer object cancels interest in it. Upon return from `TibrvTimer.destroy`, the destroyed timer is no longer dispatched. However, an active callback method of this timer continues to run and return normally, even though the timer object is invalid.

It is legal for a timer callback method to destroy its own timer object.

See Also TibrvTimer.isValid() on page 130

TibrvTimer.getClosure()

Method

Declaration `closure = TibrvTimer.getClosure()`

Purpose Extract the closure from a timer object.

Parameter	Description
`closure`	`Object` Return the closure data of the timer object.

TibrvTimer.getInterval()

Method

Declaration `interval = TibrvTimer.getInterval()`

Purpose Extract the interval from a timer object.

Parameter	Description
`interval`	`Double` Return the interval of the timer object.

TibrvTimer.getQueue()

Method

Declaration `set queue = TibrvTimer.getQueue()`

Purpose Extract the queue from a timer object.

Parameter	Description
`queue`	`TibrvQueue` Return the queue of the timer object.

TibrvTimer.isValid()

Method

Declaration `isValid = TibrvTimer.isValid()`

Purpose Test the validity of a timer.

Remarks Notice that `TibrvTimer.destroy` invalidates the timer immediately, even though active callback methods may continue to run.

Parameter	Description
`isValid`	`Boolean` Return `True` if the timer is valid. Return `False` if the timer has been destroyed.

See Also TibrvTimer.destroy on page 126

TibrvTimer_onTimer

Method

Declaration

```
Private Sub myTimer_onTimer (
    ByVal timer As TIBRVCOMLib.ITibrvTimer)
```

Purpose

Process timer events.

Remarks

Declaring a variable of class `TibrvTimer` with events triggers VB to automatically define a COM event callback, and a private subroutine callback stub named *var*`_OnTimer`. The program must complete this method for each timer object.

This method receives its parameters by value, so it cannot delete them.

To stop the timer, destroy the *original* timer object associated with this callback method.

Parameter	Description
`timer`	`TibrvTimer` This parameter receives the timer object.

TibrvTimer.resetInterval

Method

Declaration `TibrvTimer.`**`resetInterval`**` newInterval`

Purpose Reset the interval of a timer object.

Remarks The timer begins counting the new interval immediately.

Parameter	Description
`newInterval`	`Double` 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 for over 100 years. The actual granularity of intervals depends on VB, 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 a timer 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.)

TibrvQueue

Class

Purpose Event queue.

Remarks Each listener and timer object is associated with a TibrvQueue object; when the event occurs, Rendezvous software places the object in its queue. Programs dispatch queues to process events.

Default Queue Programs that need only one event queue can use the default queue (instead of using TibrvQueue.create 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. Programs cannot change the parameters of the default queue.

For more information, see Tibrv.getDefaultQueue() on page 21.

Priority 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.

Automatic Dispatch Queue Group A special queue group, called the automatic dispatch queue group, detects the presence of an event in any of its queues, and automatically dispatches the queues (according to their priority) until no more events are waiting. To automatically dispatch a queue, add it to that queue group. To retrieve that queue group, see Tibrv.getAutoDispatchQueueGroup() on page 20.

Limit Policy The type library defines these constants, which specify the possible strategies for resolving overflow of queue limit.

(Sheet 1 of 2)

Constant	Description
TIBRVCOM_QUEUE_DEFAULT_POLICY TIBRVCOM_QUEUE_DISCARD_NONE	Never discard events; use this policy when a queue has no limit on then number of events it can contain.
TIBRVCOM_QUEUE_DISCARD_FIRST	Discard the first event in the queue (that is, the oldest event in the queue, which would otherwise be the next event to dispatch).

(Sheet 2 of 2)

Constant	Description
`TIBRVCOM_QUEUE_DISCARD_LAST`	Discard the last event in the queue (that is, the youngest event in the queue).
`TIBRVCOM_QUEUE_DISCARD_NEW`	Discard the new event (which would otherwise cause the queue to overflow its maximum events limit).

(Sheet 1 of 2)

Method	Description	Page
Life Cycle		
`TibrvQueue.create`	Create an event queue.	136
`TibrvQueue.destroy`	Destroy an event queue.	137
`TibrvQueue.isValid()`	Test the validity of a queue.	145
Dispatch		
`TibrvQueue.dispatch`	Dispatch an event; if no event is ready, block.	138
`TibrvQueue.poll`	Dispatch an event, if possible.	146
`TibrvQueue.timedDispatch`	Dispatch an event, but if no event is ready to dispatch, limit the time that this call blocks while waiting for an event.	150
Properties		
`TibrvQueue.getCount()`	Extract the number of events in a queue.	139
`TibrvQueue.getDiscardAmount()`	Extract the discard amount of a queue.	140
`TibrvQueue.getLimitPolicy()`	Extract the limit policy of a queue.	141
`TibrvQueue.getMaxEvents()`	Extract the maximum event limit of a queue.	142
`TibrvQueue.getName()`	Extract the name of a queue.	143
`TibrvQueue.getPriority()`	Extract the priority of a queue.	144

(Sheet 2 of 2)

Method	Description	Page
`TibrvQueue.setName`	Set the name of a queue.	148
`TibrvQueue.setLimitPolicy`	Set the limit properties of a queue.	147
`TibrvQueue.setPriority`	Set the priority of a queue.	149

See Also TibrvQueueGroup on page 151

TibrvQueue.create

Method

Declaration `TibrvQueue.`**`create`**

Purpose Create an event queue.

Remarks Upon creation, new queues set these default values.

Property	Default Value	Set Method
`limitPolicy`	`TIBRVCOM_QUEUE_DISCARD_NONE`	TibrvQueue.setLimitPolicy on page 147
`maxEvents`	zero (unlimited)	
`discardAmount`	zero	
`name`	`tibrvQueue`	TibrvQueue.setName on page 148
`priority`	1	TibrvQueue.setPriority on page 149

Calling this method on a valid object yields the error `TIBRVCOM_ALREADY_INITIALIZED`.

TibrvQueue.destroy

Method

Declaration `TibrvQueue.`**`destroy`**

Purpose Destroy an event queue.

Remarks This method destroys the internal mechanism of the queue, leaving an unusable object. Rendezvous software automatically destroys a queue (using this method) when program execution leaves the scope of the object, or the program deletes the object.

When a queue is destroyed, events that remain in the queue are discarded. Dispatch calls report an error. Automatic dispatch ignores the destroyed queue.

It is safe to call `TibrvQueue.destroy` more than once on the same object. Repeat calls return without error.

Programs cannot destroy the default queue.

TibrvQueue.dispatch

Method

Declaration `TibrvQueue.`**`dispatch`**

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.

The Rendezvous component does not trigger dispatch while a program callback method is already running. When a program attempts to explicitly dispatch (in violation of this protective rule), the dispatch method reports the error `TIBRVCOM_EVENT_CALLBACK_ACTIVE`.

⚠ Blocking effectively freezes the GUI operation of a program. Use with caution.

We recommend automatic dispatch instead.

See Also Tibrv.getAutoDispatchQueueGroup() on page 20
TibrvQueue.poll on page 146
TibrvQueue.timedDispatch on page 150

TibrvQueue.getCount()

Method

Declaration

```
numEvents = tibrvQueue.getCount()
```

Purpose Extract the number of events in a queue.

Parameter	Description
`numEvents`	`Long` Return the number of events in the queue.

TibrvQueue.getDiscardAmount()

Method

Declaration `discardAmount = TibrvQueue.`**`getDiscardAmount`**`()`

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.

Parameter	Description
`discardAmount`	`Long` Return the discard amount of the queue.

See Also TibrvQueue.setLimitPolicy on page 147

TibrvQueue.getLimitPolicy()

Method

Declaration `limitPolicy = TibrvQueue.getLimitPolicy()`

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 on page 133.

Parameter	Description
`limitPolicy`	`Long` Return the limit policy of the queue.

See Also TibrvQueue.setLimitPolicy on page 147

TibrvQueue.getMaxEvents()

Method

Declaration

```
maxEvents = TibrvQueue.getMaxEvents()
```

Purpose

Extract the maximum event limit of a queue.

Remarks

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

Zero specifies an unlimited number of events.

Parameter	Description
`maxEvents`	`Long` Return the maximum event limit of the queue.

See Also

TibrvQueue.setLimitPolicy on page 147

TibrvQueue.getName()

Method

Declaration `queueName = TibrvQueue.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.

Parameter	Description
`queueName`	`String` Return the name of the queue.

See Also TibrvQueue.setName on page 148

TibrvQueue.getPriority()

Method

Declaration `priority = Tibrvqueue.getPriority()`

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.

Parameter	Description
`priority`	`Long` Return the priority of the queue.

See Also TibrvQueue.setPriority on page 149

TibrvQueue.isValid()

Method

Declaration `isValid = TibrvQueue.`**`isValid`**`()`

Purpose Test the validity of a queue.

Parameter	Description
`isValid`	`Boolean` Return `True` if the queue is valid. Return `False` if the queue has been destroyed.

See Also TibrvQueue.destroy on page 137

TibrvQueue.poll

Method

Declaration `TibrvQueue.`**`poll`**

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, reporting the error `TIBRVCOM_TIMEOUT`.

This call is equivalent to `timedDispatch(0.0)`.

The Rendezvous component does not trigger dispatch while a program callback method is already running. When a program attempts to explicitly dispatch (in violation of this protective rule), the dispatch method reports the error `TIBRVCOM_EVENT_CALLBACK_ACTIVE`.

See Also Tibrv.getAutoDispatchQueueGroup() on page 20
TibrvQueue.dispatch on page 138
TibrvQueue.timedDispatch on page 150

TibrvQueue.setLimitPolicy

Method

Declaration `TibrvQueue.`**`setLimitPolicy`**` limitPolicy, maxEvents, discardAmount`

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`	`Long` 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 Limit Policy on page 133. When `maxEvents` is zero (unlimited), the `policy` must be `TIBRVCOM_QUEUE_DISCARD_NONE`.
`maxEvents`	`Long` Programs can limit the number events that a queue can hold—either to curb queue growth, or implement a specialized dispatch semantics. Zero specifies an unlimited number of events; in this case, the `policy` must be `TIBRVCOM_QUEUE_DISCARD_NONE`.
`discardAmount`	`Long` When the queue exceeds its maximum event limit, discard a block of events. This property specifies the number of events to discard. When `discardAmount` is zero, the `policy` must be `TIBRVCOM_QUEUE_DISCARD_NONE`.

See Also TibrvQueue.getDiscardAmount() on page 140
TibrvQueue.getLimitPolicy() on page 141
TibrvQueue.getMaxEvents() on page 142

TibrvQueue.setName

Method

Declaration `TibrvQueue.`**`setName`**` queueName`

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`	`String` Replace the name of the queue with this new name. It is illegal to supply `vbNullString` as the new queue name. However, the empty string is a legal queue name.

See Also TibrvQueue.getName() on page 143

TibrvQueue.setPriority

Method

Declaration `TibrvQueue.`**`setPriority`**` priority`

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`	`Long` 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() on page 144

TibrvQueue.timedDispatch

Method

Declaration `TibrvQueue.timedDispatch timeout`

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, reporting the error `TIBRVCOM_TIMEOUT`.

The Rendezvous component does not trigger dispatch while a program callback method is already running. When a program attempts to explicitly dispatch (in violation of this protective rule), the dispatch method reports the error `TIBRVCOM_EVENT_CALLBACK_ACTIVE`.

Parameter	Description
`timeout`	`Double` 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.

⚠ Blocking effectively freezes the GUI operation of a program. Use with caution.

We recommend automatic dispatch instead.

Constants The type library defines these constants for use with timed dispatch methods.

Constant	Value
`TIBRVCOM_WAIT_FOREVER`	`-1`
`TIBRVCOM_NO_WAIT`	`0`

See Also Tibrv.getAutoDispatchQueueGroup() on page 20
TibrvQueue.dispatch on page 138
TibrvQueue.poll on page 146

TibrvQueueGroup

Class

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.

Priority 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.

Automatic Dispatch Queue Group A special queue group, called the automatic dispatch queue group, detects the presence of an event in any of its queues, and automatically dispatches the queues (according to their priority) until no more events are waiting. To automatically dispatch a queue, add it to that queue group. To stop automatic dispatch, remove all queues from that group. To retrieve that queue group, see Tibrv.getAutoDispatchQueueGroup() on page 20.

(Sheet 1 of 2)

Method	Description	Page
Life Cycle		
`TibrvQueueGroup.create`	Create an event queue group.	154
`TibrvQueueGroup.destroy`	Destroy an event queue group.	155
`TibrvQueueGroup.isValid()`	Test the validity of a queue group.	157
Queues		
`TibrvQueueGroup.add`	Add an event queue to a queue group.	153
`TibrvQueueGroup.remove`	Remove an event queue from a queue group.	159
Dispatch		
`TibrvQueueGroup.dispatch`	Dispatch an event from a queue group; if no event is ready, block.	156
`TibrvQueueGroup.poll`	Dispatch an event, but if no event is ready to dispatch, return immediately (without blocking).	158

(Sheet 2 of 2)

Method	Description	Page
`TibrvQueueGroup.timedDispatch`	Dispatch an event, but if no event is ready to dispatch, limit the time that this call blocks while waiting for an event.	160

See Also Tibrv.getAutoDispatchQueueGroup() on page 20
TibrvQueue on page 133

TibrvQueueGroup.add

Method

Declaration `TibrvQueueGroup.`**`add`**` queue`

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 reports an error.

Parameter	Description
`queue`	`TibrvQueue` Add this event to the queue group.

See Also TibrvQueue on page 133

TibrvQueueGroup.create

Method

Declaration `TibrvQueueGroup.`**`create`**

Purpose Create an event queue group.

Remarks The new queue group is empty.

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

Calling this method on a valid object yields the error `TIBRVCOM_ALREADY_INITIALIZED`.

See Also TibrvQueueGroup.add on page 153
TibrvQueueGroup.destroy on page 155

TibrvQueueGroup.destroy

Method

Declaration `TibrvQueueGroup.`**`destroy`**

Purpose Destroy an event queue group.

Remarks This method destroys the internal mechanism of the group, leaving an unusable object. Rendezvous software automatically destroys a queue group (using this method) when program execution leaves the scope of the object, or sets the object to `Nothing`.

The individual queues in the group continue to exist, even though the group has been destroyed.

Programs cannot destroy the automatic dispatch queue group. Attempts to destroy it return without error.

See Also TibrvQueueGroup.create on page 154

TibrvQueueGroup.dispatch

Method

Declaration `TibrvQueueGroup.`**`dispatch`**

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.

The Rendezvous component does not trigger dispatch while a program callback method is already running. When a program attempts to explicitly dispatch (in violation of this protective rule), the dispatch method reports the error `TIBRVCOM_EVENT_CALLBACK_ACTIVE`.

⚠ Blocking effectively freezes the GUI operation of a program. Use with caution.

We recommend automatic dispatch instead.

See Also TibrvQueueGroup.timedDispatch on page 160
TibrvQueueGroup.poll on page 158

TibrvQueueGroup.isValid()

Method

Declaration `isValid = TibrvQueueGroup.`**`isValid`**`()`

Purpose Test the validity of a queue group.

Parameter	Description
`isValid`	`Boolean` Return `True` if the queue group is valid. Return `False` if the queue group has been destroyed.

See Also TibrvQueueGroup.destroy on page 155

TibrvQueueGroup.poll

Method

Declaration `TibrvQueueGroup.`**`poll`**

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, then this call returns immediately, reporting the error `TIBRVCOM_TIMEOUT`.

When searching the group for a non-empty queue, this call searches according the to 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.

This call is equivalent to `timedDispatch(0.0)`.

The Rendezvous component does not trigger dispatch while a program callback method is already running. When a program attempts to explicitly dispatch (in violation of this protective rule), the dispatch method reports the error `TIBRVCOM_EVENT_CALLBACK_ACTIVE`.

See Also TibrvQueueGroup.dispatch on page 156
TibrvQueueGroup.timedDispatch on page 160

TibrvQueueGroup.remove

Method

Declaration `TibrvQueueGroup.`**`remove`**` queue`

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 reports the error `TIBRVCOM_INVALID_QUEUE`.

To stop automatic dispatch of a queue, remove it from the automatic dispatch queue group.

Parameter	Description
`queue`	`TibrvQueue` Remove this queue from a queue group.

See Also TibrvQueue on page 133

TibrvQueueGroup.timedDispatch

Method

Declaration `TibrvQueueGroup.`**`timedDispatch`**` timeout`

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, reporting the error `TIBRVCOM_TIMEOUT`.

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.

The Rendezvous component does not trigger dispatch while a program callback method is already running. When a program attempts to explicitly dispatch (in violation of this protective rule), the dispatch method reports the error `TIBRVCOM_EVENT_CALLBACK_ACTIVE`.

Parameter	Description
`timeout`	`Double` 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.

⚠ Blocking effectively freezes the GUI operation of a program. Use with caution.

We recommend automatic dispatch instead.

Constants The type library defines these constants for use with timed dispatch methods.

Constant	Value
`TIBRVCOM_WAIT_FOREVER`	`-1`
`TIBRVCOM_NO_WAIT`	`0`

See Also TibrvQueueGroup.dispatch on page 156
TibrvQueueGroup.poll on page 158

Chapter 6 Transports

Transports manage network connections and send outbound messages. Listeners (and other objects) us transports for communications.

This chapter presents the various transport classes and their methods.

Topics

See Also

Virtual Circuits on page 183
TibrvCmTransport on page 238
Distributed Queue Transport on page 276

TibrvTransport

Class

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. Listeners (and other objects) use transports for communications.

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, invalidates any listeners using that transport.

(Sheet 1 of 2)

Method	Description	Page
Life Cycle		
`TibrvTransport.create`	Create a network transport.	166
`TibrvTransport.destroy`	Destroy a transport.	170
`TibrvTransport.isValid()`	Test the validity of a transport.	175
Inbox Names		
`TibrvTransport.createInbox()`	Create a unique inbox subject name.	169
Send Messages		
`TibrvTransport.send`	Send a message.	176
`TibrvTransport.sendReply`	Send a reply message.	177
`TibrvTransport.sendRequest()`	Send a request message and wait for a reply.	178
Accessors		
`TibrvTransport.getDaemon()`	Return the TCP socket where this transport connects to the Rendezvous daemon.	171

(Sheet 2 of 2)

Method	Description	Page
`TibrvTransport.getDescription()`	Extract the program description parameter from a transport.	172
`TibrvTransport.getNetwork()`	Return the network interface that this transport uses for communication.	173
`TibrvTransport.getService()`	Return the effective service that this transport uses for communication.	174
`TibrvTransport.setDescription`	Set the program description parameter of a transport.	181
Virtual Circuits		
`TibrvTransport.createAcceptVc()`	Create a virtual circuit accept object.	186
`TibrvTransport.createConnectVc()`	Create a virtual circuit connect object.	187
`TibrvTransport.getVcConnectSubject()`	Return the connect subject of an accept terminal.	189
`TibrvTransport.waitForVcConnection`	Test the connection status of a virtual circuit.	190

See Also Chapter 7, Virtual Circuits, on page 183
Transport on page 99 in *TIBCO Rendezvous Concepts*

TibrvTransport.create

Method

Declaration

```
TibrvTransport.create service, network, daemon

TibrvTransport.CreateLicensed service, network, daemon,
licenseTicket
```

Purpose

Create a network transport.

Connecting to the Rendezvous Daemon

Rendezvous daemon processes do the work of moving messages across a network. Every network transport must connect to a Rendezvous daemon. (The intra-process transport does not connect to a 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 this method cannot connect to the Rendezvous daemon, it reports the error `TIBRVCOM_DAEMON_NOT_CONNECTED`.

The first time a program successfully connects to the Rendezvous daemon process, `rvd` starts the clock ticking for temporary license tickets. (See Licensing Information, page 11 in *TIBCO Rendezvous Administration*.)

Calling this method on a valid object yields the error `TIBRVCOM_ALREADY_INITIALIZED`.

(Sheet 1 of 2)

Parameter	Description
`service`	`String` The Rendezvous daemon divides the network into logical partitions. Each network transport communicates on a single service; a transport can communicate only with other transports on the same service. To communicate on more than one service, a program must create more than one transport—one transport for each service. You can specify the service in several ways. For details, see Service Parameter on page 103 in *TIBCO Rendezvous Concepts*. The empty string and `vbNullString` both specify the default `rendezvous` service.

(Sheet 2 of 2)

Parameter	Description
`network`	`String` 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. To communicate over more than one network, programs must create more than one transport. You can specify the network in several ways. For details, see Network Parameter on page 107 in *TIBCO Rendezvous Concepts*. The empty string and `vbNullString` both specify the primary network interface for the host computer.
`daemon`	`String` The daemon parameter instructs the transport object about how and where to find the Rendezvous daemon and establish communication. For details, see Daemon Parameter on page 110 in *TIBCO Rendezvous Concepts*. You can specify a daemon on a remote computer. For details, see Remote Daemon on page 111 in *TIBCO Rendezvous Concepts*. If you specify a secure daemon, this string must be identical to as the `daemonName` argument of TibrvSdContext.setDaemonCert() on page 27. See also, Secure Daemon on page 111 in *TIBCO Rendezvous Concepts*. The empty string and `vbNullString` both specify the default—find the local daemon on TCP socket `7500`. (These defaults are not valid when the local daemon is a secure daemon.)
`licenseTicket`	`String` Embed this special license ticket in the transport object. When a licensed transport connects to `rvd`, it presents this special ticket to validate its connection (`rvd` uses the longest-running ticket available, which can be either this special ticket, or a ticket from the ticket file, `tibrv.tkt`). Ordinary license tickets are *not* valid for this parameter; see also, Embedded License on page 168.

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 on page 181.

Embedded License

Specially-licensed third-party developers can use the second form of this method. To use this alternate form, a developer must first purchase a special license ticket. This call embeds the special ticket in the program, so that end-users do not need to purchase Rendezvous to use the program.

To purchase an embedded license, contact TIBCO Software Inc.

See Also

TibrvTransport.getDaemon() on page 171
TibrvTransport.getNetwork() on page 173
TibrvTransport.getService() on page 174

TibrvTransport.createInbox()

Method

Declaration

```
inboxSubject = TibrvTransport.createInbox()
```

Purpose Create a unique inbox subject name.

Parameter	Description
`inboxSubject`	`String` Return the new 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 throughout 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.create`. The inbox name is only valid for use with the same transport that created it. When calling `TibrvListener.create`, 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.

Calling this method on a valid object yields the error `TIBRVCOM_ALREADY_INITIALIZED`.

This method is the *only* legal way for programs to create inbox subject names.

See Also TibrvMsg.setReplySubject on page 86

TibrvTransport.destroy

Method

Declaration `TibrvTransport.`**`destroy`**

Purpose Destroy a transport.

Remarks This method destroys the internal mechanism of the transport, leaving an unusable object. Rendezvous software automatically destroys a transport (using this method) when program execution leaves the scope of the object, or sets the object to `Nothing`.

Destroying a transport achieves these effects:

- The transport flushes all outbound data to the Rendezvous daemon.

  This effect is especially important.
- The transport invalidates all its listeners.
- Subsequent calls to other methods that use the destroyed transport report an error (usually `TIBRVCOM_OBJECT_NOT_INITIALIZED`).

Storage for the transport object is freed when all listeners that use it have been deleted.

See Also TibrvTransport.isValid() on page 175

TibrvTransport.getDaemon()

Method

Declaration `daemon = TibrvTransport.getDaemon()`

Purpose Return the TCP socket where this transport connects to the Rendezvous daemon.

Remarks It is an error to call this method on the intra-process transport.

Parameter	Description
`daemon`	`String` Return the daemon parameter of the network transport.

TibrvTransport.getDescription()

Method

Declaration

```
description = TibrvTransport.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.

It is an error to call this method on the intra-process transport.

Parameter	Description
`description`	`String` Return the description of the transport.

See Also TibrvTransport on page 164
TibrvTransport.setDescription on page 181

TibrvTransport.getNetwork()

Method

Declaration `network = TibrvTransport.getNetwork()`

Purpose Return the network interface that this transport uses for communication.

Remarks It is an error to call this method on the intra-process transport.

Parameter	Description
`network`	`String` Return the network parameter of the network transport.

TibrvTransport.getService()

Method

Declaration `service = TibrvTransport.getService()`

Purpose Return the effective service that this transport uses for communication.

Remarks It is an error to call this method on the intra-process transport.

Parameter	Description
`service`	`String` Return the service parameter of the network transport.

TibrvTransport.isValid()

Method

Declaration `isValid = TibrvTransport.`**`isValid`**`()`

Purpose Test the validity of a transport.

Parameter	Description
`isValid`	`Boolean` Return `True` if the transport is valid. Return `False` if the transport has been destroyed.

See Also TibrvTransport.destroy on page 170

TibrvTransport.send

Method

Declaration `TibrvTransport.send message`

Purpose Send a message.

Remarks The message must have a valid destination subject; see TibrvMsg.setSendSubject on page 87.

Parameter	Description
`message`	`TibrvMsg` Send this message.

See Also TibrvMsg.setSendSubject on page 87

TibrvTransport.sendReply

Method

Declaration `TibrvTransport.`**`sendReply`**` replyMsg, requestMsg`

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`	`TibrvMsg` Send this *outbound* reply message.
`requestMessage`	`TibrvMsg` Send a reply to this *inbound* request message; extract its reply subject to use as the subject of the outbound reply message.

⚠ 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() on page 73
TibrvTransport.sendRequest() on page 178

TibrvTransport.sendRequest()

Method

Declaration

```
set replyMessage = TibrvTransport.sendRequest(message, timeout)
```

Purpose Send a request message and wait for a reply.

⚠ This call blocks all other activity in the program, including event dispatch and GUI operation. Use with caution.

Parameter	Description
`message`	`TibrvMsg` Send this message.
`timeout`	`Double` Maximum time (in seconds) that this call can block while waiting for a reply. -1 indicates no timeout (wait without limit for a reply).
`replyMessage`	`TibrvMsg` Return the reply message when it arrives.

Remarks 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 on page 87.

The program owns the reply message object, and must delete it to reclaim its storage.

Operation This method operates in several synchronous steps:

1. Create an inbox name, and a listener object 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, report the error `TIBRVCOM_TIMEOUT`. (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.setBatchMode

Method

Declaration

```
TibrvTransport.setBatchMode mode
```

Purpose Set the batch mode parameter of a transport.

Remarks 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

It is an error to call this method on the intra-process transport.

Parameter	Description
`mode`	Use this value as the new batch mode.

Constants The type library defines these constants for use with this method.

Constant	Description
`TIBRVCOM_TRANSPORT_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.
`TIBRVCOM_TRANSPORT_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 TibrvTransport on page 164
Batch Modes for Transports on page 118 in *TIBCO Rendezvous Concepts*

TibrvTransport.setDescription

Method

Declaration `TibrvTransport.setDescription description`

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.

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.

It is an error to call this method on the intra-process transport.

Parameter	Description
`description`	`String` Use this string as the new program description.

See Also TibrvTransport on page 164
TibrvTransport.getDescription() on page 172

Chapter 7 Virtual Circuits

Virtual circuits feature Rendezvous communication between two terminals over an exclusive, continuous, monitored connection.

See Also Virtual Circuits on page 119 in *TIBCO Rendezvous Concepts*

Topics

Virtual Circuit Transport

A virtual circuit transport object represents a terminal in a potential circuit.

Virtual circuit terminals have the same datatype as ordinary transport objects, and can fill the same roles as ordinary transports. Programs can use them to create inbox names, send messages, create listeners and other events.

Two create methods 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 types of terminal 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.

Both create methods are methods of ordinary transport objects. Calling one of these methods on an ordinary transport creates a new object (the terminal), which employs the ordinary transport for network communications. Programs may use the ordinary transport for other purposes.

Method	Description	Page
`TibrvTransport.createAcceptVc()`	Create a virtual circuit accept object.	186
`TibrvTransport.createConnectVc()`	Create a virtual circuit connect object.	187
`TibrvTransport.getVcConnectSubject()`	Return the connect subject of an accept terminal.	189
`TibrvTransport.waitForVcConnection`	Test the connection status of a virtual circuit.	190

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 272 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.

Destroying VC Transports

Programs must explicitly destroy each virtual circuit transport object. Destroying a transport object precludes subsequent communications on that transport, and frees its storage. Attempting to use a destroyed transport in any way is an error.

Destroying a virtual circuit transport does *not* affect the ordinary transport that the terminal employs.

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 on page 116 in *TIBCO Rendezvous Concepts*.

For programming details, see Specifying Direct Communication on page 105 in *TIBCO Rendezvous Concepts*.

Inherited Methods		Notes

Legal Methods	`TibrvTransport.createInbox()`
	`TibrvTransport.destroy`
	`TibrvTransport.send`
	`TibrvTransport.sendReply`
	`TibrvTransport.sendRequest()`	

Disabled Methods	`TibrvTransport.createAcceptVc()`
	`TibrvTransport.createConnectVc()`
	`TibrvTransport.getDaemon()`
	`TibrvTransport.getDescription()`
	`TibrvTransport.getNetwork()`
	`TibrvTransport.getService()`
	`TibrvTransport.setBatchMode`
	`TibrvTransport.setDescription`
		After calling either of the two create methods on an ordinary transport (to create the terminal) these methods are disabled in the terminal.

See Also TibrvTransport on page 164
Virtual Circuits on page 119 in *TIBCO Rendezvous Concepts*

TibrvTransport.createAcceptVc()

Method

Declaration `set vcTransport = TibrvTransport.createAcceptVc()`

Purpose Create a virtual circuit accept object.

Remarks Note that programs call this method on an ordinary transport. The method creates and returns a new accept transport object that employs the ordinary transport. Programs may use the ordinary transport for other purposes.

It is illegal to call this method on a virtual circuit terminal transport object (that is, you cannot nest a virtual circuit within another virtual circuit).

After this method returns, programs must extract the connect subject; see TibrvTransport.getVcConnectSubject() on page 189.

Parameter	Description
`vcTransport`	`TibrvTransport` This method creates and returns an accept transport object that employs the ordinary transport object for communications.

Test Before Using Either of two conditions indicate that the connection is ready to use:

- The terminal transport presents the `VC.CONNECTED` advisory.
- `TibrvTransport.waitForVcConnection` returns without error.

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 `TibrvTransport.waitForVcConnection` with `TIBRVCOM_NO_WAIT` as the timeout parameter.

For an explanation, see Testing the New Connection on page 123 in *TIBCO Rendezvous Concepts*.

See Also TibrvTransport.createConnectVc() on page 187
TibrvTransport.getVcConnectSubject() on page 189
TibrvTransport.waitForVcConnection on page 190
VC.CONNECTED on page 288 in *TIBCO Rendezvous Concepts*
VC.DISCONNECTED on page 289 in *TIBCO Rendezvous Concepts*

TibrvTransport.createConnectVc()

Method

Declaration

```
set vcTransport = TibrvTransport.createConnectVc(connectSubject)
```

Purpose Create a virtual circuit connect object.

Remarks Note that programs call this method on an ordinary transport. The method creates and returns a new connect transport object that employs the ordinary transport. Programs may use the ordinary transport for other purposes.

It is illegal to call this method on a virtual circuit terminal transport object (that is, you cannot nest a virtual circuit within another virtual circuit).

Parameter	Description
`vcTransport`	`TibrvTransport` This method creates and returns a connect transport object that employs the ordinary transport object for communications.
`connectSubject`	String The connect transport uses this connect subject to establish a virtual circuit with an *accept* transport in another program. The program must receive this connect subject from the accepting program. The call to `TibrvTransport.createAcceptVc()` creates this subject.

Test Before Using Either of two conditions indicate that the connection is ready to use:

- The transport presents the `VC.CONNECTED` advisory.
- `TibrvTransport.waitForVcConnection` returns without error.

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 `TibrvTransport.waitForVcConnection` with `TIBRVCOM_NO_WAIT` as the timeout parameter.

For an explanation, see Testing the New Connection on page 123 in *TIBCO Rendezvous Concepts*.

See Also

TibrvTransport.createAcceptVc() on page 186
TibrvTransport.getVcConnectSubject() on page 189
TibrvTransport.waitForVcConnection on page 190

VC.CONNECTED on page 288 in *TIBCO Rendezvous Concepts*
VC.DISCONNECTED on page 289 in *TIBCO Rendezvous Concepts*

TibrvTransport.getVcConnectSubject()

Method

Declaration `connectSubject = TibrvTransport.getVcConnectSubject()`

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 `TibrvTransport.createConnectVc()`.

It is an error to call this method on any object other than a virtual circuit accept terminal.

Parameter	Description
`connectSubject`	String The method returns the connect subject of a virtual circuit accept transport. After this call returns, the program must send a message to another program, inviting it to establish a virtual circuit. Furtherer, 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 `TibrvTransport.createConnectVc()`.

See Also TibrvTransport.createAcceptVc() on page 186
TibrvTransport.createConnectVc() on page 187

TibrvTransport.waitForVcConnection

Method

Declaration `TibrvTransport.waitForVcConnection timeout`

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.

⚠ With a non-zero argument, this call blocks all other activity in the program, including automatic dispatch of events and GUI operation. Use with caution.

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`	Double This parameter determines the behavior of the call: • For a quick test of current connection status, supply the constant `TIBRVCOM_NO_WAIT`. The call returns immediately, without blocking. • 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. • To wait indefinitely for a usable connection, supply the constant `TIBRVCOM_WAIT_FOREVER`. The call returns when the connection is complete. If the connection was already complete and is now broken, the call returns immediately.

Errors If the connection is complete (ready to use), this call returns without error. Otherwise it produces an error object; the status code in that error object gives more detail about why the connection is not usable.

Status	Description
`TIBRVCOM_TIMEOUT`	The connection is not yet complete, but the non-negative time limit for waiting has expired.
`TIBRVCOM_VC_NOT_CONNECTED`	The connection was formerly complete, but is now irreparably broken.

See Also TibrvTransport.createAcceptVc() on page 186
TibrvTransport.createConnectVc() on page 187
Testing the New Connection on page 123 in *TIBCO Rendezvous Concepts*
VC.CONNECTED on page 288 in *TIBCO Rendezvous Concepts*
VC.DISCONNECTED on page 289 in *TIBCO Rendezvous Concepts*

Chapter 8 Fault Tolerance

Rendezvous fault tolerance software coordinates a group of redundant program processes into a fault-tolerant distributed system. 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.

Topics

Fault Tolerance Road Map

For a complete discussion of concepts and operating principles, see Fault Tolerance Concepts on page 197 in *TIBCO Rendezvous Concepts*.

For suggestions to help you design programs using fault tolerance features, see Fault Tolerance Programming on page 215 in *TIBCO Rendezvous Concepts*.

For step-by-step hints for implementing fault-tolerant systems, see Developing Fault-Tolerant Programs on page 229 in *TIBCO Rendezvous Concepts*.

Fault tolerance software uses advisory messages to inform programs of status changes. For details, see Fault Tolerance (RVFT) Advisory Messages on page 315 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 on page 407 in *TIBCO Rendezvous Administration*, and discuss with your network administrator.

TibrvFtMember

Class

Purpose Represent membership in a fault tolerance group.

Remarks Upon creating the internal structure of this object, the process joins a fault tolerance group.

By destroying or deleting a member object, the process withdraws its membership in the fault tolerance group.

Method	Description	Page
Life Cycle		
`TibrvFtMember.create`	Create a member of a fault tolerance group.	196
`TibrvFtMember.destroy`	Destroy a member of a fault tolerance group.	200
`TibrvFtMember.isValid()`	Test validity of a fault tolerance member object.	206
Callback Method		
`TibrvFtMember_onFtAction`	Process fault tolerance events for a group member.	207
Accessors		
`TibrvFtMember.getClosure()`	Extract the closure of a fault tolerance member.	201
`TibrvFtMember.getGroupName()`	Extract the group name of a fault tolerance member.	202
`TibrvFtMember.getQueue()`	Extract the event queue of a fault tolerance member.	203
`TibrvFtMember.getTransport()`	Extract the transport of a fault tolerance member.	204
`TibrvFtMember.getWeight()`	Extract the weight of a fault tolerance member.	205
`TibrvFtMember.setWeight`	Change the weight of a fault tolerance member within its group.	210

TibrvFtMember.create

Method

Declaration

```
TibrvFtMember.create queue, transport, groupName, weight,
activeGoal,
    heartbeatInterval, preparationInterval, activationInterval,
closure
```

Purpose Create a member of a fault tolerance group.

Remarks Upon creating a member object, the process becomes a member of the group.

A process may hold simultaneous memberships in several distinct fault tolerance groups. For examples, see Multiple Groups on page 219 in *TIBCO Rendezvous Concepts*.

Avoid joining the same group twice. It is illegal for a process to maintain more than one membership in any one fault tolerance group. The method does not guard against this illegal situation, and results are unpredictable.

Joining a fault tolerance group requires three steps:

1. Declare a variable for the member.

   ```
   Dim WithEvents myFt as TibrvFtMember
   ```

   This declaration triggers VB to automatically define a private subroutine callback stub named `myFt_OnFtAction`. The program must complete this method for each member object.

2. Create the VB object, and assign it to that variable.

   ```
   set myFt = New TibrvFtMember
   ```

3. Create the internal implementation object to join the group.

   ```
   myFt.create ...
   ```

Calling this method on a valid object yields the error `TIBRVCOM_ALREADY_INITIALIZED`.

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 on page 237 in *TIBCO Rendezvous Concepts*.

Group Name The group name must be a legal Rendezvous subject name (see Subject Names on page 61 in *TIBCO Rendezvous Concepts*). You may use names with several elements; for examples, see Multiple Groups on page 219 in *TIBCO Rendezvous Concepts*.

(Sheet 1 of 2)

Parameter	Description
`queue`	`TibrvQueue` Place fault tolerance events for this member on this event queue.
`transport`	`TibrvTransport` Use this transport for fault tolerance internal protocol messages (such as heartbeat messages).
`groupName`	String Join the fault tolerant group with this name. The group name must conform to the syntax required for Rendezvous subject names. For details, see Subject Names on page 61 in *TIBCO Rendezvous Concepts*.
`weight`	Long 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 Rank and Weight on page 206 in *TIBCO Rendezvous Concepts*.
`activeGoal`	Long Rendezvous fault tolerance software sends callback instructions to maintain this number of active members. Acceptable values range from 1 to 65535.

(Sheet 2 of 2)

Parameter	Description
`heartbeatInterval`	Double When this member is active, it sends heartbeat messages at this interval (in seconds). The interval must be positive. To determine the correct value, see Step 4: Choose the Intervals on page 237 in *TIBCO Rendezvous Concepts*.
`preparationInterval`	Double 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 (`TIBRVCOM_FT_PREPARE_TO_ACTIVATE`) 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. 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 `TIBRVCOM_FT_PREPARE_TO_ACTIVATE` hint when this value is zero. To determine the correct value, see Step 4: Choose the Intervals on page 237 in *TIBCO Rendezvous Concepts*.
`activationInterval`	Double 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 (`TIBRVCOM_FT_ACTIVATE`) to the ranking inactive member. 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 (`TIBRVCOM_FT_ACTIVATE`) to the new member. Then interval must be positive. To determine the correct value, see Step 4: Choose the Intervals on page 237 in *TIBCO Rendezvous Concepts*.
`closure`	`Object` Store this closure data in the member object.

See Also TibrvFtMember on page 195.

TibrvFtMember_onFtAction on page 207.
TibrvFtMember.destroy on page 200.
Step 1: Choose a Group Name, page 230 in *TIBCO Rendezvous Concepts*
Step 2: Choose the Active Goal, page 232 in *TIBCO Rendezvous Concepts*
Step 4: Choose the Intervals, page 237 in *TIBCO Rendezvous Concepts*
Step 5: Program Start Sequence, page 241 in *TIBCO Rendezvous Concepts*

TibrvFtMember.destroy

Method

Declaration `TibrvFtMember.`**`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.

This method destroys only the internal structure of the member object. Rendezvous software automatically destroys a member (using this method) when program execution leaves the scope of the object, or sets the object to `Nothing`.

See Also TibrvFtMember.create on page 196
TibrvFtMember.isValid() on page 206

TibrvFtMember.getClosure()

Method

Declaration `closure = TibrvFtMember.getClosure()`

Purpose Extract the closure of a fault tolerance member.

Parameter	Description
`closure`	`Object` Return the closure of the fault tolerance object.

See Also TibrvFtMember.create on page 196

TibrvFtMember.getGroupName()

Method

Declaration

```
groupName = TibrvFtMember.getGroupName()
```

Purpose Extract the group name of a fault tolerance member.

Parameter	Description
`groupName`	String Return the name of the fault tolerance group.

See Also TibrvFtMember.create on page 196

TibrvFtMember.getQueue()

Method

Declaration

```
set queue = TibrvFtMember.getQueue()
```

Purpose Extract the event queue of a fault tolerance member.

Parameter	Description
`queue`	`TibrvQueue` Return the event queue of the fault tolerance object.

See Also TibrvQueue on page 133
TibrvFtMember.create on page 196

TibrvFtMember.getTransport()

Method

Declaration

```
set transport = TibrvFtMember.getTransport()
```

Purpose Extract the transport of a fault tolerance member.

Parameter	Description
`transport`	`TibrvTransport` Return the transport of the fault tolerance object.

See Also TibrvTransport on page 164
TibrvFtMember.create on page 196

TibrvFtMember.getWeight()

Method

Declaration

```
weight = TibrvFtMember.getWeight()
```

Purpose Extract the weight of a fault tolerance member.

Parameter	Description
`weight`	`Long` Return the weight of the fault tolerance member.

See Also TibrvFtMember.create on page 196

TibrvFtMember.isValid()

Method

Declaration `isValid = TibrvFtMember.`**`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.

Parameter	Description
`isValid`	`Boolean` Return the validity of the fault tolerance object.

See Also TibrvFtMember.destroy on page 200

TibrvFtMember_onFtAction

Method

Declaration

```
Private Sub myFtMember_onFtAction(
    ByVal ftMember As TIBRVCOMLib.ITibrvFtMember,
    ByVal groupName As String,
    ByVal action As Long)
```

Purpose

Process fault tolerance events for a group member.

Remarks

Each member program of a fault tolerance group must implement this method. Declaring a variable of class `TibrvFtMember` with events triggers VB to automatically define a COM event callback, and a private subroutine callback stub named *var*`_OnFtAction`. The program must complete this method for each member object.

This method receives its parameters by value, so it cannot delete them.

To withdraw from a fault tolerance group, destroy or delete the *original* member object associated with this callback method.

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 `TIBRVCOM_FT_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 `TIBRVCOM_FT_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 `TIBRVCOM_FT_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 `TIBRVCOM_FT_ACTIVATE` token, it will be ready to activate without delay.

For additional information see Fault Tolerance Callback Actions on page 216 in *TIBCO Rendezvous Concepts*.

Parameter	Description
`ftMember`	`TibrvFtMember` This parameter receives the member object.
`groupName`	`String` This parameter receives a string denoting the name of the fault tolerance group.
`action`	`Long` This parameter receives a token that instructs the callback method to activate, deactivate or prepare to activate. See Action Tokens on page 208.

Action Tokens

The type library defines constants to represent the fault tolerance actions. Each token designates 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 on page 216 in *TIBCO Rendezvous Concepts*).

(Sheet 1 of 2)

Field	Description
`TIBRVCOM_FT_PREPARE_TO_ACTIVATE`	Prepare to activate (hint). 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. 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.
`TIBRVCOM_FT_ACTIVATE`	Activate immediately. Rendezvous fault tolerance software passes this token to the callback method to instruct the program to activate.

(Sheet 2 of 2)

Field	Description
TIBRVCOM_FT_DEACTIVATE	Deactivate immediately. Rendezvous fault tolerance software passes this token to the callback method to instruct the program to deactivate.

See Also TibrvFtMember.create on page 196.

TibrvFtMember.setWeight

Method

Declaration `TibrvFtMember.setWeight weight`

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`	`Long` The new weight value. See weight on page 197.

See Also Adjusting Member Weights on page 227 in *TIBCO Rendezvous Concepts*.

TibrvFtMonitor

Class

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.

Destroying the queue or transport of a monitor automatically destroys the monitor as well.

(Sheet 1 of 2)

Method	Description	Page
Life Cycle		
`TibrvFtMonitor.create`	Monitor a fault tolerance group.	213
`TibrvFtMonitor.destroy`	Stop monitoring a fault tolerance group, and free associated resources.	215
`TibrvFtMonitor.isValid()`	Test validity of a fault tolerance monitor object.	220
Callback Method		
`TibrvFtMonitor_onFtMonitor`	Process fault tolerance events for a monitor.	221
Accessors		
`TibrvFtMonitor.getClosure()`	Extract the closure of a fault tolerance monitor.	216
`TibrvFtMonitor.getGroupName()`	Extract the group name of a fault tolerance monitor.	217
`TibrvFtMonitor.getQueue()`	Extract the event queue of a fault tolerance monitor.	218

(Sheet 2 of 2)

Method	Description	Page
`TibrvFtMonitor.getTransport()`	Extract the transport of a fault tolerance monitor.	219

TibrvFtMonitor.create

Method

Declaration

```
TibrvFtMonitor.create queue, transport, groupName, lostInterval,
closure
```

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 method call.

Monitoring a fault tolerance group requires three steps:

1. Declare a variable for the monitor.

   ```
   Dim WithEvents myMon as TibrvFtMonitor
   ```

   This declaration triggers VB to automatically define a private subroutine callback stub named `myMon_OnFtMonitor`. The program must complete this method for each monitor object.

2. Create the VB object, and assign it to that variable.

   ```
   set myMon = New TibrvFtMonitor
   ```

3. Create the internal implementation object to begin monitoring.

   ```
   myMon.create ...
   ```

Calling this method on a valid object yields the error `TIBRVCOM_ALREADY_INITIALIZED`.

(Sheet 1 of 2)

Parameter	Description
`queue`	`TibrvQueue` Place events for this monitor on this event queue.
`transport`	`TibrvTransport` Listen on this transport for fault tolerance internal protocol messages (such as heartbeat messages).

(Sheet 2 of 2)

Parameter	Description
`groupName`	`String` Monitor the fault tolerant group with this name. The group name must conform to the syntax required for Rendezvous subject names. For details, see Subject Names on page 61 in *TIBCO Rendezvous Concepts*. See also, Group Name on page 214.
`lostInterval`	`Double` 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. The interval must be positive. To determine the correct value, see Step 4: Choose the Intervals on page 237 in *TIBCO Rendezvous Concepts*. See also, Lost Interval on page 214.
`closure`	`Object` 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 on page 61 in *TIBCO Rendezvous Concepts*). You may use names with several elements; for examples, see Multiple Groups on page 219 in *TIBCO Rendezvous Concepts*.

See Also

TibrvFtMonitor_onFtMonitor on page 221.
TibrvFtMonitor.destroy on page 215.

TibrvFtMonitor.destroy

Method

Declaration `TibrvFtMonitor.`**`destroy`**

Purpose Stop monitoring a fault tolerance group, and free associated resources.

Remarks This method destroys only the internal structure of the monitor object. Rendezvous software automatically destroys a monitor (using this method) when program execution leaves the scope of the object, or sets the object to `Nothing`.

See Also TibrvFtMonitor.create on page 213

TibrvFtMonitor.getClosure()

Method

Declaration `closure = TibrvFtMonitor.getClosure()`

Purpose Extract the closure of a fault tolerance monitor.

Parameter	Description
`closure`	`Object` Return the closure of the fault tolerance monitor object.

See Also TibrvFtMonitor.create on page 213

TibrvFtMonitor.getGroupName()

Method

Declaration `groupName = TibrvFtMonitor.`**`getGroupName`**`()`

Purpose Extract the group name of a fault tolerance monitor.

Parameter	Description
`groupName`	String Return the name of the fault tolerance group.

See Also TibrvFtMonitor.create on page 213

TibrvFtMonitor.getQueue()

Method

Declaration

```
set queue = TibrvFtMonitor.getQueue()
```

Purpose Extract the event queue of a fault tolerance monitor.

Parameter	Description
`queue`	`TibrvQueue` Return the queue of the fault tolerance monitor object.

See Also TibrvQueue on page 133
TibrvFtMonitor.create on page 213

TibrvFtMonitor.getTransport()

Method

Declaration

```
set transport = TibrvFtMonitor.getTransport()
```

Purpose Extract the transport of a fault tolerance monitor.

Parameter	Description
`transport`	`TibrvTransport` Return the transport of the fault tolerance monitor object.

See Also TibrvTransport on page 164
TibrvFtMonitor.create on page 213

TibrvFtMonitor.isValid()

Method

Declaration `isValid = TibrvFtMonitor.isValid()`

Purpose Test validity of a fault tolerance monitor object.

Remarks Returns `True` if the monitor is valid; `False` if the monitor has been destroyed or is otherwise invalid.

Parameter	Description
`isValid`	`Boolean` Return the validity of the fault tolerance monitor object.

See Also TibrvFtMonitor.destroy on page 215

TibrvFtMonitor_onFtMonitor

Declaration

```
Private Sub myFtMon_onFtMonitor(
    ByVal ftMonitor As TIBRVCOMLib.ITibrvFtMonitor,
    ByVal groupName As String,
    ByVal numActiveMembers As Long)
```

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. Declaring a variable of class `TibrvFtMonitor` with events triggers VB to automatically define a COM event callback, and a private subroutine callback stub named *var*_`OnFtMonitor`. The program must complete this method for each member object.

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.

This method receives its parameters by value, so it cannot delete them.

To stop monitoring, destroy the *original* monitor object associated with this callback method.

Parameter	Description
`ftMonitor`	`TibrvFtMonitor` This parameter receives the monitor object.
`groupName`	`String` This parameter receives a string denoting the name of the fault tolerance group.
`numActiveMembers`	`Long` This parameter receives the number of group members now active.

See Also TibrvFtMonitor.create on page 213.

Chapter 9 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 on page 139 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 on page 291 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 on page 403 in *TIBCO Rendezvous Administration*.

Some programs require certified delivery to *one of n* worker processes. See Distributed Queue on page 183 in *TIBCO Rendezvous Concepts*.

Topics

TibrvCmListener

Class

Purpose A certified delivery listener object listens for labeled messages and certified messages.

Remarks Each call to the method `TibrvCmListener.create` 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 or deleting 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.

(Sheet 1 of 2)

Method	Description	Page
Life Cycle		
`TibrvCmListener.create`	Listen for messages that match the subject, and request certified delivery when available.	227
`TibrvCmListener.destroy`	Destroy a certified delivery listener.	229
`TibrvCmListener.isValid()`	Test validity of a CM listener object.	234
Callback Method		
`TibrvCmListener_onCmMsg`	Process inbound messages (CM listener events).	235
Confirm		
`TibrvCmListener.confirmMsg`	Explicitly confirm delivery of a certified message.	226
`TibrvCmListener.setExplicitConfirm`	Override automatic confirmation of delivery for this listener.	237
Accessors		
`TibrvCmListener.getClosure()`	Extract the closure of a CM listener.	230

(Sheet 2 of 2)

Method	Description	Page
`TibrvCmListener.getCmTransport()`	Extract the CM transport of a CM listener.	231
`TibrvCmListener.getQueue()`	Extract the event queue of a CM listener.	232
`TibrvCmListener.getSubject()`	Extract the subject of a CM listener.	233

TibrvCmListener.confirmMsg

Method

Declaration `TibrvCmListener.`**`confirmMsg`**` message`

Purpose Explicitly confirm delivery of a certified message.

Remarks Use this method only in programs that override automatic confirmation (see TibrvCmListener.setExplicitConfirm on page 237). The default behavior of certified listeners is to automatically confirm delivery when the callback method returns.

Parameter	Description
`message`	`TibrvMsg` 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` reports the error `TIBRVCOM_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 on page 154 in *TIBCO Rendezvous Concepts*.)

  A labeled but uncertified message can also result when the sender explicitly disallows or removes the listener.

See Also TibrvCmListener on page 224
TibrvCmListener.setExplicitConfirm on page 237

TibrvCmListener.create

Method

Declaration `TibrvCmListener.**create** queue, cmTransport, subject, closure`

Purpose Listen for messages that match the subject, and request certified delivery when available.

Starting a listener requires three steps:

1. Declare a variable for the listener.

   ```
   Dim WithEvents myCmListener as TibrvCmListener
   ```

   This declaration triggers VB to automatically define a private subroutine callback stub named `myCmListener_OnCmMsg`. The program must complete this method for each listener object.

2. Create the VB object, and assign it to that variable.

   ```
   set myCmListener = New TibrvCmListener
   ```

3. Create the internal implementation object to begin listening.

   ```
   myCmListener.create ...
   ```

Calling this method on a valid object yields the error `TIBRVCOM_ALREADY_INITIALIZED`.

Parameter	Description
`queue`	`TibrvQueue` For each inbound message, place the listener event on this event queue.
`cmTransport`	`TibrvCmTransport` Listen for inbound messages on this certified delivery transport.
`subject`	`String` Listen for inbound messages with subjects that match this specification. Wildcard subjects are permitted. The empty string is *not* a legal subject name.
`closure`	`Object` 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 on page 110.

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.create` 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 on page 224
TibrvCmListener_onCmMsg on page 235
TibrvCmTransport.destroy on page 249
TibrvListener.getSubject() on page 117
TibrvTransport.createInbox() on page 169

TibrvCmListener.destroy

Method

Declaration `TibrvCmListener.`**`destroy`**` cancelAgreements`

Purpose Destroy a certified delivery listener.

Remarks This method destroys only the internal structure of the listener object. Rendezvous software automatically destroys a listener (using this method) when program execution leaves the scope of the object, or sets the object to `Nothing`.

Destroying an event object cancels interest in it. Upon return from `TibrvCmListener.destroy`, the destroyed event is no longer dispatched. However, an active callback method of this event continues to run and return normally, even though the event is invalid.

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

Parameter	Description
`cancelAgreements`	`Boolean` `True` cancels all certified delivery agreements of this listener; certified senders delete from their ledgers all messages sent to this listener. `False` leaves all certified delivery agreements in effect, so certified senders continue to store messages.

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.

Deleting an certified delivery listener object (without first destroying it) cancels its agreements.

See Also TibrvCmListener on page 224

TibrvCmListener.getClosure()

Method

Declaration `closure = TibrvCmListener.`**`getClosure`**`()`

Purpose Extract the closure of a CM listener.

Parameter	Description
`closure`	`Object` Return the closure of the CM listener object.

See Also TibrvCmListener.create on page 227

TibrvCmListener.getCmTransport()

Method

Declaration `set cmTransport = TibrvCmListener.getCmTransport()`

Purpose Extract the CM transport of a CM listener.

Parameter	Description
`cmTransport`	`TibrvCmTransport` Return the CM transport of the CM listener object.

See Also TibrvCmTransport on page 238
TibrvCmListener.create on page 227
TibrvCmTransport.getTransport() on page 259

TibrvCmListener.getQueue()

Method

Declaration `set queue = TibrvCmListener.getQueue()`

Purpose Extract the event queue of a CM listener.

Parameter	Description
`queue`	`TibrvQueue` Return the event queue of the CM listener object.

See Also TibrvQueue on page 133
TibrvCmListener.create on page 227

TibrvCmListener.getSubject()

Method

Declaration

```
subject = TibrvCmListener.getSubject()
```

Purpose

Extract the subject of a CM listener.

Parameter	Description
`subject`	`String` Return the subject of the CM listener object.

See Also

TibrvCmListener.create on page 227

TibrvCmListener.isValid()

Method

Declaration `isValid = TibrvCmListener.`**`isValid`**`()`

Purpose Test validity of a CM listener object.

Remarks Returns `True` if the member is valid; `False` if the member has been destroyed or is otherwise invalid.

Parameter	Description
`isValid`	`Boolean` Return the validity of the CM listener object.

See Also TibrvCmListener.destroy on page 229

TibrvCmListener_onCmMsg

Method

Declaration

```
Private Sub myCmListener_onCmMsg(
    ByVal cmListener As TIBRVCOMLib.ITibrvCmListener,
    ByVal message As TIBRVCOMLib.ITibrvMsg)
```

Purpose

Process inbound messages (CM listener events).

Remarks

Implement this method to process inbound messages.

Declaring a variable of class `TibrvCmListener` with events triggers VB to automatically define a COM event callback, and a private subroutine callback stub named *var*`_OnCmMsg`. The program must complete this method for each CM listener object.

This method receives its parameters by value, so it cannot delete them.

To stop listening to the message subject, destroy the *original* listener object associated with this callback method.

Parameter	Description
`cmListener`	`TibrvCmListener` This parameter receives the CM listener object.
`message`	`TibrvMsg` This parameter receives the inbound message object. To preserve this message beyond exit from this callback method, the program must detach it.

CM Label Information

The callback method for certified delivery messages can use certified delivery label properties to discriminate these situations:

- If extracting the `cmSender` property yields the error `TIBRVCOM_NOT_FOUND`, then the message uses the reliable protocol (that is, it was sent from an ordinary transport).
- If the `cmSender` property is a valid sender name, then the message uses the certified delivery protocol (that is, it is a labeled message, sent from a certified delivery transport).

Once the callback method determines that the message uses the certified delivery protocol, it can discriminate further:

— If extracting the `cmSequenceNumber` property yields the error `TIBRVCOM_NOT_FOUND`, then the listener is *not registered* for certified delivery from the sender.

— If the `cmSequenceNumber` property is a positive sequence number, then a certified delivery agreement is in effect for this subject with the sender.

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 on page 154 in *TIBCO Rendezvous Concepts*.)

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

TibrvMsg.detach on page 54
TibrvCmListener.create on page 227
TibrvMsg.getProperty() on page 72

TibrvCmListener.setExplicitConfirm

Method

Declaration `TibrvCmListener.`**`setExplicitConfirm`**

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 TibrvCmListener_onCmMsg on page 235). 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 processing inbound messages involves activity that is asynchronous with respect to the message callback method; for example, additional network communications.

No method exists to restore the default behavior—that is, to reverse the effect of this method.

See Also TibrvCmListener on page 224
TibrvListener_onMsg on page 120
TibrvCmListener.confirmMsg on page 226

TibrvCmTransport

Class

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.

Programs must explicitly destroy each certified delivery transport object. Destroying a certified delivery transport object invalidates subsequent certified send calls on that object, and invalidates any certified listeners using that transport (while preserving the certified delivery agreements of those listeners).

(Sheet 1 of 3)

Method	Description	Page
Life Cycle		
`TibrvCmTransport.create`	Create a transport for certified delivery.	245
`TibrvCmTransport.destroy`	Destroy a certified delivery transport.	249
`TibrvCmTransport.isValid()`	Test validity of a CM transport object.	260
Send Messages		
`TibrvCmTransport.send`	Send a labeled message.	267
`TibrvCmTransport.sendReply`	Send a labeled reply message.	268
Certified Delivery Control		
`TibrvCmTransport.addListener`	Pre-register an anticipated listener.	241

(Sheet 2 of 3)

Method	Description	Page
`TibrvCmTransport.allowListener`	Invite the named receiver to reinstate certified delivery for its listeners, superseding the effect of any previous disallow calls.	242
`TibrvCmTransport.connectToRelayAgent`	Connect a certified delivery transport to its designated relay agent.	243
`TibrvCmTransport.disallowListener`	Cancel certified delivery to all listeners at a specific correspondent. Deny subsequent certified delivery registration requests from those listeners.	250
`TibrvCmTransport.disconnectFromRelayAgent`	Disconnect a certified delivery transport from its relay agent.	251
`TibrvCmTransport.removeListener`	Unregister a specific listener at a specific correspondent, and free associated storage in the sender's ledger.	263
Ledger		
`TibrvCmTransport.expireMessages`	Mark specified outbound CM messages as expired.	252
`TibrvCmTransport.removeSendState`	Reclaim ledger space from obsolete subjects.	265
`TibrvCmTransport.reviewLedger`	Query the ledger for stored items related to a subject name.	266
`TibrvCmTransport_onLedgerMsg`	Programs define this method to process ledger review messages.	261
`TibrvCmTransport.stopReview`	Stop a ledger review.	273
`TibrvCmTransport.syncLedger`	Synchronize the ledger to its storage medium.	274

(Sheet 3 of 3)

Method	Description	Page
Accessors		
`TibrvCmTransport.getDefaultTimeLimit()`	Get the default message time limit for all outbound certified messages from a transport.	253
`TibrvCmTransport.getLedgerName()`	Extract the ledger name of a certified delivery transport.	254
`TibrvCmTransport.getName()`	Extract the correspondent name of a certified delivery transport or distributed queue member.	255
`TibrvCmTransport.getRelayAgent()`	Extract the name of the relay agent used by a certified delivery transport.	256
`TibrvCmTransport.getRequestOld()`	Extract the request old messages flag of a certified delivery transport.	257
`TibrvCmTransport.getSyncLedger()`	Extract the sync ledger flag of a certified delivery transport.	258
`TibrvCmTransport.getTransport()`	Extract the transport employed by a certified delivery transport or a distributed queue member.	259
`TibrvCmTransport.setDefaultTimeLimit`	Set the default message time limit for all outbound certified messages from a transport.	271
`TibrvCmTransport.setTaskBacklogLimit...`	Set the scheduler task queue limits of a distributed queue transport.	272

TibrvCmTransport.addListener

Method

Declaration `TibrvCmTransport.`**`addListener`**` cmName, subject`

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` reports the error `TIBRVCOM_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
`cmName`	`String` Anticipate a listener from a correspondent with this reusable name.
`subject`	`String` Anticipate a listener for this subject. Wildcard subjects are illegal.

See Also Name, page 247
TibrvCmTransport.allowListener on page 242
TibrvCmTransport.disallowListener on page 250
TibrvCmTransport.removeListener on page 263
Anticipating a Listener, page 161 in *TIBCO Rendezvous Concepts*

TibrvCmTransport.allowListener

Method

Declaration `TibrvCmTransport.`**`allowListener`**` cmName`

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`	`String` Accept requests for certified delivery to listeners at the transport with this correspondent name.

See Also Name, page 247
TibrvCmTransport.disallowListener on page 250
Disallowing Certified Delivery, page 164 in *TIBCO Rendezvous Concepts*

TibrvCmTransport.connectToRelayAgent

Method

Declaration `TibrvCmTransport.`**`connectToRelayAgent`**

Purpose Connect a certified delivery transport to its designated relay agent.

Remarks Programs may specify a relay agent when creating a CM transport object.

Connect calls are non-blocking; they immediately return control to the program, and asynchronously attempt to connect to the relay agent (continuing until they succeed, or until the program makes a disconnect call).

When a transport attempts to connect to a relay agent, Rendezvous software automatically locates the relay agent process (if it exists). When the program successfully connects to the relay agent, they synchronize:

- The transport receives a `RELAY.CONNECTED` advisory, informing it of successful contact with the relay agent. (Listen for all advisory messages on the ordinary `TibrvTransport` that the `TibrvCmTransport` employs.)

  (When a program cannot locate its relay agent, certified delivery software produces `DELIVERY.NO_RESPONSE` advisories; however, we recommend against designing programs to rely on this side effect.)
- If the client transport is a CM *listener*, the relay agent listens to the same set of subjects on behalf of the client. The relay agent also updates its confirmation state to reflect the state of the transport.
- If the client transport is a CM *sender*, the relay agent updates its acceptance state to reflect the state of the transport. The sending client updates its confirmation state to reflect the state of the relay agent.
- The transport and relay agent exchange the CM data messages that they have been storing during the time they were disconnected.

We recommend that programs remain connected for a minimum of two minutes, to allow time for this synchronization to complete. (Two minutes is a generous estimate, which is sufficient for most situations. Actual time synchronization time can be much shorter, and varies with the number of stored messages and the degree to which protocol state has changed.)

If the transport is already connected to its relay agent, then this method returns normally, and does not trigger a `RELAY.CONNECTED` advisory.

`TibrvCmTransport.create` automatically connects a transport to its designated relay agent upon creation.

Errors The error code `TIBRVCOM_UNABLE` indicates that the transport does not have a relay agent.

See Also TibrvCmTransport.create on page 245
TibrvCmTransport.disconnectFromRelayAgent on page 251
Relay Agent, page 170 in *TIBCO Rendezvous Concepts*

TibrvCmTransport.create

Method

Declaration

```
TibrvCmTransport.create transport, cmName, requestOld,
    ledgerName, syncLedger, relayAgent
```

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.

Calling this method on a valid object yields the error `TIBRVCOM_ALREADY_INITIALIZED`.

(Sheet 1 of 3)

Parameter	Description
`transport`	`TibrvTransport` The new `TibrvCmTransport` employs this transport object for network communications. Destroying the `TibrvCmTransport` does not affect this `TibrvTransport` object. It is illegal to supply a virtual circuit transport object for this parameter.
`cmName`	`String`; optional Bind this reusable name to the new `TibrvCmTransport`, so the `TibrvCmTransport` represents a persistent correspondent with this name. 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 `TibrvCmTransport.create`. If absent, the empty string ("") or `vbNullString`, then `TibrvCmTransport.create` generates a unique, non-reusable name for the duration of the transport. For more information, see Name on page 247.

(Sheet 2 of 3)

Parameter	Description
`requestOld`	`Boolean`; optional 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. If this parameter is `true` *and* `cmName` is present, then the new `TibrvCmTransport` requires certified senders to retain unacknowledged messages sent to this persistent correspondent. When the new `TibrvCmTransport` begins listening to the appropriate subjects, the senders can complete delivery. (It is an error to supply `true` when `cmName` is omitted.) If this parameter is `false` (or absent), then the new `TibrvCmTransport` does not require certified senders to retain unacknowledged messages. Certified senders may delete those messages from their ledgers.
`ledgerName`	`String`; optional If this argument is non-null, then the new `TibrvCmTransport` 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 name. If absent, the empty string ("") or `vbNullString`, then the new `TibrvCmTransport` uses a process-based ledger. For more information, see Ledger File on page 247.
`syncLedger`	`Boolean`; optional If this argument is `true`, then operations that update the ledger file do not return until the changes are written to the storage medium. If this argument is `false` (or absent), the operating system writes changes to the storage medium asynchronously.

(Sheet 3 of 3)

Parameter	Description
`relayAgent`	`String`; optional Designate the `rvrad` process with this name as the new transport's relay agent. If absent, the empty string ("") or `vbNullString`, the new `TibrvCmTransport` does not use a relay agent. If non-null, the relay agent name must conform to the syntax rules for reusable names. For details, see Reusable Names on page 167 in *TIBCO Rendezvous Concepts*. It is illegal for a relay agent to have the same name as a CM correspondent. For more information, see Relay Agent on page 247.

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.create` 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 on page 150 in *TIBCO Rendezvous Concepts*, and Persistent Correspondents on page 159 in *TIBCO Rendezvous Concepts*. For details of reusable name syntax, see Reusable Names on page 167 in *TIBCO Rendezvous Concepts*.

Relay Agent

`TibrvCmTransport.create` automatically connects a transport to its designated relay agent upon creation; see TibrvCmTransport.connectToRelayAgent on page 243.

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 on page 274.

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 on page 249
TibrvCmTransport.connectToRelayAgent on page 243

TibrvCmTransport.destroy

Method

Declaration `TibrvCmTransport.`**`destroy`**

Purpose Destroy a certified delivery transport.

Remarks This method destroys only the internal structure of the transport object. Rendezvous software automatically destroys a transport (using this method) when program execution leaves the scope of the object, or sets the object to `Nothing`.

Destroying a certified delivery transport with a file-based ledger always leaves the ledger file intact; it neither erases nor removes a ledger file.

This method automatically disconnects the transport from its relay agent before destroying the object; see `TibrvCmTransport.disconnectFromRelayAgent`.

Distributed Queue When calling this method to destroy a distributed queue transport, the distributed queue needs the listeners, queues and dispatchers (associated with the transport) to remain operational—otherwise the distributed queue can lose reliable (non-certified) task messages before they are processed. Programs must wait until after the transport has been completely destroyed before destroying these associated objects.

See Also TibrvCmTransport.create on page 245
TibrvCmTransport.disconnectFromRelayAgent on page 251

TibrvCmTransport.disallowListener

Method

Declaration `TibrvCmTransport.`**`disallowListener`**` cmName`

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 on page 242.

Parameter	Description
`cmName`	`String` Cancel certified delivery to listeners of the transport with this name.

See Also Name, page 247
TibrvCmTransport.allowListener on page 242
Disallowing Certified Delivery, page 164 in *TIBCO Rendezvous Concepts*

TibrvCmTransport.disconnectFromRelayAgent

Method

Declaration `TibrvCmTransport.`**`disconnectFromRelayAgent`**

Purpose Disconnect a certified delivery transport from its relay agent.

Remarks Disconnect calls are non-blocking; they immediately return control to the program, and asynchronously proceed with these clean-up tasks:

- If the client transport is a CM *listener*, the relay agent attempts to synchronize its listening state with the transport (to assure that the relay agent adequately represents the listening interest of the client).
- The transport stops communicating with the relay agent.
- The transport stores subsequent outbound events—including data messages and protocol state changes. If the transport is a certified *sender*, it cancels its request for delivery confirmation of outstanding unconfirmed messages. (See also, Requesting Confirmation on page 157 in *TIBCO Rendezvous Concepts*.)
- The relay agent stores subsequent inbound events for the transport—including data messages and protocol state changes.
- A transport that explicitly disconnects without terminating receives a `RELAY.DISCONNECTED` advisory, informing it that is safe to sever the physical network connection. (Terminating transports never receive this advisory; instead, it is safe to sever the connection when the destroy call returns.)

`TibrvCmTransport.destroy` automatically disconnects a CM transport from its relay agent before termination.

Errors The error code `TIBRVCOM_UNABLE` indicates that the transport does not have a relay agent.

See Also TibrvCmTransport.connectToRelayAgent on page 243
TibrvCmTransport.destroy on page 249
Relay Agent, page 170 in *TIBCO Rendezvous Concepts*

TibrvCmTransport.expireMessages

Method

Declaration `TibrvCmTransport.`**`expireMessages`**` subject, sequenceNumber`

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.)

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`	`String` Mark messages with this subject. Wildcards 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`	`TibrvInt64` Mark messages with sequence numbers *less than or equal* to this value.

See Also DELIVERY.FAILED on page 298 in *TIBCO Rendezvous Concepts*

TibrvCmTransport.getDefaultTimeLimit()

Method

Declaration

```
timeLimit = TibrvCmTransport.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 TibrvMsg.setProperty on page 85). 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.

Parameter	Description
`timeLimit`	Double Return the default message time limit of the CM transport.

See Also TibrvCmTransport.setDefaultTimeLimit on page 271
TibrvMsg.setProperty on page 85

TibrvCmTransport.getLedgerName()

Method

Declaration `ledgerName = TibrvCmTransport.getLedgerName()`

Purpose Extract the ledger name of a certified delivery transport.

Parameter	Description
`ledgerName`	String Return the ledger file name of the CM transport.

Errors The error code `TIBRVCOM_CANNOT_RETRIEVE` indicates that the transport does not have a ledger file.

See Also Ledger File, page 247
TibrvCmTransport.create on page 245

TibrvCmTransport.getName()

Method

Declaration `cmName = TibrvCmTransport.getName()`

Purpose Extract the correspondent name of a certified delivery transport or distributed queue member.

Parameter	Description
`cmName`	String Return the correspondent name of the CM transport.

See Also Name, page 247
TibrvCmTransport.create on page 245
TibrvCmTransport.createDistributedQueue on page 279

TibrvCmTransport.getRelayAgent()

Method

Declaration `relayAgent = TibrvCmTransport.getRelayAgent()`

Purpose Extract the name of the relay agent used by a certified delivery transport.

Parameter	Description
`relayAgent`	String Return the relay agent name of the CM transport.

Errors The error code `TIBRVCOM_CANNOT_RETRIEVE` indicates that the transport does not have a relay agent.

See Also Relay Agent, page 247
TibrvCmTransport.create on page 245

TibrvCmTransport.getRequestOld()

Method

Declaration `requestOld = TibrvCmTransport.`**`getRequestOld`**`()`

Purpose Extract the request old messages flag of a certified delivery transport.

Parameter	Description
`requestOld`	Boolean Return the request old messages flag of the CM transport.

See Also requestOld on page 246
TibrvCmTransport.create on page 245

TibrvCmTransport.getSyncLedger()

Method

Declaration

```
syncLedger = TibrvCmTransport.getSyncLedger()
```

Purpose Extract the sync ledger flag of a certified delivery transport.

Parameter	Description
syncLedger	Boolean Return the sync ledger flag of the CM transport.

Errors The error code `TIBRVCOM_CANNOT_RETRIEVE` indicates that the transport does not have a ledger file.

See Also Ledger File, page 247
TibrvCmTransport.create on page 245

TibrvCmTransport.getTransport()

Method

Declaration

```
set transport = TibrvCmTransport.getTransport()
```

Purpose Extract the transport employed by a certified delivery transport or a distributed queue member.

Parameter	Description
`transport`	`TibrvTransport` Return the transport of the CM transport object.

See Also TibrvTransport on page 164
TibrvCmTransport.create on page 245
TibrvCmTransport.createDistributedQueue on page 279

TibrvCmTransport.isValid()

Method

Declaration `isValid = TibrvCmTransport.isValid()`

Purpose Test validity of a CM transport object.

Remarks Returns `True` if the transport is valid; `False` if the transport has been destroyed or is otherwise invalid.

Parameter	Description
`isValid`	`Boolean` Return the validity of the CM transport object.

See Also TibrvCmTransport.create on page 245
TibrvCmTransport.destroy on page 249

TibrvCmTransport_onLedgerMsg

Method

Declaration

```
Private Sub myCmTransport_onLedgerMsg(
    ByVal cmTransport As TIBRVCOMLib.ITibrvCmTransport,
    ByVal subject As String,
    ByVal message As TIBRVCOMLib.ITibrvMsg,
    ByVal closure As Object)
```

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.

When this method returns, `TibrvCmTransport.reviewLedger` prepares the next ledger method and calls the method again.

To stop reviewing the ledger, call `TibrvCmTransport.stopReview` from within this callback method; `TibrvCmTransport.reviewLedger` cancels the review and returns immediately.

This method receives its parameters by value, so it cannot delete them.

Declaring a variable of class `TibrvCmTransport` with events triggers VB to automatically define a COM event callback, and a private subroutine callback stub named *var*`_OnLedgerMsg`. The program must complete this method for each CM transport object (except distributed queue objects).

Parameter	Description
`cmTransport`	`TibrvCmTransport` This parameter receives the CM transport object.
`subject`	`String` This parameter receives the subject for this ledger item.
`message`	`TibrvMsg` This parameter receives a summary message describing the delivery status of messages in the ledger. For information about the fields of the summary message, see the table Field Name on page 262.
`closure`	`Object` This parameter receives closure data that the program supplied to `TibrvCmTransport.reviewLedger`.

Field Name	Description
`subject`	The subject that this message summarizes. This field has datatype `TIBRVCOM_MSG_STRING`.
`seqno_last_sent`	The sequence number of the most recent message sent with this subject name. This field has datatype `TIBRVCOM_MSG_U64`.
`total_msgs`	The total number of messages stored at this subject name. This field has datatype `TIBRVCOM_MSG_U32`.
`total_size`	The total storage (in bytes) occupied by all messages with this subject name. If the ledger contains several messages with this subject name, then this field sums the storage space over all of them. This field has datatype `TIBRVCOM_MSG_I64`.
`listener`	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. This field has datatype `TIBRVCOM_MSG_MSG`.
`listener.`**`name`**	Within each `listener` submessage, the `name` field contains the name of the listener transport. This field has datatype `TIBRVCOM_MSG_STRING`.
`listener.`**`last_confirmed`**	Within each `listener` submessage, the `last_confirmed` field contains the sequence number of the last message for which the listener confirmed delivery. This field has datatype `TIBRVCOM_MSG_U64`.

See Also TibrvCmTransport.reviewLedger on page 266
TibrvCmTransport.stopReview on page 273

TibrvCmTransport.removeListener

Method

Declaration `TibrvCmTransport.`**`removeListener`**` cmName, subject`

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` reports the error `TIBRVCOM_INVALID_SUBJECT`.

Parameter	Description
`cmName`	`String` Cancel certified delivery of the subject to listeners of this correspondent.
`subject`	`String` Cancel certified delivery of this subject to the named listener. Wildcard subjects are illegal.

See Also Name, page 247

TibrvCmTransport.addListener on page 241
TibrvCmTransport.disallowListener on page 250
Canceling Certified Delivery, page 162 in *TIBCO Rendezvous Concepts*

TibrvCmTransport.removeSendState

Method

Declaration

```
TibrvCmTransport.removeSendState subject
```

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.

⚠ Do not use this method to clear subjects that are still in use.

Parameter	Description
`subject`	`String` 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 on page 266
TibrvCmTransport.send on page 267
DELIVERY.COMPLETE on page 296 in *TIBCO Rendezvous Concepts*

TibrvCmTransport.reviewLedger

Method

Declaration `TibrvCmTransport.`**`reviewLedger`**` subject, closure`

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 calls `TibrvCmTransport.stopReview`, then `TibrvCmTransport.reviewLedger` returns immediately (without any further callbacks).

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 TibrvCmTransport_onLedgerMsg on page 261.

Parameter	Description
`subject`	`String` Query for items related to this subject name. 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`	`Object` Pass this closure data to the review callback method.

See Also TibrvCmTransport_onLedgerMsg on page 261
TibrvCmTransport.stopReview on page 273

TibrvCmTransport.send

Method

Declaration `TibrvCmTransport.send message`

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 TibrvMsg.setProperty on page 85. 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 on page 271); if that default is not set for the transport, the default time limit is zero (no time limit).

Parameter	Description
`message`	`TibrvMsg` Send this message. Wildcard subjects are illegal.

See Also TibrvCmTransport.sendReply on page 268
TibrvCmTransport.sendRequest() on page 269
TibrvCmTransport.setDefaultTimeLimit on page 271
TibrvMsg.setProperty on page 85

TibrvCmTransport.sendReply

Method

Declaration `TibrvCmTransport.`**`sendReply`**` replyMsg, requestMsg`

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`	`TibrvMsg` Send this *outbound* reply message.
`requestMsg`	`TibrvMsg` Send a reply to this *inbound* request message; extract its reply subject to use as the subject of the outbound reply message. If this message has a wildcard reply subject, the method produces an error.

⚠ 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 on page 267
TibrvCmTransport.sendRequest() on page 269

TibrvCmTransport.sendRequest()

Method

Declaration `set replyMsg = TibrvCmTransport.`**`sendRequest`**`(message, timeout)`

Purpose Send a labeled request message and wait for a reply.

Blocking can Stall Event Dispatch

⚠ This call blocks all other activity in the program.

Parameter	Description
`replyMsg`	`TibrvMsg` Return this reply message.
`message`	`TibrvMsg` Send this request message. Wildcard subjects are illegal.
`timeout`	`Double` 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 on page 87.

The program owns the reply message object, and must delete it to reclaim its storage.

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:

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. (The reply event uses a private queue that is not accessible to the program.) If the time limit expires before a reply arrives, then report the error `TIBRVCOM_TIMEOUT`.
4. Return the reply message as the value of the method call.

See Also TibrvCmTransport.send on page 267
TibrvCmTransport.sendReply on page 268

TibrvCmTransport.setDefaultTimeLimit

Method

Declaration `TibrvCmTransport.`**`setDefaultTimeLimit`**` timeLimit`

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 TibrvMsg.setProperty on page 85). 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`	`Double` Use this time limit (in whole seconds). The time limit must be non-negative. Fractional values are truncated to whole seconds.

See Also TibrvCmTransport.getDefaultTimeLimit() on page 253
TibrvMsg.setProperty on page 85

TibrvCmTransport.setTaskBacklogLimit...

Method

Declaration

```
TibrvCmTransport.setTaskBacklogLimitInBytes limitBySizeInBytes

TibrvCmTransport.setTaskBacklogLimitInMessages limitByMessages
```

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 either these properties, the default is no limit.

When the task messages in the queue exceed either of the two limits, Rendezvous software deletes new inbound task messages.

Programs may call each of these methods at most once. Those calls must occur before the transport assumes the scheduler role; after a transport acts as a scheduler, the values are fixed, and subsequent attempts to change them result in status code `TIBRVCOM_NOT_PERMITTED`.

Parameter	Description
`limitBySizeInBytes`	`Long` Use this size limit (in bytes). Zero is a special value, indicating no size limit.
`limitByMessages`	`Long` Use this message limit (number of messages). Zero is a special value, indicating no limit on the number of messages.

See Also

Distributed Queue, page 183, in *TIBCO Rendezvous Concepts*

TibrvCmTransport.stopReview

Method

Declaration `TibrvCmTransport.`**`stopReview`**

Purpose Stop a ledger review.

Remarks To stop a ledger review, call this method from within `TibrvCmTransport_onLedgerMsg`.

See Also TibrvCmTransport.reviewLedger on page 266
TibrvCmTransport_onLedgerMsg on page 261

TibrvCmTransport.syncLedger

Method

Declaration `TibrvCmTransport.`**`syncLedger`**

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.

Even when the program does not call this method, Rendezvous software periodically synchronizes 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 `TIBRVCOM_UNABLE` indicates that the transport does not have a ledger file.

See Also Ledger File, page 247
TibrvCmTransport.create on page 245
TibrvCmTransport.getSyncLedger() on page 258

Chapter 10 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 distributed queue transport objects, each in a separate process. From the outside, a distributed queue group appears as though it were a single transport object; inside, the group members act in concert to process inbound task messages. Ordinary transports and CM transports 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.

Distributed queues depend upon the certified delivery methods and the fault tolerance methods.

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 on page 411 in *TIBCO Rendezvous Administration*.

See Also Distributed Queue, page 183 in *TIBCO Rendezvous Concepts*

Topics

Distributed Queue Transport

Each distributed queue member uses a distributed queue transport for communications. A distributed queue transport is a special-case instance of class `TibrvCmTransport`.

Each distributed queue transport object employs a `TibrvTransport` for network communications. The distributed queue transport adds the accounting and coordination mechanisms needed for one-of-n delivery.

Disabled Methods

Although each distributed queue transport is an instance of `TibrvCmTransport`, all methods related to *sending* messages are disabled in distributed queue transports. These disabled methods report the error `TIBRVCOM_INVALID_TRANSPORT_OBJECT`. For a list of legal and disabled methods, see Shared Methods on page 278. See also Certified Delivery Behavior in Queue Members on page 185 in *TIBCO Rendezvous Concepts*.

Subscriptions

All members of a distributed queue must listen to exactly the same set of subjects. See Enforcing Identical Subscriptions on page 186 in *TIBCO Rendezvous Concepts*.

Certified Task Messages

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).

(Sheet 1 of 2)

Method	Description	Page
Life Cycle		
`TibrvCmTransport.createDistributedQueue`	Create a transport as a distributed queue member.	279
`TibrvCmTransport.isDistributed()`	Test whether a CM transport object implements a distributed queue member.	285

(Sheet 2 of 2)

Method	Description	Page
Accessors When called on a CM transport object (which is not initialized as a distributed transport), these methods report the error TIBRVCOM_NOT_DISTRIBUTED_TRANSPORT.		
TibrvCmTransport.getCompleteTime()	Extract the worker complete time limit of a distributed queue member.	282
TibrvCmTransport.getWorkerTasks()	Extract the worker task capacity of a distributed queue member.	283
TibrvCmTransport.getWorkerWeight()	Extract the worker weight of a distributed queue member.	284
TibrvCmTransport.setCompleteTime	Set the worker complete time limit of a distributed queue member.	286
TibrvCmTransport.setWorkerTasks	Set the worker task capacity of a distributed queue member.	287
TibrvCmTransport.setWorkerWeight	Set the worker weight of a distributed queue member.	288

Constants The type library defines these constants for use with distributed queue methods.

Constant	Value
TIBRVCOM_CM_DEFAULT_COMPLETE_TIME	0
TIBRVCOM_CM_DEFAULT_WORKER_WEIGHT	1
TIBRVCOM_CM_DEFAULT_WORKER_TASKS	1
TIBRVCOM_CM_DEFAULT_SCHEDULER_WEIGHT	1
TIBRVCOM_CM_DEFAULT_SCHEDULER_HB (heartbeat)	1.0

Shared Methods	

Legal Methods	TibrvCmTransport.destroy
	TibrvCmTransport.getName()
	TibrvCmTransport.getTransport()
	TibrvCmTransport.isValid()

Disabled Methods	TibrvCmTransport.addListener TibrvCmTransport.allowListener TibrvCmTransport.connectToRelayAgent TibrvCmTransport.create TibrvCmTransport.disallowListener TibrvCmTransport.disconnectFromRelayAgent TibrvCmTransport.getDefaultTimeLimit() TibrvCmTransport.getLedgerName() TibrvCmTransport.getRelayAgent() TibrvCmTransport.getRequestOld() TibrvCmTransport.getSyncLedger() TibrvCmTransport_onLedgerMsg TibrvCmTransport.removeListener TibrvCmTransport.removeSendState TibrvCmTransport.reviewLedger TibrvCmTransport.send TibrvCmTransport.sendReply TibrvCmTransport.sendRequest() TibrvCmTransport.setDefaultTimeLimit TibrvCmTransport.stopReview TibrvCmTransport.syncLedger TibrvTransport.send TibrvTransport.sendReply TibrvTransport.sendRequest()

When called on a distributed transport object, the disabled methods report the error TIBRVCOM_NOT_FOR_DISTRIBUTED_TRANSPORT.

TibrvCmTransport.createDistributedQueue

Method

Declaration

```
TibrvCmTransport.createDistributedQueue transport, cmName,
    workerWeight, workerTasks,
    schedulerWeight, schedulerHeartbeat, schedulerActivation
```

Purpose Create a transport as a distributed queue member.

Remarks The new distributed queue transport must employ a valid `TibrvTransport` for network communications.

Calling this method on a valid object yields the error `TIBRVCOM_ALREADY_INITIALIZED`.

(Sheet 1 of 3)

Parameter	Description
`transport`	`TibrvTransport` The new distributed queue `TibrvCmTransport` employs this `TibrvTransport` object for network communications. Destroying the distributed queue `TibrvCmTransport` does not affect this transport.
`cmName`	`String` Bind this reusable name to the new transport object, which becomes a member of the distributed queue with this name. 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 `TibrvCmTransport.create`. It cannot be the empty string. For more information, see Reusable Names on page 167 in *TIBCO Rendezvous Concepts*.

(Sheet 2 of 3)

Parameter	Description
`workerWeight`	`Long`, optional When the scheduler receives a task, it assigns the task to the available worker with the greatest worker weight. A worker is considered available unless either of these conditions are true: • The pending tasks assigned to the worker member exceed its task capacity. • The worker is also the scheduler. (The scheduler assigns tasks to its own worker role only when no other workers are available.) When omitted, the default value is `1`.
`workerTasks`	`Long`, optional 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 value must be a non-negative integer. When omitted, the default value is `1`. Zero is a special value, indicating that this distributed queue member is a dedicated scheduler (that is, it never accepts tasks). ⚠ 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 on page 188 in *TIBCO Rendezvous Concepts*.

(Sheet 3 of 3)

Parameter	Description
`schedulerWeight`	`Long`, optional 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. When omitted, the default value is `1`. 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 Rank and Weight on page 206 in *TIBCO Rendezvous Concepts*.
`schedulerHeartbeat`	`Double`, optional The scheduler sends heartbeat messages at this interval (in seconds). All `Distributed Queue Transport` 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 Step 4: Choose the Intervals on page 237 in *TIBCO Rendezvous Concepts*. When omitted, the default value is `1.0`.
`schedulerActivation`	`Double`, optional 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. All `Distributed Queue Transport` 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 Step 4: Choose the Intervals on page 237 in *TIBCO Rendezvous Concepts*. When omitted, the default value is `3.5`.

See Also TibrvCmTransport.destroy on page 249
Distributed Queue, page 183, in *TIBCO Rendezvous Concepts*

TibrvCmTransport.getCompleteTime()

Method

Declaration

```
completeTime = TibrvCmTransport.getCompleteTime()
```

Purpose Extract the worker complete time limit of a distributed queue member.

Parameter	Description
`completeTime`	`Double` Return the complete time (in seconds).

See Also Distributed Queue, page 183, in *TIBCO Rendezvous Concepts*
TibrvCmTransport.setCompleteTime on page 286

TibrvCmTransport.getWorkerTasks()

Method

Declaration `workerTasks = TibrvCmTransport.`**`getWorkerTasks`**`()`

Purpose Extract the worker task capacity of a distributed queue member.

Parameter	Description
`workerTasks`	`Long` Return the worker task capacity.

See Also Distributed Queue, page 183, in *TIBCO Rendezvous Concepts*
TibrvCmTransport.createDistributedQueue on page 279
TibrvCmTransport.setWorkerTasks on page 287

TibrvCmTransport.getWorkerWeight()

Method

Declaration `workerWeight = TibrvCmTransport.`**`getWorkerWeight`**`()`

Purpose Extract the worker weight of a distributed queue member.

Parameter	Description
`workerWeight`	`Long` Return the worker weight.

See Also Distributed Queue, page 183, in *TIBCO Rendezvous Concepts*
TibrvCmTransport.createDistributedQueue on page 279
TibrvCmTransport.setWorkerWeight on page 288

TibrvCmTransport.isDistributed()

Method

Declaration `isDistributed = TibrvCmTransport.`**`isDistributed`**`()`

Purpose Test whether a CM transport object implements a distributed queue member.

Parameter	Description
`isDistributed`	`Boolean` Return `True` if the object implements a distributed queue member. Return `False` if the object implements a CM transport.

See Also Distributed Queue, page 183, in *TIBCO Rendezvous Concepts*
TibrvCmTransport on page 238
TibrvCmTransport.create on page 245
TibrvCmTransport.createDistributedQueue on page 279

TibrvCmTransport.setCompleteTime

Method

Declaration `TibrvCmTransport.`**`setCompleteTime`**` completeTime`

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`	`Double` Use this complete time (in seconds). The time must be non-negative.

See Also Distributed Queue, page 183, in *TIBCO Rendezvous Concepts*
TibrvCmTransport.getCompleteTime() on page 282

TibrvCmTransport.setWorkerTasks

Method

Declaration `TibrvCmTransport.`**`setWorkerTasks`**` workerTasks`

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).

⚠ 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 on page 188 in *TIBCO Rendezvous Concepts*.

Parameter	Description
`workerTasks`	`Long` Use this task capacity. The value must be a non-negative integer.

See Also Distributed Queue, page 183, in *TIBCO Rendezvous Concepts*
TibrvCmTransport.createDistributedQueue on page 279
TibrvCmTransport.getWorkerTasks() on page 283

TibrvCmTransport.setWorkerWeight

Method

Declaration `TibrvCmTransport.`**`setWorkerWeight`**` workerWeight`

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 `TibrvCmTransport.createDistributedQueue`, or change it dynamically using this method.

Parameter	Description
`workerWeight`	`Long` Use this worker weight.

See Also Distributed Queue, page 183, in *TIBCO Rendezvous Concepts*
TibrvCmTransport.createDistributedQueue on page 279
TibrvCmTransport.getWorkerWeight() on page 284

Chapter 11 Errors

Topics

Error Overview

All Rendezvous COM objects support detailed error information.

VB programmers can see a list of codes in the object browser, in the category tibrvcom_errorCodes. Programmers using other languages can browse the type library to see these definitions.

Rendezvous COM methods return errors as HRESULT values—32-bit integers with hex representations 0x8004nnnn. These values divide into two ranges:

- Values in the range [0x80040200, 0x80040fff] represent errors from the TIBRVCOM implementation. For details, see Component Error Codes on page 297.
- Values in the range [0x80048001, 0x80048fff] represent errors from the Rendezvous API. For details, see Rendezvous Error Codes on page 293.

VB

When VB reports an error, it populates an Err object.

VB programmers can write error handlers to process such errors. Err.Number extracts the error status code. Err.Source identifies the object that reported the error. Err.Description interprets the error code as a descriptive string.

- Err.Number extracts the HRESULT numeric error code.
- Err.Source identifies the object that reported the error.
- Err.Description interprets the error code as a descriptive string.

C++

C++ programs must test the HRESULT status code after each method call. Microsoft Visual C++ defines these convenient macros:

```
#define FAILED(Status)    ((HRESULT)(Status) <0)
#define SUCCEEDED(Status) ((HRESULT)(Status)>=0)
```

C++ COM programmers can use the IErrorInfo interface to extract more detail from an object that returns an HRESULT indicating an error:

- GetErrorInfo() returns the error object.
- GetErrorSource() identifies the object that reported the error.
- GetErrorDescription() interprets the error code as a descriptive string.

Continuing after Errors

Although continuing after an error is not generally considered a good programming practice, it is sometimes necessary. When you must continue after an error, consider these guidelines.

- A method that usually returns an object instead returns an uninitialized object.
- A method that usually returns a value instead returns zero.
- When an error results from incorrect assignment syntax, the program may call the method again with the correct syntax. Consider these two examples:

```
      On Error Resume Next
      value = msg.getByIndex(i)
      If IsEmpty(value) then
       Set value = msg.getByIndex(i)
      End If
```

```
     For i=0 To msg.getNumFields
      On Error Resume Next
      var = msg.getByIndex(i)     'This won't work for objects.
      If (IsArray(var)) Then
       For j=LBound(var) To UBound(var)
        Write #1, "Array element " & CStr(j) & ": " & CStr(var(j))
       Next j
      Else                        'It's not an array.
       If Not IsEmpty(var) Then   'Perhaps it's an scalar.
        Write #1, CStr(var)
       End If
      End If
      If IsEmpty(var)
       Set var = msg.getByIndex(i) 'Try to extract it as an object.
       Write #1, "Extracted " & TypeName(var) & " object from
message"
       If TypeName(var) = "ITIBRVMSG" Then  'It is an object.
        Write #1, var.toString     'Write the object.
       End If
       If TypeName(var) = "ITibrvInt64" Then
        Write #1, var.getAsString
        Write #1, " or as unsigned " & var.getAsUnsignedString
        Write #1, Cstr(var.getHighOrderPart) & ", " & _
                  CStr(var.getLowOrderPart)
       End If
      End If
```

Constants

The interface defines these useful constants.

Table 6 Base Constants for HRESULT Computations

Constant Hex Code	Description
TIBRVCOM_ERROR_ITF 0x80040000	Base of all HRESULT codes from the TIBRVCOM interface.
TIBRVCOM_ERROR_TIBRVCOM 0x80040200	Base of HRESULT codes from the TIBRVCOM implementation. See Component Error Codes on page 297.
TIBRVCOM_ERROR_TIBRV 0x80048000	Base of HRESULT codes from Rendezvous API. See Rendezvous Error Codes on page 293.

Rendezvous Error Codes

Rendezvous API error codes are 32-bit integers, with hex representations `0x80048nnn`.

These constants generally correspond to error codes in the Rendezvous C API—when the COM interface constant `TIBRVCOM_`*name* has the value `0x80048`*nnn*, the C constant `TIBRV_`*name* has the value *nnn*.

Table 7 Rendezvous HRESULT Error Codes (Sheet 1 of 4)

Constant Hex Code	Description

`TIBRVCOM_INIT_FAILURE`
`0x80048001`
	Cannot create the network transport.

`TIBRVCOM_INVALID_ARG`
`0x80048003`
	An argument is invalid. Check arguments other than messages, subject names, transports, events, queues and queue groups (which have separate error codes).

`TIBRVCOM_NOT_INITIALIZED`
`0x80048004`
	The method cannot run because the Rendezvous environment is not initialized (open).

`TIBRVCOM_ARG_CONFLICT`
`0x80048005`
	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.

`TIBRVCOM_SERVICE_NOT_FOUND`
`0x80048010`
	Transport creation failed; cannot match the service name using `getservbyname()`.

`TIBRVCOM_NETWORK_NOT_FOUND`
`0x80048011`
	Transport creation failed; cannot match the network name using `getnetbyname()`.

`TIBRVCOM_DAEMON_NOT_FOUND`
`0x80048012`
	Transport creation failed; cannot match the daemon port number.

`TIBRVCOM_NO_MEMORY`
`0x80048013`
	The method could not allocate dynamic storage.

`TIBRVCOM_INVALID_SUBJECT`
`0x80048014`
	The method received a subject name with incorrect syntax.

`TIBRVCOM_DAEMON_NOT_CONNECTED`
`0x80048015`
	The Rendezvous daemon process (`rvd`) exited, or was never started. This error indicates that the program cannot start the daemon and connect to it.

Table 7 Rendezvous HRESULT Error Codes (Sheet 2 of 4)

Constant Hex Code	Description

`TIBRVCOM_VERSION_MISMATCH`
`0x80048016`
	The library, header files and Rendezvous daemon are incompatible.

`TIBRVCOM_SUBJECT_COLLISION`
`0x80048017`
	It is illegal to create two certified worker events on the same CM transport with overlapping subjects.

`TIBRVCOM_VC_NOT_CONNECTED`
`0x80048018`
	A virtual circuit terminal was once complete, but is now irreparably broken.

`TIBRVCOM_NOT_PERMITTED`
`0x8004801b`
	1. The program attempted an illegal operation.
	2. Cannot create ledger file.

`TIBRVCOM_INVALID_NAME`
`0x8004801e`
	The field name is too long; see Field Name Length on page 46.

`TIBRVCOM_INVALID_TYPE`
`0x8004801f`
	1. The field type is not registered.
	2. Cannot update field to a type that differs from the existing field's type.

`TIBRVCOM_INVALID_SIZE`
`0x80048020`
	The explicit size in the field does not match its explicit type.

`TIBRVCOM_INVALID_COUNT`
`0x80048021`
	The explicit field count does not match its explicit type.

`TIBRVCOM_NOT_FOUND`
`0x80048023`
	Could not find the specified field in the message.

`TIBRVCOM_ID_IN_USE`
`0x80048024`
	Cannot add this field because its identifier is already present in the message; identifiers must be unique.

`TIBRVCOM_ID_CONFLICT`
`0x80048025`
	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).

`TIBRVCOM_CONVERSION_FAILED`
`0x80048026`
	Found the specified field, but could not convert it to the desired datatype.

`TIBRVCOM_RESERVED_HANDLER`
`0x80048027`
	The datatype handler number is reserved for Rendezvous internal datatype handlers.

Table 7 Rendezvous HRESULT Error Codes (Sheet 3 of 4)

Constant Hex Code	Description

`TIBRVCOM_ENCODER_FAILED`
`0x80048028`
	Datatype encoder failed.

`TIBRVCOM_DECODER_FAILED`
`0x80048029`
	Datatype decoder failed.

`TIBRVCOM_INVALID_MSG`
`0x8004802a`
	The method received a message argument that is not a well-formed message.

`TIBRVCOM_INVALID_FIELD`
`0x8004802b`
	The program supplied an invalid field as an argument.

`TIBRVCOM_INVALID_INSTANCE`
`0x8004802c`
	The program supplied zero as the field instance number (the first instance is number 1).

`TIBRVCOM_CORRUPT_MSG`
`0x8004802d`
	The method detected a corrupt message argument.

`TIBRVCOM_ENCODING_MISMATCH`
`0x8004802e`
	The method attempted to add or update a field within a message. However, the encoding in the new value does not match the encoding of the message.

`TIBRVCOM_TIMEOUT`
`0x80048032`
	A timed dispatch call returned without dispatching an event.
	A send request call returned without receiving a reply message.
	A virtual circuit terminal is not yet ready for use.

`TIBRVCOM_INTR`
`0x80048033`
	Interrupted operation.

`TIBRVCOM_INVALID_DISPATCHABLE`
`0x80048034`
	The method received an event queue or queue group that has been destroyed, or is otherwise unusable.

`TIBRVCOM_INVALID_EVENT`
`0x8004803c`
	The method received an event that has been destroyed, or is otherwise unusable.

`TIBRVCOM_INVALID_CALLBACK`
`0x8004803d`
	The method received `NULL` instead of a callback method.

`TIBRVCOM_INVALID_QUEUE`
`0x8004803e`
	The method received a queue that has been destroyed, or is otherwise unusable.

Table 7 Rendezvous HRESULT Error Codes (Sheet 4 of 4)

Constant Hex Code	Description
TIBRVCOM_INVALID_QUEUE_GROUP 0x8004803f	The method received a queue group that has been destroyed, or is otherwise unusable.
TIBRVCOM_INVALID_TIME_INTERVAL 0x80048040	The method received a negative timer interval.
TIBRVCOM_SOCKET_LIMIT 0x80048043	The operation failed because of an operating system socket limitation.
TIBRVCOM_OS_ERROR 0x80048044	`Tibrv.open` encountered an operating system error.
TIBRVCOM_EOF 0x80048047	End of file.
TIBRVCOM_INVALID_FILE 0x80048048	1. A certificate file or a ledger file is not recognizable as such. 2. `TibrvSdContext.setUserCertWithKey()` or `TibrvSdContext.setUserCertWithKeyBin()` 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.
TIBRVCOM_FILE_NOT_FOUND 0x80048049	Rendezvous software could not find the specified file.
TIBRVCOM_IO_FAILED 0x8004804a	Cannot write to ledger file.
TIBRVCOM_NOT_FILE_OWNER 0x80048050	The program cannot open the specified file because another program owns it. For example, ledger files are associated with correspondent names.

Component Error Codes

TIBRVCOM component error codes are 32-bit integers, with hex representations 0x80040*nnn*. The type library defines these error code constants.

These constants correspond to string resources—when the constant TIBRVCOM_*name* has the value 0x80040*nnn*, the resource IDS_E_*name* has the value *nnn*.

Table 8 TIBRVCOM HRESULT Error Codes (Sheet 1 of 3)

Constant Hex Code	Description
TIBRVCOM_TIBRV_NOT_OPEN 0x80040200	Operation requires that Tibrv be open.
TIBRVCOM_INVALID_MSG_OBJECT 0x80040201	TibrvMsg has no internal structure.
TIBRVCOM_BAD_POINTER_ARG 0x80040202	Argument must be a pointer to existing storage.
TIBRVCOM_INVALID_ARRAY 0x80040203	Arrays must be one-dimensional.
TIBRVCOM_INVALID_DATATYPE 0x80040204	Datatype not supported for this operation.
TIBRVCOM_INVALID_QUEUE_OBJECT 0x80040205	Queue object has no internal structure.
TIBRVCOM_INVALID_TRANSPORT_OBJECT 0x80040206	Transport object has no internal structure.
TIBRVCOM_ALREADY_INITIALIZED 0x80040207	Object is already created or initialized.
TIBRVCOM_CREATE_WINDOW_FAILED 0x80040208	Unable to create hidden window for automatic dispatch.
TIBRVCOM_INVALID_FIELD_ID 0x80040209	Invalid field id—range is 0 to 65535.
TIBRVCOM_DATA_CONVERSION_RANGE 0x8004020a	Data out of range for conversion to target type.

Table 8 TIBRVCOM HRESULT Error Codes (Sheet 2 of 3)

Constant Hex Code	Description

`TIBRVCOM_NOT_CREATED`
`0x8004020b`
	Unable to create object.

`TIBRVCOM_OUT_OF_MEMORY`
`0x8004020c`
	Insufficient memory for operation.

`TIBRVCOM_INVALID_VALUE`
`0x8004020d`
	Invalid value of data argument.

`TIBRVCOM_NULL_NAME`
`0x8004020e`
	Null name parameter is illegal.

`TIBRVCOM_MSG_NOT_CREATED`
`0x8004020f`
	Unable to create message object.

`TIBRVCOM_NULL_SUBJECT`
`0x80040210`
	Null subject is illegal.

`TIBRVCOM_CONVERSION_NOT_SUPPORTED`
`0x80040211`
	Data conversion between these types is not supported.

`TIBRVCOM_DATA_CONVERSION_FAILED`
`0x80040212`
	Data conversion failed.

`TIBRVCOM_CANNOT_RETRIEVE`
`0x80040213`
	Cannot retrieve the requested information.

`TIBRVCOM_INVALID_PROPERTY_REQUEST`
`0x80040214`
	No property with that name exists.

`TIBRVCOM_INVALID_PROPERTY_VALUE`
`0x80040215`
	Property value data type is incorrect

`TIBRVCOM_NOT_DISTRIBUTED_TRANSPORT`
`0x80040216`
	Operation is valid only for distributed CM transport.

`TIBRVCOM_NOT_FOR_DISTRIBUTED_TRANSPORT`
`0x80040217`
	Operation is not allowed for distributed CM transport.

`TIBRVCOM_EVENT_CALLBACK_ACTIVE`
`0x80040218`
	Dispatching a queue from a callback is not allowed

Table 8   TIBRVCOM HRESULT Error Codes (Sheet 3 of 3)

Constant Hex Code	Description

`TIBRVCOM_OBJECT_NOT_INITIALIZED`
`0x80040219`
	The object's internal structure has not been created (or has already been destroyed).

`TIBRVCOM_UNABLE`
`0x8004021a`
	Object cannot complete the requested operation.

`TIBRVCOM_INVALID_CLOSURE`
`0x8004021b`
	Invalid closure argument.

VB Error Codes

Programs can receive errors that seem related to Rendezvous software, but really originate from the VB language implementation. While it is difficult to give complete advice to application programmers regarding these errors, Table 9 summarizes some of the common errors in this group.

Table 9 VB Errors (Sheet 1 of 2)

Error	Error Text String and Description

91	Object variable or With block variable not set
	VB cannot apply a method because the variable does not contain an appropriate object.
	This error can indicate that an object has not yet been assigned to the variable, or was once present but has been set to `Nothing`.
	This error can also stem from incorrect assignment syntax. Remember that you must use `set` to assign an object; for example:
	Right:
	`'in General (Global) Declarations`
	`  dim q as TibrvQueue`
	`'in a Subroutine`
	`  **set** q = tibrv.getDefaultQueue`
	Wrong:
	`'in General (Global) Declarations`
	`  dim q as New TibrvQueue`
	`'in a Subroutine`
	`  q = tibrv.getDefaultQueue`

429	COM rejected a call to `CreateObject()`. Check spelling of object name. Check that the component is registered. Check memory usage.

Table 9 VB Errors (Sheet 2 of 2)

Error	Error Text String and Description

438	Object doesn’t support this property or method
	Check spelling of method name and its arguments.
	Check that the component is registered.
	This error also indicates incorrect assignment syntax. Remember that you must use `set` to assign an object; for example:
	Right:
	`'in General (Global) Declarations`
	`  dim q as TibrvQueue`
	`'in a Subroutine`
	`  `**`set`**` q = tibrv.getDefaultQueue`
	Wrong:
	`'in General (Global) Declarations`
	`  dim q as TibrvQueue`
	`'in a Subroutine`
	`  q = tibrv.getDefaultQueue`

Index