JADE™

JADE TCP/IP Connection and Worker Framework

Jade Software Corporation Limited cannot accept any financial or other responsibilities that may be the result of your use of this information or software material, including direct, indirect, special or consequential damages, or loss of profits. There are no warranties extended or granted by this document or software material.

You should be very careful to ensure that the use of this software material and/or information complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used. No part of this document may be reproduced or transmitted in any form or by 4any means, electronic or mechanical, for any purpose, without the express written permission of Jade Software Corporation Limited.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Copyright © 2005 Jade Software Corporation Limited.

All rights reserved.

JADE is a trademark of Jade Software Corporation Limited. All trade names referenced are the service mark, trademark, or registered trademark of the respective manufacturer.

Contents

JADE TCP/IP Connection and Worker Framework **3**
Endpoints, Groups and Pools 4
Workers 4
Connections 5
Establishing a Transport Group 5
Exceptions 6
Events 6
Sending and Receiving Data 8
Connection Binding 9
Dynamic Worker Pool 10
Statistics 10
Examples 11
Example - WWWServ 11
Running the Example 12
Example - ChatServ and ChatClient 12
ChatServ 12
ChatClient 13
Running the Example 13

Copyright 2005, Jade Software Corporation Ltd. All rights reserved

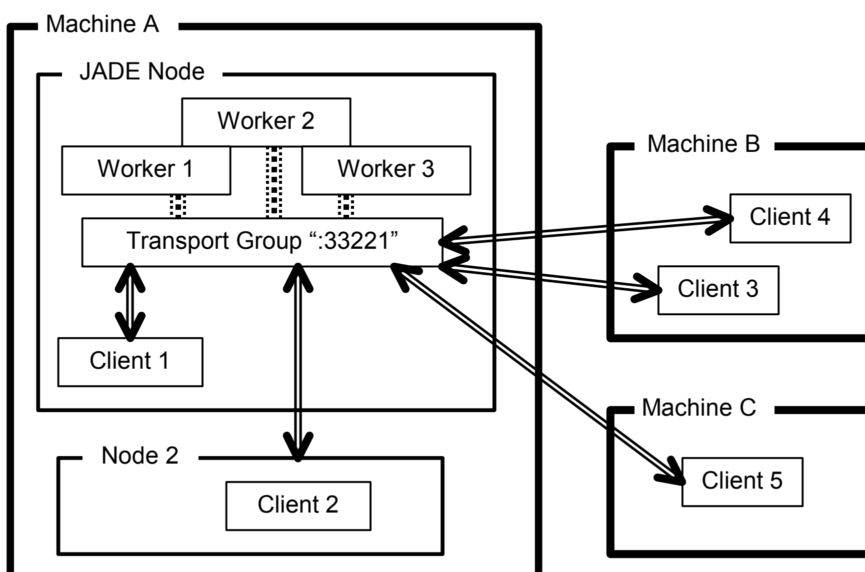
JADE TCP/IP Connection and Worker Framework

The JADE TCP/IP Connection and Worker Framework is a new facility in JADE 6.1. This facility encapsulates the server side or listen end of a TCP/IP connection, allowing a variable number of JADE processes in a node to handle any one of the associated client connections.

Features:

- Delivers robust TCP/IP-based applications with less code
- Automatic worker and connection pool management
- Automatic round-robin delivery of connection events to the next idle worker
- No management logic or management application required
- Support for connection session data
- Comprehensive statistics
- Idle connection detection
- Connection event queue is first-in/first-out
- Worker idle list is last-in/first-out
- Type-safe callback methods using a JADE interface

The following diagram shows a transport group with three workers handling connections from five clients spread over three machines.



Copyright 2005, Jade Software Corporation Ltd. All rights reserved

The classes **JadeMultiWorkerTcpTransport** and **JadeMultiWorkerTcpConnection** implement this facility.

There are many similarities with the existing **TcpIpConnection** class. Properties and methods that have the same name also have the same usage and behaviour.

Endpoints, Groups and Pools

Each **JadeMultiWorkerTcpTransport** server connection endpoint (or Transport Group) has a worker pool and a client connection pool.

The **fullName** property of the Transport Group is a string containing the **listenHostname** and **listenPortnumber** properties separated by a colon. The **listenHostname** can be an empty string. Examples are **":12345"**, **"myhost.somewhere.com:8088"**, and **"127.0.0.1:7654"**.

The worker pool consists of those JADE application processes that are attached to the transport group.

Note All worker processes must run in the same JADE node.

The client connection pool consists of the established TCP connections that client processes (on the same or other machines) have successfully opened to the server connection endpoint.

A worker can handle messages from any client connection.

The next available worker handles a message arriving on a client connection. A different worker can handle the next message from that same connection. Connection binding provides an override of this default behaviour so that a single worker can handle all activity for a specific connection.

The TCP/IP listen connection is automatically created and opened when the first worker of each transport group invokes the **beginListening** method.

The **userGroupObject** property of **JadeMultiWorkerTcpTransport** can be used to provide a common shared transient or persistent object to all workers in a transport group.

Workers

A worker is a JADE application process.

A worker joins a Transport Group by creating a **JadeMultiWorkerTcpTransport** instance, setting the **listenPortnumber** property and optionally the **listenHostname** property, and then invoking the **beginListening** method.

The worker pool associated with the Transport Group contains a minimum of one worker.

When the last worker leaves the worker pool by deleting the **JadeMultiWorkerTcpTransport** instance it used to join the pool, the listen connection and all client connections are closed and the transport group is deleted.

A worker can attach to multiple transport groups. It uses a separate instance of **JadeMultiWorkerTcpTransport** for each group it joins.

Copyright 2005, Jade Software Corporation Ltd. All rights reserved

Connections

A client connection appears to a worker application as a transient instance of the **JadeMultiWorkerTcpConnection** class. This instance is used to send and receive data.

A client connection is either idle, queued, or assigned.

A connection is idle when it has no queued events such as data ready for reading. A connection is queued when it has one or more events ready for a worker to process. A connection is assigned when a worker begins handling an event for the connection.

A worker can only manipulate a connection (read data, write data, or access properties) when it is assigned to that worker. A connection can only be assigned to one worker at any time. A connection is assigned to a worker when that worker begins processing an event for that connection by removing the connection from the connection ready queue just prior to the callback method invocation.

The connection becomes unassigned when the callback method exits.

Connection assignment prevents multiple workers from simultaneously manipulating a connection, avoiding the need for any user-coded connection locking strategy.

The **userObject** property of **JadeMultiWorkerTcpConnection** can be used to provide a common shared transient or persistent object for storing connection session data that is visible to all workers in a transport group. Create this object during your **openedEvent** method and delete it during your **closedEvent** method. The **userState** property is an Integer value that is also visible to all workers in a transport group.

Establishing a Transport Group

A JADE application executes code similar to the following example, to establish a new transport group or join an existing transport group.

```
begin
      create self.myMWTT transient;
      self.myMWTT.listenPortnumber := 33221;
      self.myMWTT.beginListening();
      self.myMWTT.notifyEventsAsync(self,
                        JadeMultiWorkerTcpTransport.Notify_Continuous);
end;
```

In this example, the transport group **fullName** is **":33221"**.

If the JADE node in which the process executing the **beginListening** method has no transport group with the fullName **":33221"**, the group is created and the TCP/IP listen endpoint is opened. When this method completes successfully, you can use a command such as **netstat** to check the listen endpoint is as expected.

```
C:\>netstat -an
Active Connections
  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:7              0.0.0.0:0              LISTENING
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:33221          0.0.0.0:0              LISTENING
```

In this case, the local address is **"0.0.0.0"** because **listenHostname** property was left as its default value of an empty string (**""**).

Copyright 2005, Jade Software Corporation Ltd. All rights reserved

The first requirement for a JADE process to be able to join an existing transport group is that it must be running in the same JADE node in which the group was created.

The second requirement to be able to join an existing transport group is that exactly the same values are assigned to the **listenPortnumber** and **listenHostname** properties.

The third requirement to be able to join an existing transport group is that the class (or subclass) of **JadeMultiWorkerTcpTransport** must be the same. This requirement makes the user reimplementation of the **validateServerProcess** method effective. This method can check process attributes such as **app.name** or **process.userCode**, and if necessary, refuse the attach attempt by returning **false**. If this method returns **false,** the **beginListening** method call will fail with an exception (31117).

Exceptions

Various exceptions can be raised when transport or connection properties are updated or methods are called.

For example, if a connection closes while the **readBinary** method is receiving a message, a **Connection** exception 31017 is raised.

It is recommended that a global exception handler be armed to catch **Connection** exceptions and that appropriate logic be written to at least handle exceptions raised during read and write method calls.

An exception raised in or above an event callback method is considered unhandled if it is not intercepted by an exception handler that returns **Ex_Resume_Next**. All unhandled exceptions that cut back through a connection event method cause the connection to be closed.

Events

Callback methods deliver connection and management events to the worker processes.

A worker uses the **notifyEventsAsync** method on its transport object to nominate the receiver object for the callbacks. The receiver object is an instance of a class that implements the **JadeMultiWorkerTcpTransportIF** interface. This interface defines the callback methods and defines constants for use as method parameters.

The connection-related event callback methods are:

- **closedEvent**

  This callback executes logic to tidy up after a connection closes. Both locally and remotely initiated closes queue this event.

- **connectionEvent**

  This is a multifunction callback method and the **type** parameter specifies the particular event type. Currently (JADE 6.1) the only event type value is:

  - **ConnEvType_IdleTimeout**. The connection idle timeout event occurs after the connection has no input for the period of time specified in the connection property **idleTimeout**.

- **openedEvent**

  This callback executes logic to initialise a connection. For example setting the connection **idleTimeout** property or creating the connection **userObject** property. It

Copyright 2005, Jade Software Corporation Ltd. All rights reserved

can also check the client address, and if necessary, close the connection to deny access.

Copyright 2005, Jade Software Corporation Ltd. All rights reserved

- **readReadyEvent**

  This callback executes logic to process input data queued on the connection. This is the only callback method that must have logic and it must read at least one byte of data to avoid a continuous **readReadyEvent** loop.

- **userEvent**

  This callback executes logic to handle a user event. The connection **causeUserEvent** and the transport **causeUserEventOnConnId** methods queue a user event for a connection. A connection can have one queued user event only. Each **causeUserEvent** method discards the previous undelivered user event.

The Transport Group related event callback method is:

- **managementEvent**

  This is a multifunction callback method and the **type** parameter specifies the particular event type. Currently (JADE 6.1) the event type values are:

  - **MgmntEvType_WorkerIdleTimeout**. This event occurs when a worker has been idle for more that the time period specified in the property **workerIdleTimeout**.
  - **MgmntEvType_QueueDepthExceeded**. This event occurs when the number of ready connections has exceeded the value in the property **queueDepthLimit** for the time period specified in the property **queueDepthLimitTimeout**.

Sending and Receiving Data

A TCP/IP connection treats the data being sent or received as a continuous stream of bytes. The application sending or receiving the data is responsible for deciding where one message stops and the next one begins. You must decide not only what data to transfer but also the packaging to be used.

A common way of encoding the data to be sent is XML, which encodes all data as plain text characters and therefore avoids architecture issues such as big- and little-endian byte-ordering. It is also much easier to read.

A simple envelope for defining message boundaries is to precede the content with a fixed-size header that includes the length of the content. The **ChatServ** example described later in this document uses this envelope style. It uses an eight-byte header that consists of eight digits whose value is the length of the remainder of the message. This envelope has the advantage that the receiver knows how long each message is as soon as it has received the header. The disadvantage is that the sender must build the entire message before sending it so that it can be prefixed with the length.

Another common envelope uses HTTP (HyperText Transport Protocol) request and reply headers. This is commonly used by Web services.

Any one of the workers in a transport group can send or receive data on a connection, provided that the connection is assigned to that worker. A connection can only be assigned to a single worker at a time, so no additional locking is required to ensure that multiple workers do not attempt to send and receive data simultaneously.

Copyright 2005, Jade Software Corporation Ltd. All rights reserved

It is recommended that the text in a message be limited to characters in the US ASCII 7-bit character set for maximum compatibility with other hardware architectures and operating systems. If extended or Unicode characters are required, it is recommended that the entire message content be encoded in UTF-8 format. Special attention is required when using an ANSI version of JADE under multiple locales, as the extended characters (those between decimal 127 and 255) can be interpreted differently in each locale.

A worker uses the **JadeMultiWorkerTcpConnection** class methods **writeBinary** and **writeBinaryAndFile** to send data. Both methods place the contents of the binary buffer parameter on the wire in the order that the bytes appear in the buffer. The **writeBinaryAndFile** method has additional parameters that allow all or a section of a file to be sent following the contents of the buffer. An HTTP reply message can easily be sent by a Web server using this method, as it consist of a header followed by the contents of a file. (See the **WWWServ** example.)

A worker uses the **JadeMultiWorkerTcpConnection** class methods **readBinary** and **readUntil** to receive data.

The **readBinary** method returns a Binary containing at most the number of bytes specified in the **length** parameter. If the connection property **fillReadBuffer** is **true**, the read returns the specified length, if necessary waiting until enough bytes have been received. If **fillReadBuffer** is **false**, only the bytes already received from the wire are returned. When using length delimited messages, set **fillReadBuffer** to **true** and make two **readBinary** calls. The first call specifies a maximum length equal to the number of bytes in the message length (for example, eight). Convert the header text to the body length and use this new length as the parameter for the second read.

The **readUntil** method also returns a JADE Binary containing at most the number of bytes specified in the **maxLength** parameter. It also stops copying received bytes into the binary and returns when it encounters the bytes specified in the **delimiter** parameter. The delimiter bytes are discarded. This method is used when a message or message part is delimited by a specific byte or bytes. An HTTP request or reply header is an example of this format, as it consists of a number of lines each terminated by a carriage-return and linefeed (CRLF) and terminated by an empty line. The **WWWServ** example demonstrates working with this message format.

Unlike the **TcpIpConnection** class, the **JadeMultiWorkerTcpConnection** class only has synchronous I/O methods that do not return until the requested action has completed. The **JadeMultiWorkerTcpConnection** class **timeout** property can be used to prevent a worker hanging “forever” waiting for a read or write to complete. When this timeout expires, a connection exception is raised and data for an incomplete read or an incomplete write is discarded. The simplest recovery from this state is to close the connection (which happens automatically if no exception handler is armed).

Connection Binding

By default, a connection event is handled by any available worker. This means that any session data for a connection that is retained from one message to the next must be visible to all the workers.

Use connection binding to direct all events for a connection to a single worker.

A connection can be bound and unbound multiple times and the connection can be bound to a different worker each time.

Copyright 2005, Jade Software Corporation Ltd. All rights reserved

A worker can bind only a connection that is assigned to itself. It can bind the connection to itself using the connection **bindToAssignedWorker** method or it can bind the connection to another worker using the **bindToWorkerId** method, specifying the **workerId** of the other worker process.

Only the worker to which the connection is bound can **unbind** the connection.

Dynamic Worker Pool

Properties and events are available that allow the number of worker applications to be adjusted to meet the current workload.

When the size of the connection ready queue exceeds the **queueDepthLimit** property value for the time period specified by **queueDepthLimitTimeout** property, a management event **QueueDepthExceeded** is queued. The **managementEvent** callback method can start additional worker applications to handle the queued connections.

When a worker has a non-zero **workerIdleTimeout** property and that worker has been idle for that period, the worker is sent a **WorkerIdleTimeout managementEvent**. The **managementEvent** callback method can choose to terminate the worker application if there are more than the required minimum number of workers.

Statistics

Statistics are available for an individual worker (**getMyStatistics**) and also for the Transport Group (**getGroupStatistics**).

The following code obtains and displays worker statistics.

```
vars
    jdo : JadeDynamicObject;

begin
    create jdo transient;
    myMWTT.getMyStatistics(0, jdo);
    write jdo.display();
epilog
    delete jdo;
end;
```

This example produces the following output (in JADE 6.1).

```
---MWTTransportStatsSet(402)---
fullName = :33221
groupId = 7
workerId = 1
created = 30 June 2005, 12:35:08
elapsedTime = 14151337
busyCount = 32
busyTime = 517427
callbackEtcCount = 2
callbackEtcTime = 34081
callbackReadReadyCount = 30
callbackReadReadyTime = 471288
connClosedCount = 0
connOpenedCount = 2
connReadReadyCount = 30
connUserEventCount = 0
```

Copyright 2005, Jade Software Corporation Ltd. All rights reserved

```
idleCount = 6
idleTime = 8461347
lastWentIdleTime = 5151557
lockAcquireCount = 129
lockAcquireTime = 249
wakeupsDiscarded = 0
wakeupsNoaction = 0
```

The values whose names end in **Time** are measured in microseconds (millionths of a second). Retrieve these from the dynamic object into **Decimal[18]** (or wider) local variables.

Examples

The following examples are available in the example schema file **MWTTExample**.

Example - WWWServ

This example demonstrates how to receive a message on a client connection and send a reply back to the client. It also uses the **writeBinaryAndFile** method to send the contents of a file.

This example is a simple Web server that implements the HTTP protocol. It assumes that each request contains a simple file name. If it finds the file present in the **http** directory subordinate to the database path (**system**) directory, it sends the contents of the file back to the client; otherwise it sends an HTTP 404 (Not Found) error.

Caution This example does *not* provide a Web server that is suitable for exposing to the Internet.

The server side is implemented in the **WWWServ** form class.

The logic is coded so that multiple workers can run together. The menu item File|Fork launches another copy of the application.

The **WWWServ** class **load** method creates and initialises the **MWTT** object by setting the **listenPortnumber** property and then calling the **beginListening** and **notifyEventsAsync** methods.

A connection event delivered via the **mwtt_readReadyEvent** method advises the server that a client connection has a message available. This method reads the HTTP header from the connection a line at a time, using the **readUntil** method. It breaks up the first line of the header and saves the various request details. It receives and discards the message body if it exists. If the request verb is not **GET,** it sends a 405 (method not allowed) reply. If the request URI is **/home**, it calls the **getHomePage** method to build an HTML page to send as the reply. Otherwise it calls the **sendAFile** method, which attempts to find a file in the **html** directory and send it back to the client.

Copyright 2005, Jade Software Corporation Ltd. All rights reserved

Running the Example

1. Launch the **WWWServ** application.
2. Launch a Web browser (on the same machine).
3. Enter the **http://localhost:33221/home** URL.
4. This should return the built-in home page. When you click the browser **Refresh** button, the time displayed on the page should update.
5. Create a directory called **html** below the system directory for your JADE environment. Place a text file or a static Web page along with its images in this directory.
6. Enter the **http://localhost:33221/xxx** URL, where **xxx** is the name of the text file or Web page.
7. If the Web page includes a number of images and you run a second worker (menu File|Fork), you will see the HTTP requests handled by both workers.

Example - ChatServ and ChatClient

This example is a simple Chat client/server application in which client applications connect to a server application to exchange messages.

This example demonstrates how to receive a message on one client connection and send it out via another client connection attached to the same transport group.

ChatServ

The server side of this example is implemented in the **ChatServ** form class.

The logic is coded so that multiple worker applications can run together. The menu item File|Fork launches another copy of the application.

The application uses an instance of the **JadeMultiWorkerTcpTransport** class to manage the client connections.

The **ChatServ** form **load** method creates and initialises the **MWTT** object by setting the **listenPortnumber** property then calling the **beginListening** and **notifyEventsAsync** methods.

It also locates or creates the shared **ClientData** collection by calling the **createClientTable** method that stores it in the transport **userGroupObject** property.

An **openedEvent** delivered via the **mwtt_openedEvent** method advises the server that a new client has connected. This method creates a shared transient **ClientData** object for the connection and adds it to the global collection.

A **closedEvent** delivered via the **mwtt_closedEvent** method advises the server that a client connection has closed. This method removes the **ClientData** object from the global list and deletes it.

A **readReadyEvent** delivered via the **mwtt_readReadyEvent** method advises the server application that a client connection has a message available. This method reads the message from the connection. The message format is length-delimited (eight decimal digits) with the first character of the body defining the message type.

Copyright 2005, Jade Software Corporation Ltd. All rights reserved

If the message type is text (**$**), the **broadcastMessage** method queues a message and a **userEvent** for each of the other current client connections by adding the text to the **ClientData** msgList and calling the **causeUserEventOnConnId** method.

A **userEvent** delivered via the **mwtt_userEvent** method advises the server application that it should check the client connection for messages to send. This method calls the **sendQueuedMessages** method that loops, removing messages from the list and sending them via the connection to the client application.

ChatClient

The client side of this example is implemented in the **ChatClient** form class.

It uses an instance of the **TcpIpConnection** class to connect to the server application. It uses the **readBinaryAsynch** method to read messages from the server and displays the content of the messages received in the upper text box.

The user must enter text in the nickname text box then click the menu item File|Connection to establish the connection to the server application. The user enters text in the lower text box and clicks the **Send** button to send the text to the server where it is broadcast to all other connected client applications.

Running the Example

1. Launch the **ChatServ** application.
2. Click on the menu item File|Log All.
3. Launch the first copy of the **ChatClient** application.
4. Enter a name in the nickname text box.
5. Click on menu item File|Connect.
6. The client should connect to the server, which displays a message in the text box.
7. Launch additional clients, ensuring that each has a unique nickname.
8. Enter text in the lower text box on the client and click the **Send** button to broadcast it to the other clients.

Copyright 2005, Jade Software Corporation Ltd. All rights reserved