Overview
- Home
- News
- FAQ
- Download

Documentation
- Protocol
- Stomp JMS
- Articles

Software
- Clients
- Brokers
- StompConnect

Community
- Contributing
- Site
- Wiki
- Mailing Lists
- Team

Support
- Issues
- Roadmap
- Change log

Developers
- Subversion

Feeds
- Site
- News

Protocol

Stomp Protocol Specification, Version 1.0

Initially the client must open a socket (I'm going to presume TCP, but really it is kind of irrelevant). The client then sends:

```
CONNECT
login: <username>
passcode:<passcode>

^@
```

The ^@ is a null (control-@ in ASCII) byte. The entire thing will be called a Frame in this doc. The frame starts with a command (in this case CONNECT), followed by a newline, followed by headers in a <key>:<value> with each header followed by a newline. A blank line indicates the end of the headers and beginning of the body (the body is empty in this case), and the null indicates the end of the frame.

After the client sends the CONNECT frame, the server will always acknowledge the connection, by sending a frame which looks like:

```
CONNECTED
session: <session-id>

^@
```

The session-id header is a unique identifier for this session (though it isn't actually used yet).

At this point there are a number of commands the client may send

- SEND
- SUBSCRIBE
- UNSUBSCRIBE
- BEGIN
- COMMIT
- ABORT
- ACK
- DISCONNECT

Client Commands

SEND

The SEND command sends a message to a destination in the messaging system. It has one required header, **destination**, which indicates where to send the message. The body of the SEND command is the message to be sent. For example:

```
SEND
destination:/queue/a

hello queue a
^@
```

This sends a message to the **/queue/a** destination. This name, by the way, is arbitrary, and despite seeming to indicate that the destination is a "queue" it does not, in fact, specify any such thing. Destination names are simply strings which are mapped to some form of destination on the server - how the server translates these is left to the server implementation. See **this note on mapping destination strings to JMS Destinations** for more detail.

SEND supports a **transaction** header which allows for transaction sends.

It is recommended that SEND frames include a content-length header which is a byte count for the length of the message body. If a content-length header is included, this number of bytes should be read, regardless of whether or not there are null characters in the body. The frame still needs to be terminated with a null byte and if a content-length is not specified, the first null byte encountered signals the end of the frame.

SUBSCRIBE

The SUBSCRIBE command is used to register to listen to a given destination. Like the SEND command, the SUBSCRIBE command requires a **destination** header indicating which destination to subscribe to. Any messages received on the subscription will henceforth be delivered as MESSAGE frames from the server to the client. The **ack** header is optional, and defaults to **auto**.

```
SUBSCRIBE
destination: /queue/foo
ack: client

^@
```

In this case the **ack** header is set to **client** which means that messages will only be considered delivered after the client specifically acknowledges them with an ACK frame. The valid values for **ack** are **auto** (the default if the header is not included) and **client**.

The body of the SUBSCRIBE command is ignored.

Stomp brokers may support the **selector** header which allows you to specify an **SQL 92 selector** on the message headers which acts as a filter for content based routing.

You can also specify an **id** header which can then later on be used to UNSUBSCRIBE from the specific subscription as you may end up with overlapping subscriptions using selectors with the same destination. If an **id** header is supplied then Stomp brokers should append a **subscription** header to any MESSAGE commands which are sent to the client so that the client knows which subscription the message relates to. If using **Wildcards** and **selectors** this can help clients figure out what subscription caused the message to be created.

UNSUBSCRIBE

The UNSUBSCRIBE command is used to remove an existing subscription - to no longer receive messages from that destination. It requires either a **destination** header or an **id** header (if the previous SUBSCRIBE operation passed an id value). Example:

```
UNSUBSCRIBE
destination: /queue/a

^@
```

BEGIN

BEGIN is used to start a transaction. Transactions in this case apply to sending and acknowledging - any messages sent or acknowledged during a transaction will be handled atomically based on the transaction.

```
BEGIN
transaction: <transaction-identifier>

^@
```

The **transaction** header is required, and the transaction identifier will be used for SEND, COMMIT, ABORT, and ACK frames to bind them to the named transaction.

COMMIT

COMMIT is used to commit a transaction in progress.

```
COMMIT
transaction: <transaction-identifier>

^@
```

The **transaction** header is required, you must specify which transaction to commit!

ACK

ACK is used to acknowledge consumption of a message from a subscription using client acknowledgment. When a client has issued a SUBSCRIBE frame with the **ack** header set to **client** any messages received from that destination will not be considered to have been consumed (by the server) until the message has been acknowledged via an ACK.

ACK has one required header, **message-id**, which must contain a value matching the **message-id** for the MESSAGE being acknowledged. Additionally, a **transaction** header may be specified, indicating that the message acknowledgment should be part of the named transaction.

```
ACK
message-id: <message-identifier>
transaction: <transaction-identifier>

^@
```

The **transaction** header is optional.

ABORT

ABORT is used to roll back a transaction in progress.

```
ABORT
transaction: <transaction-identifier>

^@
```

The **transaction** header is required, you must specify which transaction to abort!

DISCONNECT

DISCONNECT does a graceful disconnect from the server. It is quite polite to use this before closing the socket.

```
DISCONNECT

^@
```

Standard Headers

Some headers may be used, and have special meaning, with most packets

Receipt

Any client frame other than CONNECT may specify a **receipt** header with an arbitrary value. This will cause the server to acknowledge receipt of the frame with a RECEIPT frame which contains the value of this header as the value of the **receipt-id** header in the RECEIPT packet.

```
SEND
destination:/queue/a
receipt:message-12345

Hello a!^@
```

Server Frames

The server will, on occasion, send frames to the client (in additional to the initial CONNECTED frame). These frames may be one of:

- MESSAGE
- RECEIPT
- ERROR

MESSAGE

MESSAGE frames are used to convey messages from subscriptions to the client. The MESSAGE frame will include a header, **destination**, indicating the destination the message was delivered to. It will also contain a **message-id** header with a unique identifier for that message. The frame body contains the contents of the message:

```
MESSAGE
destination:/queue/a
message-id: <message-identifier>

hello queue a^@
```

Would be a sample message.

It is recommended that MESSAGE frames include a **content-length** header which is a byte count for the length of the message body. If a **content-length** header is included, this number of bytes should be read, regardless of whether or not there are null characters in the body. The frame still needs to be terminated with a null byte, and if a **content-length** is not specified the first null byte encountered signals the end of the frame.

RECEIPT

Receipts are issued from the server when the client has requested a receipt for a given command. A RECEIPT frame will include the header **receipt-id**, where the value is the value of the **receipt** header in the frame

which this is a receipt for.

```
RECEIPT
receipt-id:message-12345

^@
```

The receipt body will be empty.

ERROR

The server may send ERROR frames if something goes wrong. The error frame should contain a **message** header with a short description of the error, and the body may contain more detailed information (or may be empty).

```
ERROR
message: malformed packet received

The message:
-----
MESSAGE
destined:/queue/a

Hello queue a!
-----
Did not contain a destination header, which is required for message propagation.
^@
```

It is recommended that ERROR frames include a **content-length** header which is a byte count for the length of the message body. If a **content-length** header is included, this number of bytes should be read, regardless of whether or not there are null characters in the body. The frame still needs to be terminated with a null byte, and if a **content-length** is not specified the first null byte encountered signals the end of the frame.