Stomp.Frames

Safe Haskell	Safe
Language	Haskell2010

The Frames module provides abstract data types for representing STOMP Frames, and convenience functions for working with those data types.

Documentation

data **Header**

A Header contains a name and a value for a STOMP header

Constructors

Header HeaderName HeaderValue

Instances

Show Header

data **Headers**

A Headers is a recursive, list-like data structure representing the set of Headers for a given STOMP frame

Constructors

Some Header Headers

EndOfHeaders

Instances

Show Headers

data **Body**

A Body represents the body of a STOMP frame. It is either empty or consists of a single ByteString.

Constructors

EmptyBody

Body ByteString

Instances

Show Body

data **Command**

A Command represents the command portion of a STOMP Frame.

Constructors

SEND

SUBSCRIBE

UNSUBSCRIBE

BEGIN

COMMIT
ABORT
ACK
NACK
DISCONNECT
CONNECT
STOMP
CONNECTED
MESSAGE
RECEIPT
ERROR

Instances

Show Command

`data` **Frame**

An abstract data type representing a STOMP Frame. It consists of a Command, Headers, and Body.

Constructors

Frame Command Headers Body

Instances

Show Frame

`data` **AckType**

Data type representing the possible acknowledgement types for a subscription to a STOMP broker.

Constructors

Auto
Client
ClientIndividual

Instances

Show AckType

---

abort :: String -> Frame

Generate an ABORT Frame given a transaction identifier as a String.

---

ack :: String -> Frame

Generate an ACK Frame given a message identifier as a String.

---

ackHeader :: String -> Header

Given a String, generate an ack header.

addFrameHeaderFront :: Header -> Frame -> Frame

Given a Frame, add a Header to the front of its Headers.

addFrameHeaderEnd :: Header -> Frame -> Frame

Given a Frame, add a Header to the end of its Headers.

addHeaderEnd :: Header -> Headers -> Headers

Add a Header to the end of the Headers

addHeaderFront :: Header -> Headers -> Headers

Add a Header to the front of the Headers

addReceiptHeader :: String -> Frame -> Frame

Add a receipt Header to the Frame.

begin :: String -> Frame

Generate a BEGIN Frame given a transaction identifier as a String.

commit :: String -> Frame

Generate a COMMIT Frame given a transaction identifier as a String.

connect :: String -> Int -> Int -> Frame

Generate a CONNECT Frame given a host identifier as a String.

connected :: String -> Int -> Int -> Frame

Generate a CONNECTED Frame given a version identifier as a String.

disconnect :: String -> Frame

Generate a DISCONNECT Frame given a receipt identifier as a String.

errorFrame :: String -> Frame

Generate an ERROR Frame given an error message as a String.

getAckType :: Frame -> Maybe AckType

Given a Frame, get the AckType if it is present

getBody :: Frame -> Body

Convenience function for retrieving the Body portion of a Frame.

getCommand :: Frame -> Command

Convenience function for retrieiving the Command portion of a Frame.

getContentLength :: Headers -> Maybe Int

Given Headers, get the value of the content-length Header if it is present.

getDestination :: Frame -> Maybe String

Given a Frame, get the value of the destination header if it is present.

getHeaders :: Frame -> Headers

Convenience function for retrieving the Headers portion of a Frame.

getHeartbeat :: Frame -> (Int, Int)

Given a Frame, get the (x, y) values in its 'heart-beat' header. If the header is not present, or the values are malformed, returns (0, 0).

getId :: Frame -> Maybe String

Given a Frame, get the value of the id header if it is present.

getReceipt :: Frame -> Maybe String

Given a Frame, get the value of the receipt header if it is present.

getReceiptId :: Frame -> Maybe String

Given a Frame, get the value of the receipt-id header if it is present.

getSupportedVersions :: Frame -> Maybe [String]

Given a Frame, get a list of supported STOMP versions, provided that the accept-version Header is present and well-formed.

getTransaction :: Frame -> Maybe String

Given a Frame, get the value of the transaction header if it is present.

getValueForHeader :: String -> Headers -> Maybe String

Given a header name and a Frame, get the value for that header if it is present.

idHeader :: String -> Header

Given an ID as a String, create an id Header.

makeHeaders :: [Header] -> Headers

Given a list of Header datatypes, return a Headers datatype.

---

messageIdHeader :: `String -> Header`

Given a message identifier as a String, generate a message-id Header.

---

nack :: `String -> Frame`

Generate a NACK Frame given a message identifier as a String.

---

receipt :: `String -> Frame`

Generate a RECEIPT Frame given a receipt identifier as a String.

---

sendText :: `String -> String -> Frame`

Generate a plain text SEND Frame given a message as a String and a destination as a String.

---

subscribe :: `String -> String -> AckType -> Frame`

Generate a SUBSCRIBE Frame given a subscription identifer and destination as Strings, an

---

subscriptionHeader :: `String -> Header`

Given a subscription identifier as a String, generate a subscription Header.

---

txHeader :: `String -> Header`

Given a transaction identifier as a String, generate a transaction Header.

---

unsubscribe :: `String -> Frame`

Generate an UNSUBSCRIBE Frame given a subscription identifier.

_getDestination :: `Frame -> String`

Given a Frame, get the value of the destination header. If it is not present, throw an error.

_getAck :: `Frame -> String`

Given a Frame, get the value of the ack header; throws an error if it is not present.

_getId :: `Frame -> String`

Given a Frame, get the value of the id header. If it is not present, throws an error.

Stomp.Frames.IO

Safe Haskell	None
Language	Haskell2010

The IO module of the Frames package encapsulates error-handling IO operations on Handles that are expected to be receiving STOMP frames.

Documentation

`data` **FrameHandler**

A FrameHandler encapsulates the work of sending and receiving frames on a Handle. In most cases, this will be a Handle to a TCP socket connection in a STOMP client or broker.

`data` **FrameEvt**

A FrameEvt is a type of event that can be received from a FrameHandler (see the get function).

Constructors

NewFrame Frame
ParseError String
Heartbeat
GotEof
TimedOut

Instances

Show FrameEvt

initFrameHandler :: Handle -> IO FrameHandler

Given a resource Handle to which STOMP frames will be read from and written to, initializes a FrameHandler and returns it to the caller.

put :: FrameHandler -> Frame -> IO ()

Puts the given Frame into the given FrameHandler in an IO context. This function will block until the Frame has been processed.

putEvt :: Frame -> FrameHandler -> Evt ()

Puts the given Frame into the FrameHandler in an Evt context.

get :: FrameHandler -> IO FrameEvt

Get the next FrameEvt from the given FrameHandler and return it in an IO context. This function will block until a FrameEvt is available.

getEvt :: FrameHandler -> Evt FrameEvt

Get the next FrameEvt from the given FrameHandler and return it in an Evt context.

getEvtWithTimeOut :: FrameHandler -> Int -> Evt FrameEvt

Get the next FrameEvt from the given FrameHandler and return it an Evt context. If the given timeout (in microseconds) is exceeded prior to receiving activity on the channel, this will return TimedOut.

close :: FrameHandler -> IO ()

Kills all threads associated with the FrameHandler.

frameToBytes :: Frame -> ByteString

Convert a Frame to a STOMP protocol adherent ByteString suitable for transmission over a handle.

updateHeartbeat :: FrameHandler -> Int -> IO ()

Update the rate at which this FrameHandler sends heart-beats. A rate of 0 or less means that no heart-beats will be transmitted. Otherwise, we will send one heart-beat every n microseconds.

Stomp.Frames.Router

Safe Haskell	None
Language	Haskell2010

The Router module implements an asynchronous notification system for events received on a FrameHandle. Callers can request various types of events and receive those events on a dedicated communications channel.

Documentation

initFrameRouter :: FrameHandler -> IO RequestHandler

Given a FrameHandler on which FramesEvts are being received, initalize a FrameRouter and return a RequestHandler for that FrameRouter to the caller.

requestResponseEvents :: RequestHandler -> IO (SChan FrameEvt)

Given a RequestHandler, request a new SChan on which to receive response events. Response events are defined as any Frame received other than a MESSAGE Frame.

requestSubscriptionEvents :: RequestHandler -> String -> IO (SChan FrameEvt)

Given a RequestHandler, request a new SChan on which to receive subscription events for a given subscription identifier.

requestHeartbeatEvents :: RequestHandler -> IO (SChan FrameEvt)

Given a RequestHandler, request a new SChan on which to receive heart-beats

requestErrorEvents :: RequestHandler -> IO (SChan FrameEvt)

Given a RequestHandler, request a new SChan on which to receive ERROR frames

data **RequestHandler**

The RequestHandler is used to request various types of notifications from the FrameRouter.

Stomp.Subscriptions

Safe Haskell	None
Language	Haskell2010

The Subscriptions module deals with managing subscriptions on the STOMP broker.

Documentation

```
type ClientId = Integer
```

A unique client identifier

```
type Destination = String
```

A unique destination identifier

```
data SubscriptionManager
```

The SubscriptionManager allows the server to add and remove new subscriptions, send messages to destinations, and send ACK/NACK responses.

```
clientDisconnected :: SubscriptionManager -> ClientId -> IO ()
```

Report a client disconnect

```
initManager :: IO SubscriptionManager
```

Initialize a SubscriptionManager and return it in an IO context.

```
unsubscribe :: SubscriptionManager -> ClientId -> SubscriptionId -> IO ()
```

Unsubscribe from a destination.

```
subscribe :: SubscriptionManager -> Destination -> ClientId -> SubscriptionId ->
AckType -> FrameHandler -> IO ()
```

Subscribe to a destination; if the destination does not already exist it will be created.

```
sendAckResponse :: SubscriptionManager -> ClientId -> Frame -> IO ()
```

Send an ack response in an IO context

```
ackResponseEvt :: SubscriptionManager -> ClientId -> Frame -> Evt ()
```

Send an ack response in an Evt context

```
sendMessage :: SubscriptionManager -> Destination -> Frame -> IO ()
```

Send a Frame to a Destination in an IO context.

```
sendMessageEvt :: SubscriptionManager -> Destination -> Frame -> Evt ()
```

Send a Frame to a Destination in an Evt context.

Stomp.Transaction

Safe Haskell	None
Language	Haskell2010

The Stomp.Transaction module implements a ClientTransactionManager that can be used, in conjunction with a SubscriptionManager, to handle STOMP transactiosn for a single client.

Documentation

`data` **`ClientTransactionManager`**

Encapuslates the communications channels for a client transaction manager.

`data` **`UpdateResponse`**

An update will either be successful or generate an error message.

Constructors

`Success`

`Error` `String`

`type` **`TransactionId`** `=` `String`

A TransactionId is a unique (per client) transaction identifier

`initTransactionManager` `::` `SubscriptionManager -> IO ClientTransactionManager`

Initialize a ClientTransactionManager and return it in an IO context.

`begin` `::` `TransactionId -> ClientTransactionManager -> IO UpdateResponse`

Begin a Transaction with the given TransactionId

`commit` `::` `TransactionId -> ClientTransactionManager -> IO UpdateResponse`

Commit the Transaction with the given TransactionId

`abort` `::` `TransactionId -> ClientTransactionManager -> IO UpdateResponse`

Abort the Transaction with the given TransactionId

`ackResponse` `::` `TransactionId -> ClientId -> Frame -> ClientTransactionManager -> IO UpdateResponse`

Add an AckResponse to the Transaction with the given TransactionId

`send` `::` `TransactionId -> Destination -> Frame -> ClientTransactionManager -> IO UpdateResponse`

Add a SEND frame to the Transaction with the given TransactionId

disconnect :: ClientTransactionManager -> IO UpdateResponse

Send a "disconnect" notice; all pending transactions will be aborted. The ClientTransactionManager should not be used after calling this function.

Stomp.TLogger

Safe Haskell	None
Language	Haskell2010

The TLogger module implements a transactional logger. It is a convenient way to write output to a handle in a multi-threaded context in which it is necessary to provide sequential locking access to the handle. Examples of situations in which this might be used is in a logfile or a command-prompt in a multi-threaded application.

Documentation

initLogger :: Handle -> IO Logger

Initialize and return a logger for the given Handle.

dateTimeLogger :: Handle -> IO Logger

Initialize and return a logger that automatically timestamps all of its messages with the current UTC time.

log :: Logger -> String -> IO ()

Send a line of output to the Logger's Handle.

prompt :: Logger -> String -> IO ()

Send output to the Logger's handle, but do not append a newline.

addTransform :: (IO String -> IO String) -> Logger -> Logger

Get a new Logger for the same Handle as the original Logger, but with some additional String transformation applied.

data **Logger**

A Logger can be used to send output to its Handle. It is guaranteed to be thread-safe.

Stomp.Util

Safe Haskell	Safe
Language	Haskell2010

The Util module exports convenience functions from the Stomp library.

Documentation

tokenize :: String -> String -> [String]

Given a delimiter and a String, return a list of the tokens in the String given by splitting on that delimiter.