Modules

- HsMemcached
  - ASCII
    - HsMemcached.ASCII.Common
    - HsMemcached.ASCII.Parser
    - HsMemcached.ASCII.Parser2
    - HsMemcached.ASCII.Server
  - HsMemcached.AdvisedHandler
  - Backend
    - HsMemcached.Backend.HashtableBasic
    - HsMemcached.Backend.HashtableCuckoo
    - HsMemcached.Backend.HashtableLinear
    - HsMemcached.Backend.STMMap
    - HsMemcached.Backend.STMMapv2
    - HsMemcached.Backend.TChan
  - Binary
    - HsMemcached.Binary.Common
    - HsMemcached.Binary.Parser
    - HsMemcached.Binary.Server
  - HsMemcached.Common
  - HsMemcached.Core
  - HsMemcached.EffectiveAdvice
  - HsMemcached.Log
  - Network
    - HsMemcached.Network.GHC
  - HsMemcached.ServerCommon
- Main

HsMemcached.ASCII.Common

Safe Haskell None

Documentation

mcEOL :: ByteString

mcNL :: ByteString

mcWS :: ByteString

mcCas :: ByteString

mcSet :: ByteString

mcAdd :: ByteString

mcReplace :: ByteString

mcAppend :: ByteString

mcPrepend :: ByteString

mcGet :: ByteString

mcGets :: ByteString

mcDelete :: ByteString

mcIncr :: ByteString

mcDecr :: ByteString

mcTouch :: ByteString

mcFlushAll :: ByteString

mcVersion :: ByteString

mcQuit :: ByteString

mcStats :: ByteString

mcSlabs :: ByteString

mcSlabsReassign :: ByteString

mcSlacbsAutomove :: ByteString

mcNoReply :: ByteString

mcStored :: ByteString

mcNotStored :: ByteString

mcTouched :: ByteString

mcExists :: ByteString

mcNotFound :: ByteString

mcDeleted :: ByteString

mcValue :: ByteString

mcEnd :: ByteString

mcVersionReply :: ByteString

mcOK :: ByteString

mcError :: ByteString

mcClientError :: ByteString

mcServerError :: ByteString

showNoReply :: NoReply -> ByteString

cmdToBS :: McCmd -> ByteString

keysToBS :: [Key] -> ByteString

isValidKeyW8 :: Word8 -> Bool

replyToBS :: McReply -> ByteString

w2c :: Word8 -> Char

w8lToStr :: [Word8] -> String

c2w :: Char -> Word8

isSpaceW8 :: Word8 -> Bool

Return True for any space character and the control characters t, n, r, f, v

isCarriageReturnW8 :: Word8 -> Bool

HsMemcached.ASCII.Parser

Safe Haskell None

Documentation

```
parseASCII :: Parser McCmd
```

HsMemcached.ASCII.Parser2

Safe Haskell None

Documentation

data **ParseState**

Constructors

ParseState

bytestring :: ByteString

offset :: Int

Instances

Show ParseState

newtype **Parse** a

Constructors

Parse

runParse :: ParseState -> Either String (a, ParseState)

Instances

Monad Parse

Functor Parse

getState :: Parse ParseState

putState :: ParseState -> Parse ()

assert :: Bool -> String -> Parse ()

parseByte :: Parse Word8

Return exactly one byte

parseBytes :: Int -> Parse ByteString

Returns the next n bytes

consumePattern :: ByteString -> Parse ()

consumeWS :: Parse ()

Consumes excactly one whitespace

consumeNewLine :: Parse ()

Consumes a new line pattern. Which is rn

parseChar :: Parse Char

Returns the next byte as Char

peekByte :: Parse (Maybe Word8)

Just peeks next byte WITHOUT modifing the state

peekByteSafe :: Parse Word8

peekCharSafe :: Parse Char

parseWhile :: (Word8 -> Bool) -> Parse [Word8]

Parses while the given condition is true.

parseValue :: ValueLength -> Parse Value

Parses the value of the key The value's length must be given

parseTo :: (String -> Maybe a) -> Parse a

Parsing helper function. Takes a function that tries to parse a string where this string is the current stream position until a newline or space is encountered

parseCas :: Parse Cas

parseCounter :: Parse Counter

parseFlags :: Parse Flags

parseExptime :: Parse Exptime

parseValueLength :: Parse ValueLength

parseFlushSeconds :: Parse FlushSeconds

```
parseWord32 :: Parse Word32
```

```
parseNoReply :: Parse Bool
```

```
parseKey :: Parse Key
```

```
splitKeys :: [Word8] -> [Key]
```

```
parseKeys :: Parse [Key]
```

```
parseInsertGroup :: Parse (Key, Value, Flags, Exptime, NoReply)
```

```
parseCasCmd :: Parse McCmd
```

```
parseSetCmd :: Parse McCmd
```

```
parseAddCmd :: Parse McCmd
```

```
parseReplaceCmd :: Parse McCmd
```

```
parseAppendCmd :: Parse McCmd
```

```
parsePrependCmd :: Parse McCmd
```

```
parseGetCmd :: Parse McCmd
```

```
parseGetsCmd :: Parse McCmd
```

```
parseDeleteCmd :: Parse McCmd
```

```
parseCounterGroup :: Parse (Key, Counter, NoReply)
```

```
parseIncrCmd :: Parse McCmd
```

```
parseDecrCmd :: Parse McCmd
```

```
parseTouchCmd :: Parse McCmd
```

```
parseFlushCmd :: Parse McCmd
```

parseQuitCmd :: Parse McCmd

parseVersionCmd :: Parse McCmd

parseVerbosityCmd :: Parse McCmd

parseStatsCmd :: Parse McCmd

parseAsciiCmd :: Parse McCmd

parse :: ByteString -> Either String (McCmd, ByteString)

runParse main entry point

parseFile :: String -> IO ()

HsMemcached.ASCII.Server

Safe Haskell None

Documentation

```
tcpASCIIHandler :: BE b => ByteString -> Socket -> M b ()
```

HsMemcached.AdvisedHandler

Safe Haskell None

Documentation

type **XpendArguments** = (Key, Value, NoReply, Maybe Cas)

type **StandardArguments** = (Key, Value, NoReply, Maybe Flags, Maybe Exptime, Maybe Cas)

type **StandardSetArguments** = (Key, Value, NoReply, Flags, Exptime, Maybe Cas)

type **ModValueFunc** = ByteString -> ByteString -> ByteString

baseHandler :: BE b => Advice (McCmd -> M b McReply)

xcrHandler :: BE b => Advice (McCmd -> M b McReply)

xpendHandler :: BE b => Advice (McCmd -> M b McReply)

addHandler :: BE b => Advice (McCmd -> M b McReply)

replaceHandler :: BE b => Advice (McCmd -> M b McReply)

touchHandler :: BE b => Advice (McCmd -> M b McReply)

flushHandler :: BE b => Advice (McCmd -> M b McReply)

statsHandler :: BE b => Advice (McCmd -> M b McReply)

validKeyHandler :: BE b => Advice (McCmd -> M b McReply)

maxItemSizeHandler :: BE b => Advice (McCmd -> M b McReply)

handleStats :: BE b => M b McReply

handleTouch :: BE b => Key -> Exptime -> NoReply -> M b McReply

handleSetCond :: BE b => StandardSetArguments -> Bool -> M b McReply

Set a key under a condition either that key has to exist or it must not exists to set

handleSet :: BE b => StandardSetArguments -> M b McReply

handleGet :: BE b => [Key] -> Bool -> M b McReply

handleDel :: BE b => Key -> Bool -> M b McReply

handleVersion :: BE b => M b McReply

handleVerbosity :: BE b => Word32 -> NoReply -> M b McReply

handleXpend :: BE b => XpendArguments -> ModValueFunc -> M b McReply

handleXcr :: BE b => (Key, NoReply) -> Maybe (Exptime, Counter) -> (Counter -> Counter) -> M b McReply

extendedCounterFunction :: Exptime -> Counter -> Key -> (Counter -> Counter) -> ModItemFunc

counterFunction :: (Counter -> Counter) -> ModItemFunc

(->>) :: BE b => NoReply -> McReply -> M b McReply

Safe Haskell None

HsMemcached.Backend.HashtableBasic

Documentation

```
type HashTable = HashTable
```

```
data Backend
```

Constructors

Backend

```
hashtable :: MVar (HashTable RealWorld Key Item)
```

Instances

BE Backend

Safe Haskell None

HsMemcached.Backend.HashtableCuckoo

Documentation

```
type HashTable = HashTable
```

```
data Backend
```

Constructors

Backend

```
hashtable :: MVar (HashTable RealWorld Key Item)
```

Instances

BE Backend

Safe Haskell None

HsMemcached.Backend.HashtableLinear

Documentation

```
type HashTable = HashTable
```

```
data Backend
```

Constructors

Backend

```
hashtable :: MVar (HashTable RealWorld Key Item)
```

Instances

BE Backend

Safe Haskell None

HsMemcached.Backend.STMMap

Documentation

data **Backend**

Constructors

Backend

stmMap :: TVar (Map Key Item)

Instances

BE Backend

Safe Haskell None

HsMemcached.Backend.STMMapv2

Documentation

data **Backend**

Constructors

Backend

stmMap :: TVar (Map Key (TVar Item))

backendSize :: Word64

Instances

BE Backend

HsMemcached.Backend.TChan

Safe Haskell None

Documentation

data **Operation**

Constructors

AddItem Key Item (Maybe Cas)

GetItem Key

DelItem Key

FlushEverything

AddItemCond Bool Key Item (Maybe Cas)

ModifyItem Key (Maybe Cas) Flags

data **Response**

Constructors

BResponse Bool

IResponse (Maybe Item)

RResponse McReply

IRResponse (Either McReply Item)

data **Backend**

Constructors

Backend

channel :: TChan (Operation, TMVar Response)

Instances

BE Backend

backendThread :: TChan (Operation, TMVar Response) -> Map Key Item -> IO ()

HsMemcached.Binary.Common

Safe Haskell None

Documentation

type **OpCode** = Word8

type **Opaque** = Word32

runPut' :: Put -> ByteString

strict version of runPut

fromHexToW8 :: String -> Word8

fromHexToW16 :: String -> Word16

magicRequest :: Word8

magicReply :: Word8

opGet :: Word8

opGetQ :: Word8

opGetK :: Word8

opGetKQ :: Word8

opSet :: Word8

opSetQ :: Word8

opAdd :: Word8

opAddQ :: Word8

opReplaceQ :: Word8

opDelete :: Word8

opDeleteQ :: Word8

opIncr :: Word8

opIncrQ :: Word8

opDecr :: Word8

opDecrQ :: Word8

opQuit :: Word8

opQuitQ :: Word8

opFlush :: Word8

opFlushQ :: Word8

opAppend :: Word8

opAppendQ :: Word8

opPrepend :: Word8

opPrependQ :: Word8

opTouch :: Word8

opGAT :: Word8

opGATQ :: Word8

opNoop :: Word8

opVersion :: Word8

opStat :: Word8

opVerbosity :: Word8

sNoError :: Word16

sNotFound :: Word16

sKeyExists :: Word16

sValueTooLarge :: Word16

sInvalidArg :: Word16

sItemNotStored :: Word16

sXcrFailure :: Word16

sUnkownCmd :: Word16

sOOM :: Word16

withKey :: OpCode -> Bool

cmdToBS :: McCmd -> ByteString

shortHeader :: Word8 -> Word8 -> Word16 -> Opaque -> Put

shortCasHeader :: Word8 -> Word8 -> Word16 -> Opaque -> Cas -> Put

header :: Word8 -> Word16 -> Word8 -> Opaque -> Cas -> Int -> Int -> Int -> Put

requestComplete :: Word8 -> Opaque -> Cas -> Key -> Value -> Flags -> Exptime -> Put

requestXcr :: Word8 -> Opaque -> Key -> Counter -> Counter -> Exptime -> Put

requestTouch :: Word8 -> Opaque -> Key -> Exptime -> Put

requestKeyValue :: Word8 -> Opaque -> Key -> Value -> Cas -> Put

requestKey :: Word8 -> Opaque -> Key -> Put

requestFlush :: Word8 -> Opaque -> Maybe FlushSeconds -> Put

requestHeader :: Word8 -> Opaque -> Cas -> Int -> Int -> Int -> Put

shortRequestCasHeader :: Word8 -> Word16 -> Opaque -> Cas -> Put

shortRequestHeader :: Word8 -> Opaque -> Put

replyToBS :: (McReply, OpCode, Opaque) -> ByteString

storedReplyToBs :: Item -> Word8 -> Opaque -> (Put -> t) -> t

noReply :: (McReply, OpCode, Opaque) -> ByteString

Unwraps the actual reply in case it is wrapped within NotStored

extraFlags :: Flags -> ByteString

shortReplyCasHeader :: Word8 -> Word16 -> Opaque -> Cas -> Put

shortReplyHeader :: Word8 -> Word16 -> Opaque -> Put

successResponse :: Word8 -> Opaque -> Put

fullResponse :: Word8 -> ByteString -> ByteString -> ByteString -> Word16 -> Opaque -> Cas -> Put

toWord64be :: ByteString -> ByteString

HsMemcached.Binary.Parser

Safe Haskell None

Documentation

type **ParserResult** = (McCmd, OpCode, Opaque)

parseBinary :: Parser ParserResult

parseHeader :: Parser (Int, Int, Int, Word32, Word64)

parseBody :: Int -> Int -> Int -> Parser (Flags, Exptime, Key, Value)

parseKVBody :: Int -> Int -> Int -> Parser (Key, Value)

parseGet :: Parser ParserResult

parseSet :: Parser ParserResult

parseAdd :: Parser ParserResult

parseReplace :: Parser ParserResult

parseDelete :: Parser ParserResult

parseXcr :: Parser ParserResult

parseQuit :: Parser ParserResult

parseFlush :: Parser ParserResult

parseNoop :: Parser ParserResult

parseVersion :: Parser ParserResult

parseXpend :: Parser ParserResult

```
parseVerbosity :: Parser ParserResult
```

```
parseTouchGAT :: Parser ParserResult
```

```
maybeCas :: Cas -> Maybe Cas
```

HsMemcached.Binary.Server

Safe Haskell None

Documentation

```
tcpBinaryHandler :: BE b => ByteString -> Socket -> M b ()
```

HsMemcached.Common

Safe Haskell None

Documentation

hsMemcachedVersion :: Double

mcVersionCode :: ByteString

mebibyte :: Integer

gibibyte :: Integer

type **ModItemFunc** = (Cas, Maybe Item) -> Either McReply Item

class **MemcachedCommand** t where

Methods

isQuit :: t -> Bool

Instances

MemcachedCommand McCmd

MemcachedCommand (McCmd, OpCode, Opaque)

type **Key** = ByteString

type **Value** = ByteString

type **Flags** = Word32

type **Cas** = Word64

type **Exptime** = Word32

type **ValueLength** = Word32

type **NoReply** = Bool

type **FlushSeconds** = Word32

data **Item**

Constructors

Item

itemKey :: Key

itemValue :: Value

itemFlags :: Flags

itemExptime :: Exptime

itemCas :: Cas

Instances

Eq Item

Read Item

Show Item

data **StatsArgument**

Constructors

StatsSettings

StatsItem

StatsItemSize

StatsSlab

Instances

Eq StatsArgument

Read StatsArgument

Show StatsArgument

data **McCmd**

Constructors

Append

key :: Key

value :: Value

noreply :: NoReply

cas :: Maybe Cas

Prepend

key :: Key

value :: Value

noreply :: NoReply

```
  cas :: Maybe Cas
Set
  key :: Key
  value :: Value
  noreply :: NoReply
  flags :: Flags
  exptime :: Exptime
  cas :: Maybe Cas
Add
  key :: Key
  value :: Value
  noreply :: NoReply
  flags :: Flags
  exptime :: Exptime
  cas :: Maybe Cas
Replace
  key :: Key
  value :: Value
  noreply :: NoReply
  flags :: Flags
  exptime :: Exptime
  cas :: Maybe Cas
Get
  keys :: [Key]
Gets
  keys :: [Key]
GAT
  key :: Key
  exptime :: Exptime
  noreply :: NoReply
Del
  key :: Key
  noreply :: NoReply
Incr
  key :: Key
  count :: Counter
  noreply :: NoReply
  extra :: Maybe (Exptime, Counter)
Decr
```

```
    key :: Key
    count :: Counter
    noreply :: NoReply
    extra :: Maybe (Exptime, Counter)
  Touch
    key :: Key
    exptime :: Exptime
    noreply :: NoReply
  FlushAll
    seconds :: Maybe FlushSeconds
    noreply :: NoReply
  GetVersion
  SetVerbosity
    verbosity :: Word32
    noreply :: NoReply
  Stats
    arguments :: Maybe StatsArgument
  NoOp
  Quit
    noreply :: NoReply
```

Instances

```
Eq McCmd
Read McCmd
Show McCmd
MemcachedCommand McCmd
MemcachedCommand (McCmd, OpCode, Opaque)
```

```
type ReplyValue = (Key, Flags, Value)
```

```
type ReplyValueCas = (Key, Flags, Value, Cas)
```

```
data McReply
```

Constructors

```
  Error
  ClientError
    message :: String
  ServerError
    message :: String
```

```
  Stored
    storedItem :: Item
  NotStored
    reason :: McReply
  Exists
  NotFound
  Touched
  Deleted
  NotNumeric
    message :: String
  Values
    values :: [ReplyValue]
  ValuesCas
    valuesCas :: [ReplyValueCas]
  Value
    val :: Value
    valCas :: Cas
  Version
    version :: ByteString
  OK
  StatsInformation
    info :: ByteString
  NoReply
    actualReply :: McReply
```

Instances

Eq McReply

Show McReply

```
toReplyValue :: Item -> ReplyValue
```

```
toReplyValueCas :: Item -> ReplyValueCas
```

```
isValidKey :: Key -> Bool
```

```
isValidKeyChar :: Char -> Bool
```

```
(<+>) :: ByteString -> ByteString -> ByteString
```

```
valueLength :: Value -> ByteString
```

```
isItemValid :: Word32 -> Word32 -> Exptime -> Cas -> Bool
```

```
maybeReadWord16 :: String -> Maybe Word16
```

```
maybeReadWord32 :: String -> Maybe Word32
```

```
maybeReadWord64 :: String -> Maybe Word64
```

```
maybeReadInt64 :: String -> Maybe Int64
```

```
itemSize :: Item -> Word32
```

```
generateCasPure :: Word32 -> Word32 -> Word64
```

```
toStrict :: ByteString -> ByteString
```

HsMemcached.Core

Safe Haskell None

Documentation

newtype **M** b a

Constructors

M

unM :: ReaderT (MConf b) (StateT (MState b) IO) a

Instances

Typeable2 M
Monad (M b)
Functor (M b)
MonadIO (M b)
MonadState (MState b) (M b)
MonadReader (MConf b) (M b)

runM :: BE b => MConf b -> MState b -> M b a -> IO (a, MState b)

data **MConf** b

Constructors

MConf

port :: Word16
advicedHandler :: McCmd -> M b McReply
maxItemSize :: Word32
maxCacheSize :: Word64
debug :: Bool

Instances

MonadReader (MConf b) (M b)

data **MState** b

Constructors

MState

backend :: b
casCounter :: MVar Word32

verbosity :: MVar Word32

Instances

MonadState (MState b) (M b)

---

class **BE** b where

Methods

---

newBackend :: IO b

---

addItem :: Key -> Item -> Maybe Cas -> M b Bool

---

getItem :: Key -> M b (Maybe Item)

---

delItem :: Key -> M b Bool

---

flushAll :: M b ()

---

addItemCond :: Bool -> Key -> Item -> Maybe Cas -> M b McReply

---

modifyItem :: Key -> Maybe Cas -> ModItemFunc -> M b (Either McReply Item)

Instances

BE Backend

BE Backend

BE Backend

BE Backend

BE Backend

BE Backend

---

getBackend :: BE b => M b b

---

getUnixTime :: BE b => M b Word32

---

getAbsoluteTime :: BE b => Word32 -> M b Word32

---

getAbsoluteExptime :: BE b => Word32 -> M b Word32

---

getCasCounter :: BE b => M b Word32

Returns the next counter value from the global synchronized counter

---

generateCas :: BE b => M b Word64

generateItem :: BE b => Key -> Value -> Flags -> Exptime -> M b Item

isItemOutdated :: BE b => Item -> M b Bool

isItemOk :: BE b => Item -> M b Bool

io :: MonadIO m => forall a. IO a -> m a

liftMaybe :: MonadPlus m => Maybe a -> m a

liftMaybeIntoMaybeT :: Monad m => Maybe a -> MaybeT m a

void :: Functor f => f a -> f ()

Safe Haskell Safe-Inferred

HsMemcached.EffectiveAdvice

Documentation

```
type Advice s = s -> s
```

```
weave :: Advice s -> s
```

```
zero :: Advice s
```

```
(<@>) :: Advice s -> Advice s -> Advice s
```

```
type AAdvice a b c m = (a -> m c, a -> b -> c -> m ())
```

```
augment :: Monad m => AAdvice a b c m -> Advice (a -> m b)
```

```
before :: Monad m => (a -> m ()) -> Advice (a -> m b)
```

```
after :: Monad m => (a -> b -> m ()) -> Advice (a -> m b)
```

```
type RAdvice s = s
```

```
replace :: RAdvice s -> Advice s
```

```
type NAdvice a b c m = (a -> m Bool, AAdvice a b c m, RAdvice (a -> m
b))
```

```
narrow :: Monad m => NAdvice a b c m -> Advice (a -> m b)
```

```
type NIAdvice a b t = forall m. (Monad m, Monad (t m)) => Advice (a -> t
m b)
```

```
niadvice :: (Monad m, MonadTrans t, Monad (t m)) => NIAdvice a b t ->
Advice (a -> t m b)
```

```
nibase :: (Monad m, MonadTrans t, Monad (t m)) => NIBase a b m -> Advice
(a -> t m b)
```

```
orthogonal :: (Monad m, MonadTrans t, Monad (t m)) => (forall m'. (Monad
m', Monad (t m')) => Advice (a -> t m' b)) -> (forall t'. (MonadTrans
t', Monad (t' m)) => Advice (a -> t' m b)) -> Advice (a -> t m b)
```

```
actuation' :: (Monad m, MonadTrans t, Monad (t m)) => Advice (a -> t m
b) -> (forall t'. (MonadTrans t', Monad (t' m)) => Advice (a -> t' m b))
-> Advice (a -> t m b)
```

```
inv_actuation :: (Monad m, MonadTrans t, Monad (t m)) => (forall m'.
(Monad m', Monad (t m')) => Advice (a -> t m' b)) -> Advice (a -> t m b)
-> Advice (a -> t m b)
```

```
interference :: forall s. Advice s -> Advice s -> Advice s
```

```
class Monad m => MonadGet s m | m -> s where
```

Methods

```
get :: m s
```

```
class Monad m => MonadPut s m | m -> s where
```

Methods

```
put :: s -> m ()
```

```
class (MonadGet s m, MonadPut s m) => MonadState s m
```

```
type ROAdvice a b t s = forall m. (MonadGet s m, MonadGet s (t m)) =>
Advice (a -> t m b)
```

```
type WOAdvice a b t s = forall m. (MonadPut s m, MonadPut s (t m)) =>
Advice (a -> t m b)
```

HsMemcached.Log

Safe Haskell None

Documentation

logRequest :: BE b => McCmd -> M b ()

logRequestAdvice :: BE b => Advice (McCmd -> M b McReply)

logReply :: BE b => McCmd -> McReply -> M b ()

logReplyAdvice :: BE b => Advice (McCmd -> M b McReply)

logMessage :: BE b => String -> M b ()

HsMemcached.Network.GHC

Safe Haskell None

Documentation

data **Socket**

Instances

Eq Socket

Show Socket

Typeable Socket

createTCPSocket :: BE b => MConf b -> M b Socket

acceptTCPSocket :: BE b => Socket -> M b Socket

sendTCPSocket :: BE b => Socket -> ByteString -> M b ()

sendTCPSocket' :: BE b => Socket -> ByteString -> M b ()

recvTCPSocket :: BE b => Socket -> M b ByteString

isConnected :: BE b => Socket -> M b Bool

closeTCPSocket :: BE b => Socket -> M b ()

HsMemcached.ServerCommon

Safe Haskell None

Documentation

```
tcpServer :: BE b => ConnectionHandler -> M b ()
```

Main

Safe Haskell None

Documentation

```
main :: IO ()
```