mtl-2.3.1: Monad classes for transformers, using functional dependencies

Quick Jump · Instances · Source · Contents · Index

Copyright	(c) Andy Gill 2001 (c) Oregon Graduate Institute of Science and Technology 2001
License	BSD-style (see the file LICENSE)
Maintainer	libraries@haskell.org
Stability	experimental
Portability	non-portable (multi-param classes, functional dependencies)
Safe Haskell	Safe
Language	Haskell2010

Control.Monad.Writer.Lazy

Contents

Lazy writer monads.

Inspired by the paper *Functional Programming with Overloading and Higher-Order Polymorphism*, Mark P Jones (http://web.cecs.pdx.edu/~mpj/pubs/springschool.html) Advanced School of Functional Programming, 1995.

MonadWriter class

```
class (Monoid w, Monad m) => MonadWriter w m | m -> w where
```
 # Source

Minimal complete definition

```
(writer | tell), listen, pass
```

Methods

```
writer :: (a, w) -> m a
```
 # Source

`writer (a,w)` embeds a simple writer action.

```
tell :: w -> m ()
```
 # Source

`tell w` is an action that produces the output `w`.

listen :: m a -> m (a, w) # Source

listen m is an action that executes the action m and adds its output to the value of the computation.

pass :: m (a, w -> w) -> m a # Source

pass m is an action that executes the action m, which returns a value and a function, and returns the value, applying the function to the output.

▽ Instances

▽ MonadWriter w m => MonadWriter w (MaybeT m) # Source

Defined in Control.Monad.Writer.Class

Methods

writer :: (a, w) -> MaybeT m a # Source

tell :: w -> MaybeT m () # Source

listen :: MaybeT m a -> MaybeT m (a, w) # Source

pass :: MaybeT m (a, w -> w) -> MaybeT m a # Source

▽ Monoid w => MonadWriter w ((,) w) # Source

Since: 2.2.2

Defined in Control.Monad.Writer.Class

Methods

writer :: (a, w) -> (w, a) # Source

tell :: w -> (w, ()) # Source

listen :: (w, a) -> (w, (a, w)) # Source

pass :: (w, (a, w -> w)) -> (w, a) # Source

▽ (Monoid w', MonadWriter w m) => MonadWriter w (AccumT w' m) # Source

There are two valid instances for AccumT. It could either:

1. Lift the operations to the inner MonadWriter
2. Handle the

operations itself, à la a `WriterT`.

This instance chooses (1), reflecting that the intent of `AccumT` as a type is different than that of `WriterT`.

Since: 2.3

Defined in Control.Monad.Writer.Class

Methods

```
writer :: (a, w) -> AccumT w' m a
```
 # Source

```
tell :: w -> AccumT w' m ()
```
 # Source

```
listen :: AccumT w' m a -> AccumT w' m (a, w)
```
 # Source

```
pass :: AccumT w' m (a, w -> w) -> AccumT w' m a
```
 # Source

▿ `MonadWriter w m => MonadWriter w (ExceptT e m)` **# Source**

Since: 2.2

Defined in Control.Monad.Writer.Class

Methods

```
writer :: (a, w) -> ExceptT e m a
```
 # Source

```
tell :: w -> ExceptT e m ()
```
 # Source

```
listen :: ExceptT e m a -> ExceptT e m (a, w)
```
 # Source

```
pass :: ExceptT e m (a, w -> w) -> ExceptT e m a
```
 # Source

▿ `MonadWriter w m => MonadWriter w (IdentityT m)` **# Source**

Defined in Control.Monad.Writer.Class

Methods

```
writer :: (a, w) -> IdentityT m a
```
 # Source

```
tell :: w -> IdentityT m ()
```
 # Source

```
listen :: IdentityT m a -> IdentityT m (a, w)
```
 # Source

```
pass :: IdentityT m (a, w -> w) -> IdentityT m a
```
Source

▿ `MonadWriter w m => MonadWriter w (ReaderT r m)` # Source

Defined in Control.Monad.Writer.Class

Methods

```
writer :: (a, w) -> ReaderT r m a
```
Source

```
tell :: w -> ReaderT r m ()
```
Source

```
listen :: ReaderT r m a -> ReaderT r m (a, w)
```
Source

```
pass :: ReaderT r m (a, w -> w) -> ReaderT r m a
```
Source

▿ `MonadWriter w m => MonadWriter w (StateT s m)` # Source

Defined in Control.Monad.Writer.Class

Methods

```
writer :: (a, w) -> StateT s m a
```
Source

```
tell :: w -> StateT s m ()
```
Source

```
listen :: StateT s m a -> StateT s m (a, w)
```
Source

```
pass :: StateT s m (a, w -> w) -> StateT s m a
```
Source

▿ `MonadWriter w m => MonadWriter w (StateT s m)` # Source

Defined in Control.Monad.Writer.Class

Methods

```
writer :: (a, w) -> StateT s m a
```
Source

```
tell :: w -> StateT s m ()
```
Source

```
listen :: StateT s m a -> StateT s m (a, w)
```
Source

```
pass :: StateT s m (a, w -> w) -> StateT s m a
```
Source

▿ `(Monoid w, Monad m) => MonadWriter w (WriterT w m)` # Source *Since: 2.3*

Defined in Control.Monad.Writer.Class

Methods

```
writer :: (a, w) -> WriterT w m a                          # Source

tell :: w -> WriterT w m ()                                # Source

listen :: WriterT w m a -> WriterT w m (a, w)              # Source

pass :: WriterT w m (a, w -> w) -> WriterT w m a           # Source
```

▿ `(Monoid w, Monad m) => MonadWriter w (WriterT w m)` **Source** **#**

Defined in Control.Monad.Writer.Class

Methods

```
writer :: (a, w) -> WriterT w m a                          # Source

tell :: w -> WriterT w m ()                                # Source

listen :: WriterT w m a -> WriterT w m (a, w)              # Source

pass :: WriterT w m (a, w -> w) -> WriterT w m a           # Source
```

▿ `(Monoid w, Monad m) => MonadWriter w (WriterT w m)` **Source** **#**

Defined in Control.Monad.Writer.Class

Methods

```
writer :: (a, w) -> WriterT w m a                          # Source

tell :: w -> WriterT w m ()                                # Source

listen :: WriterT w m a -> WriterT w m (a, w)              # Source

pass :: WriterT w m (a, w -> w) -> WriterT w m a           # Source
```

▿ `(Monoid w, Monad m) => MonadWriter w (RWST r w s m)` **Source** **#** *Since: 2.3*

Defined in Control.Monad.Writer.Class

Methods

```
writer :: (a, w) -> RWST r w s m a                                  # Source

tell :: w -> RWST r w s m ()                                        # Source

listen :: RWST r w s m a -> RWST r w s m (a, w)                     # Source

pass :: RWST r w s m (a, w -> w) -> RWST r w s m a                  # Source
```

▿ `(Monoid w, Monad m) => MonadWriter w (RWST r w s m)` **Source #**

Defined in Control.Monad.Writer.Class

Methods

```
writer :: (a, w) -> RWST r w s m a                                  # Source

tell :: w -> RWST r w s m ()                                        # Source

listen :: RWST r w s m a -> RWST r w s m (a, w)                     # Source

pass :: RWST r w s m (a, w -> w) -> RWST r w s m a                  # Source
```

▿ `(Monoid w, Monad m) => MonadWriter w (RWST r w s m)` **Source #**

Defined in Control.Monad.Writer.Class

Methods

```
writer :: (a, w) -> RWST r w s m a                                  # Source

tell :: w -> RWST r w s m ()                                        # Source

listen :: RWST r w s m a -> RWST r w s m (a, w)                     # Source

pass :: RWST r w s m (a, w -> w) -> RWST r w s m a                  # Source
```

`listens` `:: MonadWriter w m => (w -> b) -> m a -> m (a, b)` **# Source**

`listens f m` is an action that executes the action `m` and adds the result of applying `f` to the output to the value of the computation.

- `listens f m = liftM (id *** f) (listen m)`

```
censor :: MonadWriter w m => (w -> w) -> m a -> m a
```
Source

`censor f m` is an action that executes the action `m` and applies the function `f` to its output, leaving the return value unchanged.

- `censor f m = pass (liftM (\x -> (x,f)) m)`

The Writer monad

```
type Writer w = WriterT w Identity
```
#

A writer monad parameterized by the type `w` of output to accumulate.

The `return` function produces the output `mempty`, while >>= combines the outputs of the subcomputations using `mappend`.

```
runWriter :: Writer w a -> (a, w)
```
#

Unwrap a writer computation as a (result, output) pair. (The inverse of `writer`.)

```
execWriter :: Writer w a -> w
```
#

Extract the output from a writer computation.

- `execWriter m = snd (runWriter m)`

```
mapWriter :: ((a, w) -> (b, w')) -> Writer w a -> Writer w' b
```
#

Map both the return value and output of a computation using the given function.

- `runWriter (mapWriter f m) = f (runWriter m)`

The WriterT monad transformer

```
newtype WriterT w (m :: Type -> Type) a
```
#

A writer monad parameterized by:

- `w` - the output to accumulate.
- `m` - The inner monad.

The `return` function produces the output `mempty`, while >>= combines the outputs of the subcomputations using `mappend`.

Constructors

WriterT (m (a, w))

▽ **Instances**

▽ (MonadAccum w' m, Monoid w) => MonadAccum w' (WriterT w m) *Since: 2.3*

 # Source

Defined in Control.Monad.Accum

Methods

look :: WriterT w m w' **# Source**

add :: w' -> WriterT w m () **# Source**

accum :: (w' -> (a, w')) -> WriterT w m a **# Source**

▽ (Monoid w, MonadError e m) => MonadError e (WriterT w m)

 # Source

Defined in Control.Monad.Error.Class

Methods

throwError :: e -> WriterT w m a **# Source**

catchError :: WriterT w m a -> (e -> WriterT w m a) -> WriterT w m a **Source**

#

▽ (Monoid w, MonadReader r m) => MonadReader r (WriterT w m)

 # Source

Defined in Control.Monad.Reader.Class

Methods

ask :: WriterT w m r **# Source**

local :: (r -> r) -> WriterT w m a -> WriterT w m a **# Source**

reader :: (r -> a) -> WriterT w m a **# Source**

▽ (MonadSelect w' m, Monoid w) => MonadSelect w' (WriterT w m)

 # Source

'Readerizes' the writer: the 'ranking' function can see the value that's been accumulated (of

type w), but can't add anything to the log. Effectively, can be thought of as 'extending' the 'ranking' by all values of w, but *which* w gets given to any rank calls is predetermined by the 'outer writer' (and cannot change).

Since: 2.3

Defined in Control.Monad.Select

Methods

```
select :: ((a -> w') -> a) -> WriterT w m a
```
 # Source

▿ `(Monoid w, MonadState s m) => MonadState s (WriterT w m)`
 # Source

Defined in Control.Monad.State.Class

Methods

```
get :: WriterT w m s
```
 # Source

```
put :: s -> WriterT w m ()
```
 # Source

```
state :: (s -> (a, s)) -> WriterT w m a
```
 # Source

▿ `(Monoid w, Monad m) => MonadWriter w (WriterT w m)` **# Source**

Defined in Control.Monad.Writer.Class

Methods

```
writer :: (a, w) -> WriterT w m a
```
 # Source

```
tell :: w -> WriterT w m ()
```
 # Source

```
listen :: WriterT w m a -> WriterT w m (a, w)
```
 # Source

```
pass :: WriterT w m (a, w -> w) -> WriterT w m a
```
 # Source

▿ `Monoid w => MonadTrans (WriterT w)`

Defined in Control.Monad.Trans.Writer.Lazy

Methods

lift :: Monad m => m a -> WriterT w m a #

▿ (Monoid w, MonadFail m) => MonadFail (WriterT w m)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

fail :: String -> WriterT w m a #

▿ (Monoid w, MonadFix m) => MonadFix (WriterT w m)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

mfix :: (a -> WriterT w m a) -> WriterT w m a #

▿ (Monoid w, MonadIO m) => MonadIO (WriterT w m)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

liftIO :: IO a -> WriterT w m a #

▿ (Monoid w, MonadZip m) => MonadZip (WriterT w m)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

mzip :: WriterT w m a -> WriterT w m b -> WriterT w m (a, b) #

mzipWith :: (a -> b -> c) -> WriterT w m a -> WriterT w m b -> WriterT w m c #

munzip :: WriterT w m (a, b) -> (WriterT w m a, WriterT w m b) #

▿ Foldable f => Foldable (WriterT w f)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

fold :: Monoid m => WriterT w f m -> m #

foldMap :: Monoid m => (a -> m) -> WriterT w f a -> m #

```
foldMap' :: Monoid m => (a -> m) -> WriterT w f a -> m #
foldr :: (a -> b -> b) -> b -> WriterT w f a -> b #
foldr' :: (a -> b -> b) -> b -> WriterT w f a -> b #
foldl :: (b -> a -> b) -> b -> WriterT w f a -> b #
foldl' :: (b -> a -> b) -> b -> WriterT w f a -> b #
foldr1 :: (a -> a -> a) -> WriterT w f a -> a #
foldl1 :: (a -> a -> a) -> WriterT w f a -> a #
toList :: WriterT w f a -> [a] #
null :: WriterT w f a -> Bool #
length :: WriterT w f a -> Int #
elem :: Eq a => a -> WriterT w f a -> Bool #
maximum :: Ord a => WriterT w f a -> a #
minimum :: Ord a => WriterT w f a -> a #
sum :: Num a => WriterT w f a -> a #
product :: Num a => WriterT w f a -> a #
```

▽ `(Eq w, Eq1 m) => Eq1 (WriterT w m)`

Defined in Control.Monad.Trans.Writer.Lazy

Methods

```
liftEq :: (a -> b -> Bool) -> WriterT w m a -> WriterT w m b -> Bool #
```

▽ `(Ord w, Ord1 m) => Ord1 (WriterT w m)`

Defined in Control.Monad.Trans.Writer.Lazy

Methods

```
liftCompare :: (a -> b -> Ordering) -> WriterT w m a -> WriterT w m b ->
Ordering #
```

▿ (Read w, Read1 m) => Read1 (WriterT w m)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

liftReadsPrec :: (Int -> ReadS a) -> ReadS [a] -> Int -> ReadS (WriterT w m a) #

liftReadList :: (Int -> ReadS a) -> ReadS [a] -> ReadS [WriterT w m a] #

liftReadPrec :: ReadPrec a -> ReadPrec [a] -> ReadPrec (WriterT w m a) #

liftReadListPrec :: ReadPrec a -> ReadPrec [a] -> ReadPrec [WriterT w m a] #

▿ (Show w, Show1 m) => Show1 (WriterT w m)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

liftShowsPrec :: (Int -> a -> ShowS) -> ([a] -> ShowS) -> Int -> WriterT w m a -> ShowS #

liftShowList :: (Int -> a -> ShowS) -> ([a] -> ShowS) -> [WriterT w m a] -> ShowS #

▿ Contravariant m => Contravariant (WriterT w m)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

contramap :: (a' -> a) -> WriterT w m a -> WriterT w m a' #

(>$) :: b -> WriterT w m b -> WriterT w m a #

▿ Traversable f => Traversable (WriterT w f)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

traverse :: Applicative f0 => (a -> f0 b) -> WriterT w f a -> f0 (WriterT w f b) #

sequenceA :: Applicative f0 => WriterT w f (f0 a) -> f0 (WriterT w f a) #

```
mapM :: Monad m => (a -> m b) -> WriterT w f a -> m (WriterT w f b)
sequence :: Monad m => WriterT w f (m a) -> m (WriterT w f a)
```

▿ (Monoid w, Alternative m) => Alternative (WriterT w m)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

```
empty :: WriterT w m a
(<|>) :: WriterT w m a -> WriterT w m a -> WriterT w m a
some :: WriterT w m a -> WriterT w m [a]
many :: WriterT w m a -> WriterT w m [a]
```

▿ (Monoid w, Applicative m) => Applicative (WriterT w m)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

```
pure :: a -> WriterT w m a
(<*>) :: WriterT w m (a -> b) -> WriterT w m a -> WriterT w m b
liftA2 :: (a -> b -> c) -> WriterT w m a -> WriterT w m b -> WriterT w m c
(*>) :: WriterT w m a -> WriterT w m b -> WriterT w m b
(<*) :: WriterT w m a -> WriterT w m b -> WriterT w m a
```

▿ Functor m => Functor (WriterT w m)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

```
fmap :: (a -> b) -> WriterT w m a -> WriterT w m b
(<$) :: a -> WriterT w m b -> WriterT w m a
```

▿ (Monoid w, Monad m) => Monad (WriterT w m)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

```
(>>=) :: WriterT w m a -> (a -> WriterT w m b) -> WriterT w m b
(>>) :: WriterT w m a -> WriterT w m b -> WriterT w m b
return :: a -> WriterT w m a
```

▿ (Monoid w, MonadPlus m) => MonadPlus (WriterT w m)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

```
mzero :: WriterT w m a
mplus :: WriterT w m a -> WriterT w m a -> WriterT w m a
```

▿ (Monoid w, MonadCont m) => MonadCont (WriterT w m) **# Source**

Defined in Control.Monad.Cont.Class

Methods

```
callCC :: ((a -> WriterT w m b) -> WriterT w m a) -> WriterT w m a
```
 # Source

▿ (Read w, Read1 m, Read a) => Read (WriterT w m a)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

```
readsPrec :: Int -> ReadS (WriterT w m a)
readList :: ReadS [WriterT w m a]
readPrec :: ReadPrec (WriterT w m a)
readListPrec :: ReadPrec [WriterT w m a]
```

▿ (Show w, Show1 m, Show a) => Show (WriterT w m a)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

```
showsPrec :: Int -> WriterT w m a -> ShowS
show :: WriterT w m a -> String
showList :: [WriterT w m a] -> ShowS
```

▿ (Eq w, Eq1 m, Eq a) => Eq (WriterT w m a)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

(==) :: WriterT w m a -> WriterT w m a -> Bool #

(/=) :: WriterT w m a -> WriterT w m a -> Bool #

▿ (Ord w, Ord1 m, Ord a) => Ord (WriterT w m a)

Defined in Control.Monad.Trans.Writer.Lazy

Methods

compare :: WriterT w m a -> WriterT w m a -> Ordering #

(<) :: WriterT w m a -> WriterT w m a -> Bool #

(<=) :: WriterT w m a -> WriterT w m a -> Bool #

(>) :: WriterT w m a -> WriterT w m a -> Bool #

(>=) :: WriterT w m a -> WriterT w m a -> Bool #

max :: WriterT w m a -> WriterT w m a -> WriterT w m a #

min :: WriterT w m a -> WriterT w m a -> WriterT w m a #

runWriterT :: WriterT w m a -> m (a, w) #

execWriterT :: Monad m => WriterT w m a -> m w #

Extract the output from a writer computation.

- `execWriterT m = liftM snd (runWriterT m)`

mapWriterT :: (m (a, w) -> n (b, w')) -> WriterT w m a -> WriterT w' n b #

Map both the return value and output of a computation using the given function.

- `runWriterT (mapWriterT f m) = f (runWriterT m)`

```
module Control.Monad.Trans
```