

# Monad Transformer Compatibility

A monad transformer stack will provide access to many *monad type classes*. But you have to be careful, which transformers can be stacked.

[deriving-trans](#) uses 3 *categories* of monad transformers to implement instances for any monad transformer without knowing its exact definition, which solves [mtl](#)'s “n<sup>2</sup> problem”.

## MonadTrans

`lift` a monadic computation `m a` into a monad transformer `t m a`.

## MonadTransControl

`liftWith` runs a transformer's computation `t m a` in the base monad `m a`, but you have to take care of the transformer's monadic state `StT t a` explicitly with `restoreT`.

## MonadTransControlIdentity

`liftWithIdentity` is just like `liftWith`, but without monadic state.

These 3 categories form a hierarchy where `MonadTransControlIdentity` is stronger than `MonadTransControl`, which is stronger than `MonadTrans`.

## Compatibility Matrix

The following table gives you two pieces of information. The “*transformer category*” tells you, which kind of transformer you can stack on top and still keep access to the type class. The “*set by transformers*” column tells you, which transformers implement the type class by themselves.

**NOTE** This table is valid for [deriving-trans 0.8.0.0](#).

	monad type class	transformer category	set by transformers
	<code>MonadBase b</code>	<code>MonadTrans</code>	<i>set by base monad</i>
	<code>MonadBaseControl b</code>	<code>MonadTransControl</code>	<i>set by base monad</i>
	<code>MonadBaseControlIdentity b</code>	<code>MonadTransControlIdentity</code>	<i>set by base monad</i>
<b>base</b>	<code>Alternative</code>	<code>MonadTransControl</code>	<ul style="list-style-type: none"> <li><code>CatchT</code></li> <li><code>ExceptT e</code> with <code>Monoid e</code></li> <li><code>MaybeT</code></li> </ul>
	<code>MonadFail</code>	<code>MonadTrans</code>	<ul style="list-style-type: none"> <li><code>CatchT</code></li> <li><code>MaybeT</code></li> </ul>
	<code>MonadFix</code>	<code>MonadTransControlIdentity</code>	
	<code>MonadIO</code>	<code>MonadTrans</code>	<i>set by base monad</i>
	<code>MonadPlus</code>	<code>MonadTransControl</code>	
	<code>MonadZip</code>	<code>MonadTransControlIdentity</code>	

	monad type class	transformer category	set by transformers
<b>exceptions</b>	MonadThrow	MonadTrans	• CatchT
	MonadCatch	MonadTransControl	• CatchT
<b>mtl</b>	MonadAccum w	MonadTrans	• AccumT w
	MonadCont	MonadTransControl	• ContT r
	MonadError e	MonadTransControl	• ExceptT e
	MonadReader r	MonadTransControl	• ReaderT r • RWST r w s with Monoid w ◦ Lazy ◦ Strict ◦ CPS
	MonadRWS r w s	MonadTransControl	• RWST r w s with Monoid w ◦ Lazy ◦ Strict ◦ CPS
	MonadSelect r	MonadTrans	• SelectT r
	MonadState s	MonadTrans	• RWST r w s with Monoid w ◦ Lazy ◦ Strict ◦ CPS • StateT s ◦ Lazy ◦ Strict
	MonadWriter w	MonadTransControl	• RWST r w s with Monoid w ◦ Lazy ◦ Strict ◦ CPS • WriterT w with Monoid w ◦ Lazy ◦ Strict ◦ CPS
<b>primitive</b>	PrimMonad	MonadTrans	<i>set by base monad</i>
<b>random</b>	StatefulGen g	MonadTrans	
	FrozenGen f	MonadTrans	
	RandomGenM g r	MonadTrans	

	monad type class	transformer category	set by transformers
<b>resourceT</b>	MonadResource	MonadTrans	• ResourceT
<b>unliftIO</b>	MonadUnLiftIO	MonadTransControlIdentity	set by base monad

And now let me quickly explain how to make use of this table with an example.

*Example 1. Understanding MonadReader r as an example.*

In the table you will find a row on `MonadReader`, which will give you the following information.

1. A `MonadReader r m` instance can also imply `MonadReader r (t m)` when `t` satisfies `MonadTransControl`.
2. `ReaderT r` or `RWST r w s` can be used to implement an instance by themselves.

Here are some examples of transformer stacks for any `Monad m` using `(.>)` from `deriving-trans`.

`(TransparentT .> ReaderT r .> ExceptT e) m`

- ☑ will have a `MonadReader r` instance, because `ExceptT e` satisfies `MonadTransControl`.

`(TransparentT .> ReaderT r .> ContT r) m`

- ☐ won't have a `MonadReader r` instance, because `ContT r` doesn't satisfy `MonadTransControl`.

`(TransparentT .> ReaderT r1 .> ReaderT r2) m`

- ☑ will have a `MonadReader r2` instance.
- ☑ will also have a `MonadReader r1` instance, unless `r1 ~ r2`.

`(TransparentT .> ExceptT e) m`

- will have a `MonadReader r` instance, whenever `m` satisfies `MonadReader r`.

Feel free to use this table as a cheat sheet or learning material. There are some intricacies though, which are hard to express in this format.

#### TIP

Some methods like `ask` from `MonadReader` don't require the “*transformer category*” from the table. In this case you might want to use your own type class, which you can call `MonadAsk` for example. This might actually be default in the future anyways though.

Some monad type classes are “*set by base monad*”. I chose this for a few type classes, which only make sense when the instances come from the base monad `m`.

*Compare these instances to understand the difference.*

#### NOTE

```
-- recursive instance
(MonadExample (t2 m) {-, ... -}) => MonadExample (ComposeT t1 t2 m)

-- base monad instance
```

```
(MonadExample m {-, ... -}) => MonadExample (ComposeT t1 t2 m)
```

## Outlook

Currently I don't have proofs for the compatibility matrix, so it's possible, that some instances are not lawful and will change in the future. I am working on supporting [logict](#), but in this case [I am not yet sure](#), whether we are allowed to lift it through any `t` satisfying `MonadTransControl`.