NAME

"IO::Async::Channel" − pass values into or out from an IO::Async::Routine

DESCRIPTION

A `IO::Async::Channel` object allows Perl values to be passed into or out of an IO::Async::Routine. It is intended to be used primarily with a Routine object rather than independently. For more detail and examples on how to use this object see also the documentation for IO::Async::Routine.

A Channel object is shared between the main process of the program and the process running within the Routine. In the main process it will be used in asynchronous mode, and in the Routine process it will be used in synchronous mode. In asynchronous mode all methods return immediately and use IO::Async−style futures or callback functions. In synchronous within the Routine process the methods block until they are ready and may be used for flow-control within the routine. Alternatively, a Channel may be shared between two different Routine objects, and not used directly by the controlling program.

The channel itself represents a FIFO of Perl reference values. New values may be put into the channel by the `send` method in either mode. Values may be retrieved from it by the `recv` method. Values inserted into the Channel are snapshot by the `send` method. Any changes to referred variables will not be observed by the other end of the Channel after the `send` method returns.

PARAMETERS

The following named parameters may be passed to `new` or `configure`:

codec => STR

Gives the name of the encoding method used to represent values over the channel.

This can be set to `Storable` to use the core Storable module. As this only supports references, to pass a single scalar value, `send` a SCALAR reference to it, and dereference the result of `recv`.

If the Sereal::Encoder and Sereal::Decoder modules are installed, this can be set to `Sereal` instead, and will use those to perform the encoding and decoding. This optional dependency may give higher performance than using `Storable`. If these modules are available, then this option is picked by default.

CONSTRUCTOR

new

```
$channel = IO::Async::Channel->new
```

Returns a new `IO::Async::Channel` object. This object reference itself should be shared by both sides of a `fork()`ed process. After `fork()` the two `setup_*` methods may be used to configure the object for operation on either end.

While this object does in fact inherit from IO::Async::Notifier, it should not be added to a Loop object directly; event management will be handled by its containing IO::Async::Routine object.

METHODS

The following methods documented with a trailing call to `->get` return Future instances.

configure

```
$channel->configure( %params )
```

Similar to the standard `configure` method on IO::Async::Notifier, this is used to change details of the Channel's operation.

on_recv => CODE

May only be set on an async mode channel. If present, will be invoked whenever a new value is received, rather than using the `recv` method.

```
$on_recv->( $channel, $data )
```

on_eof => CODE

May only be set on an async mode channel. If present, will be invoked when the channel gets closed by the peer.

```
$on_eof->( $channel )
```

send

```
$channel->send( $data )
```

Pushes the data stored in the given Perl reference into the FIFO of the Channel, where it can be received by the other end. When called on a synchronous mode Channel this method may block if a `write()` call on the underlying filehandle blocks. When called on an asynchronous mode channel this method will not block.

send_encoded

```
$channel->send_encoded( $record )
```

A variant of the `send` method; this method pushes the byte record given. This should be the result of a call to `encode`.

encode

```
$record = $channel->encode( $data )
```

Takes a Perl reference and returns a serialised string that can be passed to `send_encoded`. The following two forms are equivalent

```
$channel->send( $data )
$channel->send_encoded( $channel->encode( $data ) )
```

This is provided for the use-case where data needs to be serialised into a fixed string to ''snapshot it'' but not sent yet; the returned string can be saved and sent at a later time.

```
$record = IO::Async::Channel->encode( $data )
```

This can also be used as a class method, in case it is inconvenient to operate on a particular object instance, or when one does not exist yet. In this case it will encode using whatever is the default codec for `IO::Async::Channel`.

send_frozen

```
$channel->send_frozen( $record )
```

Legacy name for `send_encoded`. This is no longer preferred as it expects the data to be encoded using `Storable`, which prevents (or at least makes more awkward) the use of other codecs on a channel by default. This method should not be used in new code and may be removed in a later version.

recv

```
$data = $channel->recv
```

When called on a synchronous mode Channel this method will block until a Perl reference value is available from the other end and then return it. If the Channel is closed this method will return `undef`. Since only references may be passed and all Perl references are true the truth of the result of this method can be used to detect that the channel is still open and has not yet been closed.

```
$data = $channel->recv->get
```

When called on an asynchronous mode Channel this method returns a future which will eventually yield the next Perl reference value that becomes available from the other end. If the Channel is closed, the future will fail with an `eof` failure.

```
$channel->recv( %args )
```

When not returning a future, takes the following named arguments:

on_recv => CODE
: Called when a new Perl reference value is available. Will be passed the Channel object and the reference data.

```
$on_recv->( $channel, $data )
```

on_eof => CODE
: Called if the Channel was closed before a new value was ready. Will be passed the Channel object.

```
$on_eof->( $channel )
```

close

```
$channel->close
```

Closes the channel. Causes a pending `recv` on the other end to return undef or the queued `on_eof` callbacks to be invoked.

AUTHOR

Paul Evans <leonerd@leonerd.org.uk>