

Environmental Sample Processor: Things that go “bump” in the night (understanding error messages :-)

8/18/10 Brent Roman brent@mbari.org



Errors are Unhandled Exceptions

- Great. So, what is an Unhandled Exception?
 - Exceptions are:
 - Unusual conditions that obstruct the normal flow of a program
 - Handled by special code outside the usual flow
 - In modern languages, when a method cannot return a valid value...
 - It “throws” (or “raises”) an exception instead!
 - $10/0 \Rightarrow \text{ZeroDivisionError}$
 - `Math.sqrt(-1)` $\Rightarrow$ `ArgumentError: square root for number < 0`
 - `Math.sqrt()` $\Rightarrow$ `ArgumentError: wrong # of arguments(0 for 1)`
 - Exceptions propagate up the call stack in search of their “handler”
 - Handler code may be very specific or generic
 - If no handler is found, the exception becomes an Error.
 - Actually, the top level of code has a generic Exception handler



Ruby Exception Objects

- Consist of:
 - a text message describing the exception
 - A backtrace to locate the point of failure in nested methods
 - Subclasses may (and do) associate extra information
 - e.g. The servo status associated with `Slide::Error`
- Only subclasses of built-in Exception class may be raised or thrown
 - One cannot throw a Thread or an Integer, etc.
 - Exceptions are otherwise just like any other Ruby object
 - Exceptions are always raised on a specific Thread
- Ruby's “rescue” clause encloses all exception handlers
- If no matching rescue clause found, the thread is quietly terminated
 - It's a good practice to put a generic handler at the highest level
 - Otherwise, you won't know what exception was unhandled!



Deriving Ruby Exception Objects

- Define my own error (exception) class and raise it
 - `class MyErr < StandardError; end`
 - `raise MyErr.new "Your honor, I respectfully object!"`
- Define a `Slide::Error` with associated (servo status) reply and axis:

```
class Slide < LinearAxis
  class Error < LinearAxis::Error
    def initialize text, axis, reply=nil
      @reply = reply
      super text, axis
    end
    attr_reader :reply
  end
end
```

- So, in addition to the base Exception's `backtrace` and `message` `Slide::Error` exceptions support `reply` and `axis` methods



Exception Class Hierarchy

- `ArgumentError.ancestors =>`
`[ArgumentError, StandardError, Exception, Object, ... ]`
- `Slide::Error.ancestors =>`
`[Slide::Error, LinearAxis::Error, Axis::Error, AxisKernel::Error, StandardError, Exception, Object, ... ]`
- `NameError.ancestors =>`
 - `[NameError, ScriptError, Exception, Object, ... ]`
- An example of a class that cannot be raised as an Exception:
 - `Float.ancestors =>`
`[Float, Precision, Numeric, Comparable, Object, ...]`



Backtraces

- Answers the question: *Where was the exception raised?*
- Example:

```
ESPmack:011:0> CC.to :spoon #there is no spoon
```

```
Axis::Error in quick -- Unknown Collection Clamp position: spoon
```

```
ESPmack:012:0> backtrace
```

```
/home/brent/esp2/lib/axis.rb:346:in `baseRaw' #innermost is the "raise" method call
```

```
/home/brent/esp2/lib/axis.rb:164:in `raw'
```

```
/home/brent/esp2/lib/axis.rb:159:in `fetch'
```

```
/home/brent/esp2/lib/axismap.rb:147:in `fetch'
```

```
/home/brent/esp2/lib/axismap.rb:147:in `fetch'
```

```
/home/brent/esp2/lib/axis.rb:152:in `fetch'
```

```
/home/brent/esp2/lib/axis.rb:159:in `raw'
```

```
/home/brent/esp2/lib/axis.rb:382:in `raw'
```

```
/home/brent/esp2/lib/slide.rb:250:in `seek'
```

```
/home/brent/esp2/lib/slide.rb:299:in `moveTo'
```

```
(ESP):11 #this is the eleventh command the user typed
```

```
/usr/local/lib/ruby/1.6/irb/workspace.rb:55:in `irb_binding'
```

```
/usr/local/lib/ruby/1.6/irb/workspace.rb:55
```

```
=> #<Axis::Error: Unknown Collection Clamp position: spoon>
```

Use your text editor to seek to line numbers in each file ref'd

In vi, simply enter a line number at the : prompt

In nedit, type control-L to type line number into a dialog box



Rescuing Ruby Exceptions

- Exception handlers are just blocks of code within a rescue clause

```
def safeDivide num,den
  begin
    num/den
  rescue ZeroDivisionError  #handle div by 0
    puts "Can't divide by zero"
  rescue StandardError => err  #handle most others
    puts err
  end
end
```

- The exception's derived class determines how it is handled
 - Not the message text
 - Text messages are for humans to interpret



ESP Top-Level Exception Handling

- Each ESP thread has an associated queue of unhandled exceptions
 - `Thread[name].exception` => list of most recent errors
 - Only the most recent 10 or so unhandled exceptions are preserved
 - The last is the most recent, the first is the oldest
 - `puts Thread[name].exception` displays all thread's recent errors
- The `backtrace` method with no arguments method displays `Thread.current.exception.last.backtrace`
- `backtrace :name` displays `Thread[:name].exception.last.backtrace`
- `backtrace thread` displays `thread.exception.last.backtrace`
 - e.g. `backtrace MainThread == backtrace :MAIN`
- To save the 2nd to last error (prevent losing it off the queue)
`myErr = thread.exception[-2]`
- Later use: `backtrace myErr` to display exception's backtrace



Ruby Script Errors

- *NameError* ==> specified method or variable is not defined
- *SyntaxError* ==> grammatical error
puts "foo" If 3>2 #If should be lowercase if
- *LoadError* ==> cannot process specified Ruby script file
execute "missingFile"
- **Only** the above errors will *always* require that Ruby script be edited.



Generic Runtime Ruby Errors

- *ArgumentError* ==> number and/or class of objects being passed into a method are incompatible with its definition
- *TypeError* ==> method does not handle the type of object passed in
- *Interrupt* ==> Linux kill signal sent to Ruby process
- *IRB::Abort* ==> Control-C pressed on interactive console
- *RuntimeError* ==> generic error (text message will describe it)
 - raise “something bad's happened” #raises a RuntimeError
- *ZeroDivisionError* ==> e.x. 10/0
- None of the above necessary require script changes to fix
 - Just changing objects may suffice



Internal ESP logging errors

These errors indicate serious bugs or configuration problems

- *Log::CannotDump* ==> attempt to log object containing files or procs
Certain objects cannot be converted to a byte stream
- *Log::Error* ==> other internal error
- *Log::Reader::Error* ==> invalid log file format encountered by dumplog
 - May be caused by read log from different type of ESP
 - i.e. trying to dump a standard core's log from an MFB
 - Or trying to dump MFB equipped ESP's log from one lacking MFB



Scheduler Errors

- *Schedule::Error* ==> time is in the past
trying to schedule an operation (or delay) before current time
- *Schedule::Stop* ==> scheduler has been stopped by error or user
produced as ESP app terminates (no recovery possible)
- *Delay::Error* ==> invalid duration syntax
e.g. delay “1 fortnight”



Thread Errors

- *Thread::Aborted* ==> another thread requested this one be aborted
t.abort #raises Thread::Abort in thread t
- *Thread::ParentDied* ==> the thread that spawned us had a fatal error
Thread::ChildDied ==> a thread this one spawned had a fatal error
Child threads may “orphan” themselves to avoid these errors
- *Thread::Checkpoint::Resume* ==> user should never see this...
exception raised in a moribund thread to resume it



I2C Bus Errors

- *I2C::DuplicateAddress::Error* ==> two dwarves have same address
check dwarves' dip switches very carefully
- *I2C::LAN::NoGateway::Error* ==> network lacks a I2C gateway
configuration error – not generally recoverable
- *I2C::Parser::Error* ==> response sent by dwarf improperly formatted
 - could be caused by very outdated firmware or electrical noise
- *I2C::Request::Timeout* ==> expected response not received in time
usually indicates a motor or sensor is failing – not a network failure
- *I2C::UnexpectedReply* ==> received unexpected dwarf response
May happen when rapidly logging data. Unexpected replies ignored.
- *I2C::NodeOffline* ==> dwarf is not responding to its address
This is a network problem
- *I2C::MsgErr* ==> host is trying to send improperly formatted message
Also (regularly) occurs in simulation on “unmodeled” operations



I2C Message Processing Errors

- *I2C::Solenoid::Error* ==> trying to send invalid solenoid control msg likely a bug in lib/solenoid.rb
- *I2C::Servo::Error* ==> trying to send invalid servo control message likely a bug in lib/slide.rb or very outdated dwarf firmware
- *I2C::Shaft::Error* ==> trying to send invalid rotary valve control msg likely a bug in lib/shaft.rb
- *I2C::SerialPort::Error* ==> trying to send invalid dwarf serial port msg likely a bug in lib/serialport.rb
- *I2C::SerialPort::Configuration::Error* ==> invalid RS232 configuration unsupported port baud rate, parity, etc.
- *I2C::RS232Port::Error* ==> invalid dwarf RS232 serial port config port baud rate, parity, stop bits, etc.
- *I2C::RS232Port::ReadError* ==> dwarf received garbled serial data parity or framing errors usually indicate wrong baud rate or cabling
- *I2C::Thermal::Error* ==> trying to send invalid thermal control message



Contextual Sensor Errors

- *Instrument::ISUS::NoACK* ==> ISUS didn't acknowledge cmd receipt
cabling problem?
- *Instrument::CTDSample::Error* ==> corrupt sample received
likely trying to run a new v2 CTD with old Ruby driver
- *Instrument::CTD::NotWhileLoggingError* ==> can't sample if logging
CTD should never be put into autonomous logging mode
- *Instrument::CTDCore::CalFileMismatch* ==> bad seabird cal file
or a valid cal file given the wrong file name
- *Instrument::CTD::Warning* ==> missing cal file
will still log data, but engineering units are suspect
- *Instrument::ReadTimeout* ==> instrument did not respond in time
check cables, batteries, try CTD or ISUS.term
- *Instrument::NoDataError* ==> no sample available (yet)
- *Instrument::Sample::Error* ==> generic sample error



Axis Errors

- *AxisKernel::Missing* ==> some dwarf did not respond to role call
check I2C and power cabling, verify configure.rb matches hardware
- *AxisKernel::Error* ==> trying to define the same axis object twice
likely a bug problem with your configure.rb file
- *Linear or Rotary Axis::Error* ==> seeking unknown position
could be high level protocol bug or missing info in configure.rb
- *Slide::Error* ==> not yet homed or other servo error
likely missing ESP.ready!, mechanical problem or servo out of tune
- *Scale::Error* ==> invalid Scale object configuration
 - lacking 2 numeric positions or have numeric aliases for same position
- *Clamp::VelocityError* ==> puck detection algorithm failed
e.g. Clamp never reached plateau velocity



Valve Errors

- *Valve::Error* ==> configuration error or selecting undefined position
if during configuration, two positions likely have the same name
- *Valve::Manifold::Error* ==> config error or selecting undefined valve
if during configuration, two valves likely have the same name
- *Solenoid::Error* ==> low-level configuration error
likely a low-level solenoid type is defined ambiguously
i.e. two states sharing the same name



Puck, Clamp & Arm Errors

- *Puck::Error* ==> one of various high-level sanity checks failed
Puck counting logic detected a misplaced puck
Failure to specify type of puck to load or unload
Unspecified Source or Destination tube number
Out of pucks (emptied tube 7)
- *Puck::Warning* ==> specified puck type does not match that in clamp
you explicitly specify unload an :sh2, but you'd loaded an :sh1 puck
Not fatal, just a warning written to the log
- *Clamp::Error* ==> clamp open/closed inappropriately or missing puck
likely someone left a puck in a clamp or forget to put one there
Clamp::VelocityError ==> puck detection algorithm failed
e.g. Clamp never reached plateau velocity
- *Arm::Error* ==> failure in Arm.stretch!
Forearm may be mechanically jammed, unable to reach stops

