It's broken! How to debug a debugger …

Christoph Thiede

The Debugger

Halt: Carpe Squeak!

UndefinedObject(Object)>>halt:
UndefinedObject>>DoIt
Compiler>>evaluateCue:ifFail:
Compiler>>evaluateCue:ifFail:logged:
Compiler>>evaluate:in:to:environment:notifying:ifFail:logged:
[] in SmalltalkEditor(TextEditor)>>evaluateSelectionAndDo:
FullBlockClosure(BlockClosure)>>on:do:
SmalltalkEditor(TextEdito
SmalltalkEditor(TextEdito
SmalltalkEditor(TextEdito
SmalltalkEditor(TextEdito

Proceed | Restart | Into

```
halt: aString
	"This is the typical mess
	debugging. It creates a
	aString, as the label."

	Halt new signal: aString
```

self
all inst vars

<- Select receiver's

thisContext
stackTop
all temp vars
aString
stack2
stack3

<- Select context's f

Show annotation pane in the debugger. — enabled — local

Show stack variables in debugger — enabled — local

When true, append the unnamed stack variables (if any) below the named temps in the debugger's context inspector.

ContextVariablesInspector showStackVariables

more

Cmd dot enabled — enabled — local

suspendedContext

sender

method

TestObjectsAsMethods

Halt:

TestObjectsAsMethods>>testAddNumbers
TestObjectsAsMethods(TestCase)>>performTest
TestObjectsAsMethods(TestCase)>>openDebuggerOnFailingTestMethod
[] in TestObjectsAsMethods(TestCase)>>runCaseAsFailure:
FullBlockClosure(BlockClosure)>>ensure:
TestObjectsAsMethods(TestCase)>>runCaseAsFailure:
TestObjectsAsMethods(TestCase)>>debugAsFailure
Browser(CodeHolder)>>testDebugTest
Browser(StringHolder)>>perform:orSendTo:
[] in MenuItemMorph>>invokeWithEvent:

Proceed | Restart | Into | Over | Through | Full St

```
testAddNumbers
    self assert: (self add: 3 with: 4) = 7.
    self assert: (self perform: #add:with: withArguments: #(3 4)) =
```

PluggableButtonMorphPlus

PluggableButtonMorphPlus
- inspect morph
- inspect owner chain
- explore morph
- viewer for Morph
- browse morph class
- make own subclass
- save morph in file
- call #tempCommand
- define #tempCommand
- control-menu...
- edit balloon help
- browse action code
- debug action invocation

TestObjectsAsMethods

MessageNotUnderstood: ObjectsAsMethodsExample>>numArgs

ObjectsAsMethodsExample(Object)>>doesNotUnderstand: #numArgs
Context>>send:to:with:lookupIn:
Context>>send:super:numArgs:
Context(InstructionStream)>>interpretNextSistaV1InstructionFor:
EncoderForSistaV1 class>>interpretNextInstructionFor:in:
Context(InstructionStream)>>interpretNextInstructionFor:
Context>>step
[] in Process>>stepToHome:

Proceed | Restart | Into | Over | Through | Full Stack | Where | Tally It

```
send: selector to: rcvr with: arguments lookupIn: lookupClass
	"Simulate the action of sending a message with selector and arguments to rcvr. The argument,
lookupClass, is the class in which to lookup the message. This is the receiver's class for normal messages, but
for super messages it will be some specific class related to the source method."

	| meth primIndex val ctxt |
	(meth := lookupClass lookupSelector: selector) ifNil:
		[selector == #doesNotUnderstand: ifTrue:
			[self error: 'Recursive message not understood!' translated].
		^self send: #doesNotUnderstand:
				to: rcvr
				with: {(Message selector: selector arguments: arguments) lookupClass: lookupClass}
				lookupIn: lookupClass].

	meth numArgs = arguments size ifFalse:
		[^ self error: ('Wrong number of arguments in simulated message {1}' translated format:
{selector})].
	(primIndex := meth primitive) > 0 ifTrue:
```

self
all inst vars
sender
pc
stackp
method
closureOrNil

<- Select receiver's field

thisContext
all temp vars
selector
rcvr
arguments
lookupClass
meth

<- Select context's field

Objects as Methods Simulation

```
Context»send: selector to: rcvr with: arguments lookupIn: lookupClass
    "Simulate the action of sending a message with selector and arguments to rcvr. The argument, lookupClass, is the class in which to lookup the message.
This is the receiver's class for normal messages, but for super messages it will be some specific class related to the source method."

    | meth primIndex val ctxt |
    (meth := lookupClass lookupSelector: selector) ifNil:
    [selector == #doesNotUnderstand: ifTrue:
        [self error: 'Recursive message not understood!' translated].
    ^self send: #doesNotUnderstand:
        to: rcvr
        with: {(Message selector: selector arguments: arguments) lookupClass: lookupClass}
        lookupIn: lookupClass].

    meth isCompiledMethod ifFalse:
        ["Object as Methods (OaM) protocol: 'The contract is that, when the VM encounters an ordinary object (rather than a compiled method) in the
method dictionary during lookup, it sends it the special selector #run:with:in: providing the original selector, arguments, and receiver.'. DOI:
10.1145/2991041.2991062."
        ^self send: #run:with:in:
            to: meth
            with: {selector. arguments. rcvr}].

        meth numArgs = arguments size ifFalse:
            [^ self error: ('Wrong number of arguments in simulated message {1}' translated format: {selector})].
        (primIndex := meth primitive) > 0 ifTrue:
            [val := self doPrimitive: primIndex method: meth receiver: rcvr args: arguments.
            (self isPrimFailToken: val) ifFalse:
                [^val]].

    (selector == #doesNotUnderstand: and: [lookupClass == ProtoObject]) ifTrue:
        [^self error: ('Simulated message {1} not understood' translated format: {arguments first selector})].

    ctxt := Context sender: self receiver: rcvr method: meth arguments: arguments.
    (primIndex isInteger and: [primIndex > 0]) ifTrue:
    [ctxt failPrimitiveWith: val].

    ^ctxt
```

Process-Faithful Debugging

```
Processor activeProcess
  evaluate: [
    self error.
    self inform: #foo]
  onBehalfOf: [] newProcess
```

```
Processor activeProcess
  environmentAt: #foo put: 42.
Processor activeProcess
  environmentAt: #foo.  → 42
```

Process≫evaluate: aBlock onBehalfOf: aProcess

```
  "Evaluate aBlock setting effectiveProcess to
aProcess. Used in the execution simulation
machinery to ensure that Processor
activeProcess evaluates correctly when
debugging.“
   | oldEffectiveProcess |
  oldEffectiveProcess := effectiveProcess.
  effectiveProcess := aProcess.
  ^ aBlock ensure: [
    effectiveProcess := oldEffectiveProcess]
```

ProcessorScheduler≫activeProcess

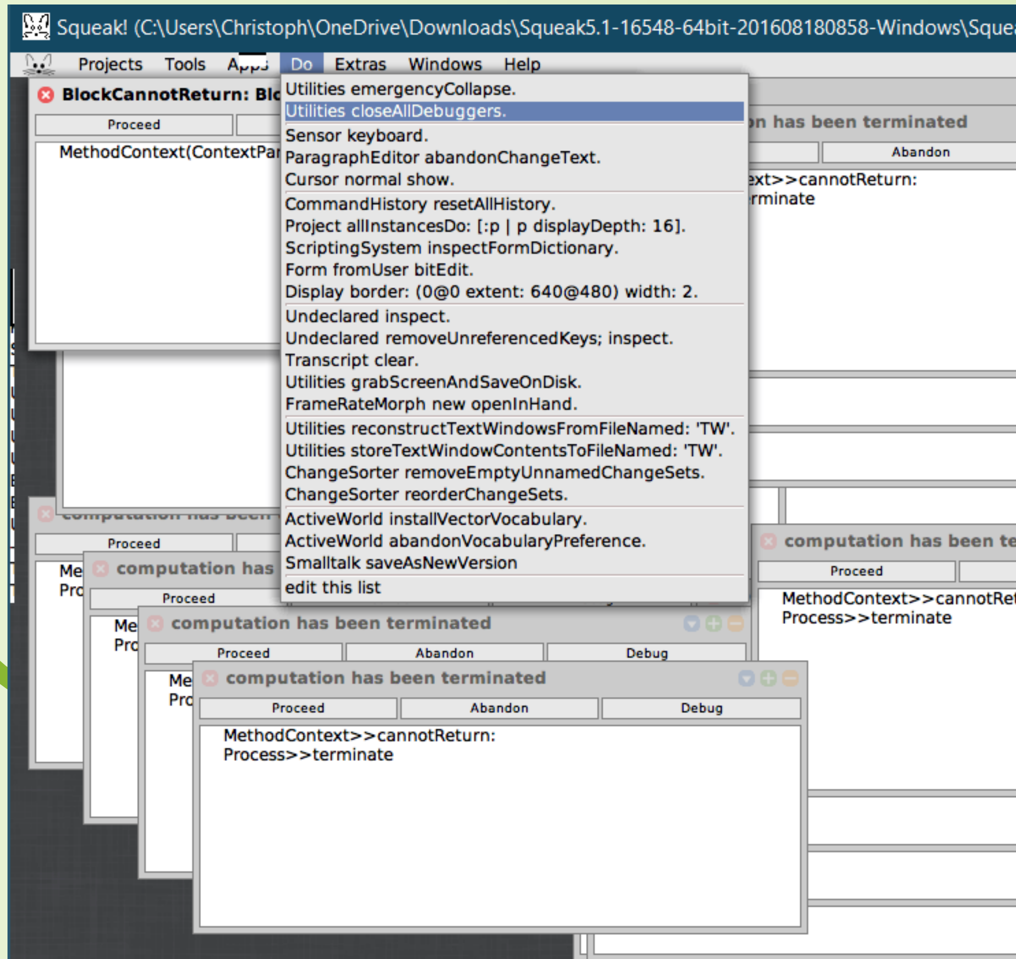
```
  "Answer the currently running Process.“
  ^ activeProcess effectiveProcess
```

StandardToolSet»handleError: anError

"Double dispatch. Let the active process take care of that error, which usually calls back here to #debugProcess:..."

^ Processor basicA~~a~~ctiveProcess
debug: anError signalerContext
title: anError description

Debugger Chains



MessageNotUnderstood: SmallInteger>>foo

```
SmallInteger(Object)>>doesNotUnderstand: #foo
Message>>sentTo:
SmallInteger(Object)>>doesNotUnderstand: #foo
UndefinedObject>>DoIt
CompiledMethod>>valueWithReceiver:arguments:
[] in Process class>>forMethod:receiver:
```

Proceed | Restart | Into | Over | Through | Full Stack

```
doesNotUnderstand: aMessage
	"Handle the fact that there was an attempt to send the given message
receiver does not understand this message (typically sent from the machin
sent to the receiver and no method is defined for that selector)."

	| exception resumeValue |
	(exception := MessageNotUnderstood new)
		message: aMessage;
		receiver: self.
	resumeValue := exception signal.
	^exception reachedDefaultHandler
		ifTrue: [aMessage sentTo: self]
		ifFalse: [resumeValue]
```

self
all inst vars

<- Select receiver's field

thisContext
stackTop
all temp vars
aMessage
exception

Context»#runUntilErrorOrReturnFrom:

```
Context»runUntilErrorOrReturnFrom: aSender
	"ASSUMES aSender is a sender of self. Execute self's stack until aSender returns or an unhandled exception is raised. Return a pair containing the new top context and a possibly nil exception. The exception is not nil if it was raised before aSender returned and it was not handled. The exception is returned rather than openning the debugger, giving the caller the choice of how to handle it."
	"Self is run by jumping directly to it (the active process abandons thisContext and executes self). However, before jumping to self we insert an ensure block under aSender that jumps back to thisContext when evaluated. We also insert an exception handler under aSender that jumps back to thisContext when an unhandled exception is raised. In either case, the inserted ensure and exception handler are removed once control jumps back to thisContext."

	| error ctxt here topContext |
	here := thisContext.

	"Insert ensure and exception handler contexts under aSender"
	error := nil.
	ctxt := aSender insertSender: (Context
		contextOn: UnhandledError do: [:ex |
			error
				ifNil: [
					error := ex exception.
					topContext := thisContext.
					ex resumeUnchecked: here jump]
				ifNotNil: [ex pass] ]).
	ctxt := ctxt insertSender: (Context
		contextEnsure: [error ifNil: [
			topContext := thisContext.
			here jump] ]).
	self jump.  "Control jumps to self"

	"Control resumes here once above ensure block or exception handler is executed"
	^ error
		ifNil: [
			"No error was raised, remove ensure context by stepping until popped"
			[ctxt isDead] whileFalse: [topContext := topContext stepToCallee].
			{topContext. nil}]
		ifNotNil: [
			"Error was raised, remove inserted above contexts then return signaler context"
			aSender terminateTo: ctxt sender.  "remove above ensure and handler contexts"
			{topContext. error}]
```

```
Generator on: [:stream |
	stream nextPut: #foo]
```

Further reading :-)

- Thread on “Debugger chains”: [squeak-dev] Fixing the infinite debugger chains?
- Simulation of Objects as Methods: Kernel-ct.1357 (Trunk)
- Thread on sender swaps and #runUntilErrorOrReturnFrom:: [squeak-dev] BUG/REGRESSION while debugging Generator >> #nextPut:
- In your image …:
  - Debugger>>step{Over, Through}
  - Process>>evaluate:onBehalfOf:
  - {Process, Context}>>step:
  - Context>>runUntilErrorOrReturnFrom: