HaverOnCuis Documentation

Release 0.0.4

Gerald Klix

Apr 30, 2022

CONTENTS

News

- Release `Alpha 4` will be released on 2022-04-29.
- Started to create a new release: `Alpha 2`. It will be available on 2021-05-31.
- Moved my blog – The Haver – to a web-site of it's own.
- Version `Alpha 1` is out.

CHAPTER

ONE

OVERVIEW

HaverOnCuis – for short Haver – provides an opensmalltalkvm based Smalltalk with modules. Haver's module semantics are inspired by Python's modules and, to a greater extent, by Scheme's module semantics.

Important: Please note that Haver is an *extension* of Cuis and *not a fork*. For any conceivable future it will remain an extension!

Haver's name was inspired by the Proclaimers' song 'I'm Gonna Be (500 Miles)':

> And if I haver, hey I know I'm gonna be
> I'm gonna be the man who's havering to you

1.1 Simple Example

Classes with the same name as global classes can be placed in a module like this:

```
SystemWindow subclass: #SystemWindow
      instanceVariableNames: ''
      classVariableNames: ''
      poolDictionaries: ''
      category: 'SystemMorphs'
      inModule: #SystemMorphs
```

This class can be accessed like this:

```
Modules>>#SystemMorphs>>#SystemWindow
```

Environments can be created with this message send:

```
Modules environment: #MyNewModule
```

If one uses >> like:

```
Modules>>#MyOldModule.
```

the module is not created, if it does not exist.

1.2 Installation

In the future Haver can be downloaded from https://www.klix.ch/haver/releases:

- **Haver-unix-linux-gnu-x86_64-64bit-4-alpha-5116.zip** for 64-bit Intel x86 Linux system
- **Haver-unix-linux-gnu-aarch64-64bit-4-alpha-5116.zip** for PinePhones and other 64bit ARM Linux Systems. May run on a Raspberry Pi.
- **Haver-unix-linux-gnueabihf-armv7l-32bit-4-alpha-5116.zip** for Raspberry Pis with old 32-bit userspace and kernel and maybe other 32bit ARM Linux Systems.
- **Haver-Win32-10.0-X64-64bit-4-alpha-5116.zip** for Intel/AMD 64-bit Windows (10).
- **Haver-Win32-10.0-IX86-32bit-4-alpha-5116.zip** for Intel/AMD 32-bit Windows (10)

Currently there is no MacOS version available[1]. However you can provide your own and use the images.

1.3 Starting the Image

The file should be unzipped like[2]

```
unzip Haver-unix-linux-gnu-x86_64-64bit-4-alpha-5116.zip
```

```
cuis
```

Will start the original Cuis image, with *no* updates installed.

```
haver
```

Will start the original Haver image, with all updates installed[3]. *Please note that image will be generated from source code, when started after unpacking. This will take some time, but saves space in the ZIP-file distributed.*

1.4 Internet Presence

The whole documentation is available at in the following formats:

HTML http://haver.klix.ch/ (Also serves as *Haver*'s homepage)

Online Demo https://www.klix.ch/haver/SqueakJS/haver/haver.html

This demo uses Vanessa Freudenberg's SqueakJS.

PDF http://haver.klix.ch/pdf/HaverOnCuis.pdf

Development takes place on sourcehut:

Repository (Mercurial) https://sr.ht/~cy-de-fect/HaverOnCuis/

Issue Tracker https://todo.sr.ht/~cy-de-fect/HaverOnCuis

Linux Virtual Machine https://github.com/CyDefect/opensmalltalk-vm/tree/HVR_MVP_ALPHA4_WINKEY_2022-04-21

Help can be obtained at:

Mastodon @Haver@mastodon.technology

[1] This probably will not work on Windows.
[2] Needless to say, that only the updates available up to the zip-file creation time are included.
[3] Due to COVID-19-induced financial restrictions this will be the case for the foreseeable future.

IRC irc://freenet.net/#Haver

IRC channel called *#Haver* at Libera (irc://irc.libera.chat/#Haver). My nickname is `CyDefect`, from time to time I am available there.

1.5 Releases

You will find information about releases and release-planning in the *Releases and Planning* section.

CHAPTER

TWO

THEMES

Haver adds two new themes to the standard themes.

A dark theme called *HaverDarkTheme*:

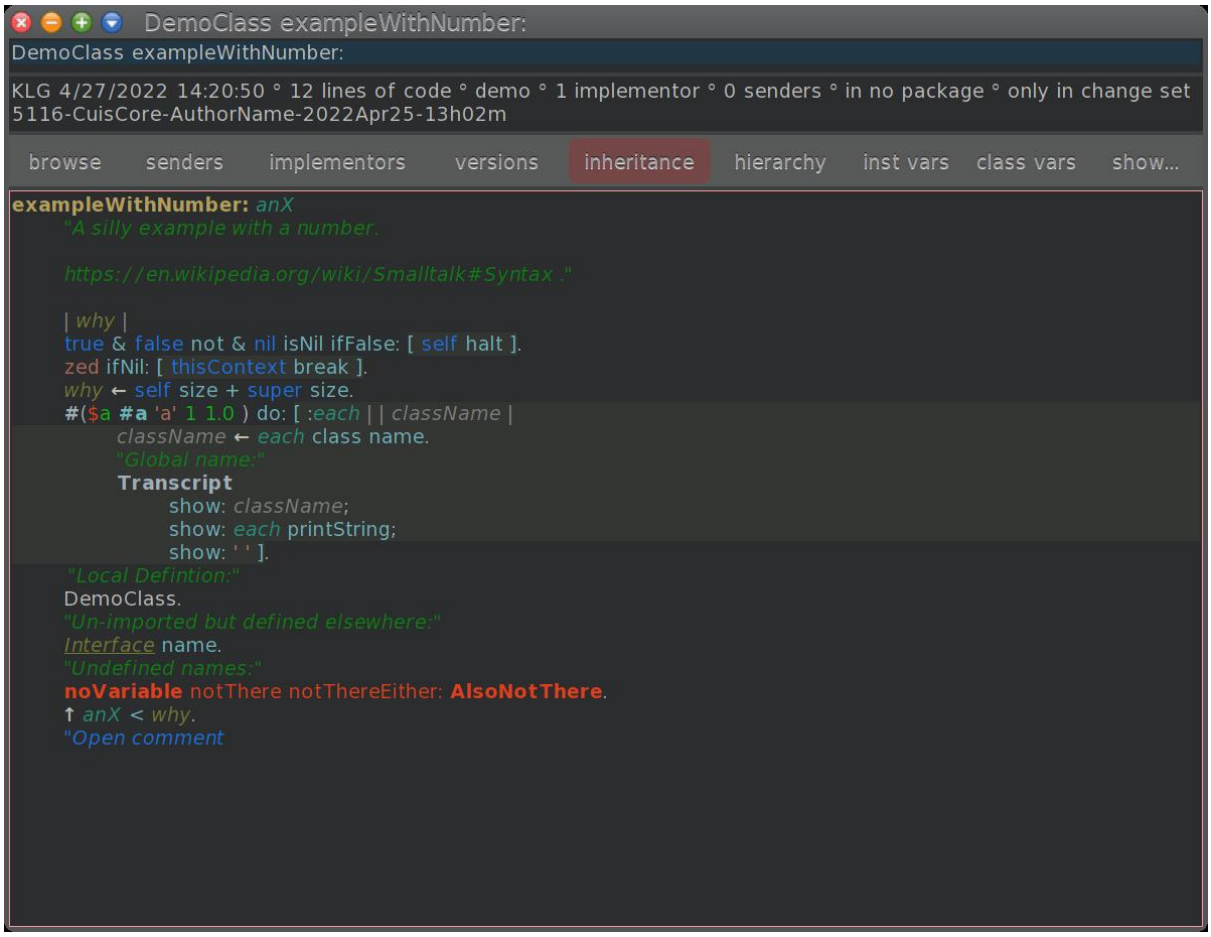


Fig. 1: Haver's dark theme.

and another theme called *HaverDarkPhisherBryceTheme*.The later adds distinctive colors for the shout syntax highlighting and is the default theme[1]:

Themes can be selected with the *Preferences* menu – select *World* → *Preferences* → *Extension Themes* . . . ..[#]:

[1] I am aware that some people will accuse me of endangering them with eye cancer, but the colors help me to overcome the disadvantages of my diminishing eyesight.

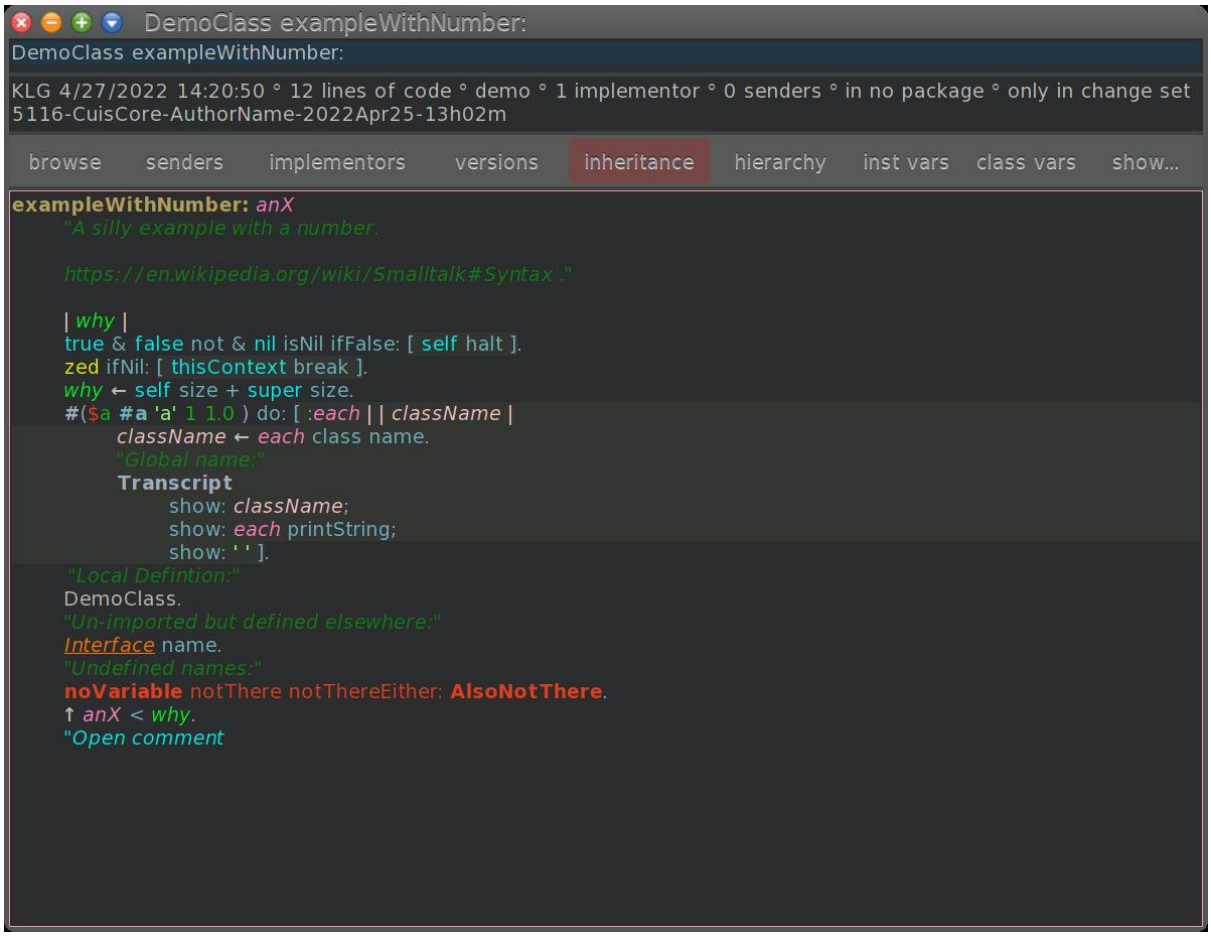


Fig. 2: Haver's dark theme with distinctive syntax highlighting colors.

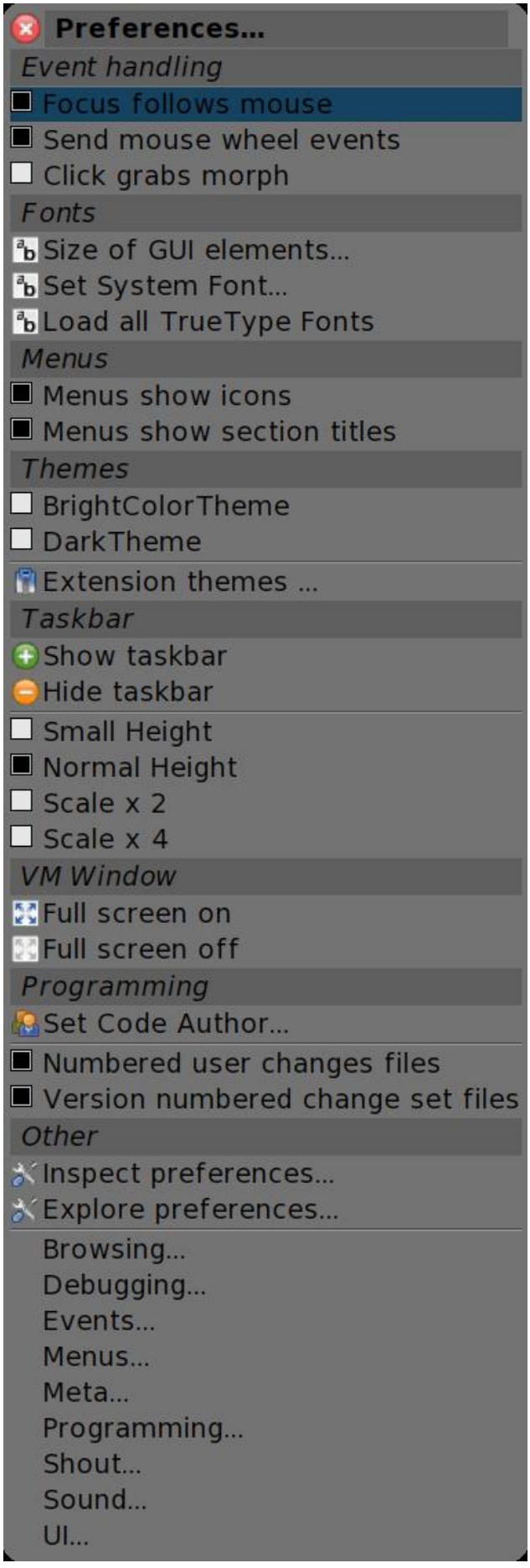


Fig. 3: Haver's preferences menu.

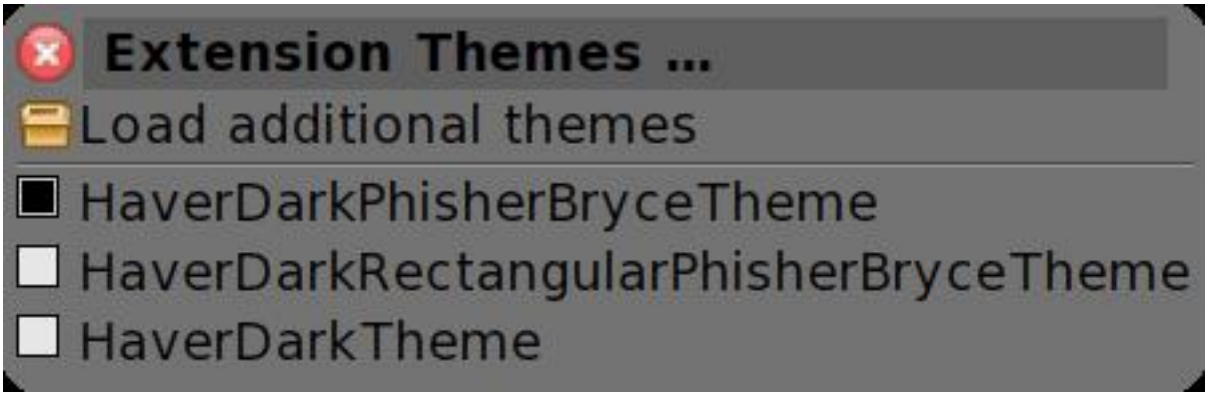


Fig. 4: Haver's extension themes menu.

CHAPTER

THREE

THE SEMANTICS OF ENVIRONMENTS AND MODULES

This section describes the semantics of modules.

3.1 Description

The following sections provide a sketchy description of the semantics of modules.

3.1.1 Environments

Modules are bases on so called environments. Environments fulfill basically the same purpose as *Smalltalk* the single instance of `SystemDictionary`. They bind names that objects. Nearly all of the objects bound in an Environment are classes. All classes in an environment are immediately available to the code in all other classes bound the same environment by just mentioning their name. *Obviously classes defined in an environment can be used by classes outside their environment only with some hassle*.

3.1.2 Modules

Modules however provide means to make bound names available to other modules by exporting them. Exported symbols can be grouped into so called *Interfaces*. Otherwise Modules are just environments.

3.1.3 Interfaces

Interfaces can be imported by other modules.

When a module is created it starts out with two *Interfaces*:

***API* The Application Programming Interface** This interface should contain all the classes that are useful to the module's clients, that is to classes that want use (instances of) the modules classes.

***SPI* The System Programming Interface** This interface is created as an alias of the *API*[1]. It should contain all classes another module or class needs to enhance the functionality of the exporting module. Usually this is done by sub-classing these classes.

Some of Haver's own modules defined so called *UTIs* (Unit Test Interfaces) that export classes which are not exported by the *SPI*. This should make white-box testing easier.

Note: Of course one can name a module's interfaces as one pleases.

---

[1] There is *no* UI-support to explicitly create interfaces that are aliases for other interfaces. Likewise new interfaces can *not* be created as an alias.

3.1.4 Imports

Modules can import an interface by creating an *Import Specification*. *Import Specifications* can import an interface of another module as a whole or they can explicitly select symbols of an interface. It also possible to specify, that you want to import all symbols of an interface and exclude it some of the exported symbols.

It is possible to import an interface more than once into a module.

3.1.5 Renaming

Symbols can be renamed upon export and import.

3.1.6 Packages

Each module, interface, exported symbol, import specification and imported symbol has an associated package, denoted by the package's name. The default package name is the module name (of the interface, exported symbol, import . . .).

The export an import definitions are stored in the package-file of the package denoted by the package name.

Note: This makes it possible to specify a different package name for a Unit Testing Interface *UTI* and store the definition of this interface in a special unit test package.

The FileFinder package and its unit test package is a good example for this technique.

3.2 Formal Semantics

In the following section I try to give a formal definition for the semantics of modules and environments.

`Environment Identifiers` The set of "Environment Identifiers" is a finite set of arbitrary Smalltalk objects.

`Symbols` In essence symbols are immutable strings.

Symbols come in two variants:

`Interned Symbols` "Interned Symbols" are those symbols known to the symbol class. Two interned symbols are identical wrt. to == when their character strings are equal wrt. =. This code snipped may serve as an explanation:

```
('one', 'two') asSymbol == 'onetwo' asSymbol
```

evaluates to *true*[2]. Especially:

```
Symbol lookup: 'onetwo'
```

and:

```
Symbol lookup: 'one', 'two'
```

should evaluate to the same symbol: *#onetwo*

`Uninterned Symbols` All other symbols are uninterned.

All symbols in Haver respond to the message *environment*. All interned symbols answer *nil* when the message *environment* is sent no them.

[2] This should be true for every Smalltalk implementation.

Local Symbols Local symbols answer an environment identifier when the message *environment* is sent to them. This environment is not *nil*.

It follows from above propositions, that all local symbols are uninterned symbols.

Environments "Environments" are mappings from local symbols to arbitrary Smalltalk objects. All symbols in the domain of the aforementioned mapping answer the environment when sent the message *environment*.

A local symbol can be created from any symbol by sending the message *#forEnvironment:* to it. The message needs an environment identifier as an argument.

Bound Symbol A bound symbol is a symbol that is the domain of an environment's mapping.

Modules "Modules" are environments associated with two other sets, A set of interfaces and a set of import specifications.

Export Specification An export specification is a mapping from a symbols to symbols.

Invalid Export Specifications An "invalid export specification" is an export specification whose domain symbol is not bound in the module.

Interface An interface is a set of export specifications.

Symbol Import Specification An "symbol import specification" is mapping from an export specification to a symbol.

Import Specification An import specification is a pair consisting of an interface and a set of symbol import specifications.

Invalid Import Specification An "invalid import specification" is an "import specification" if its interface is not a member of any module's set of interfaces.

Invalid Symbol Import Specification An "invalid symbol import specification" is a symbol import specification if its export specification is not a member of any module's interface or if its export specification is invalid.

The mappings in export specifications and symbol import specifications provide a means to rename a symbol on export and on import.

If the compiler uses *#bindingOf:* to find the binding of a global variable the binding is searched first in the environments then in *Smalltalk*.

In the case of an environment the binding only searched in the bindings of an environment[3].

In the case of a module the same is done. If this yields no binding, all valid symbol import specifications are searched for a matching name and the association in the exporting module is answered.

[3] Of course users may implement their own environment managers with vastly different semantics.

CHAPTER

FOUR

MODULE TOOLS

Haver provides three UI tools to deal directly with modules. Additionally the browsers have been enhanced to select a current module for various purposes.

Both tools can be opened by using the (enhanced) *Open* menu – select *World* → *Open*:

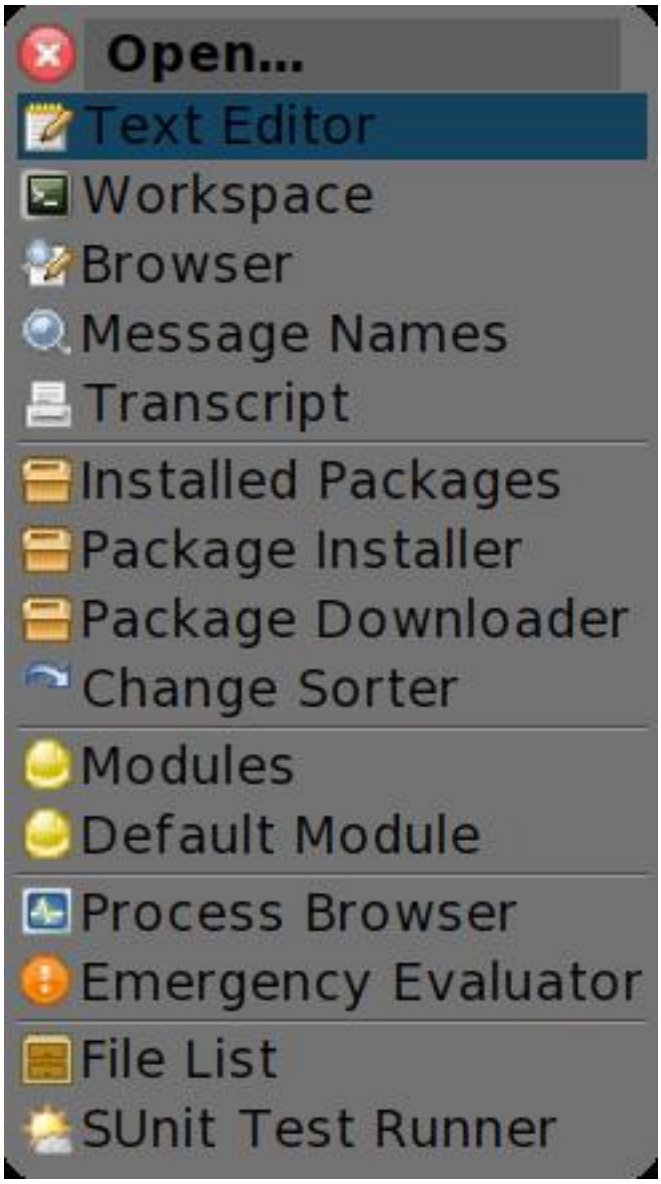


Fig. 1: Haver's Open menu.

4.1 The Modules Browser

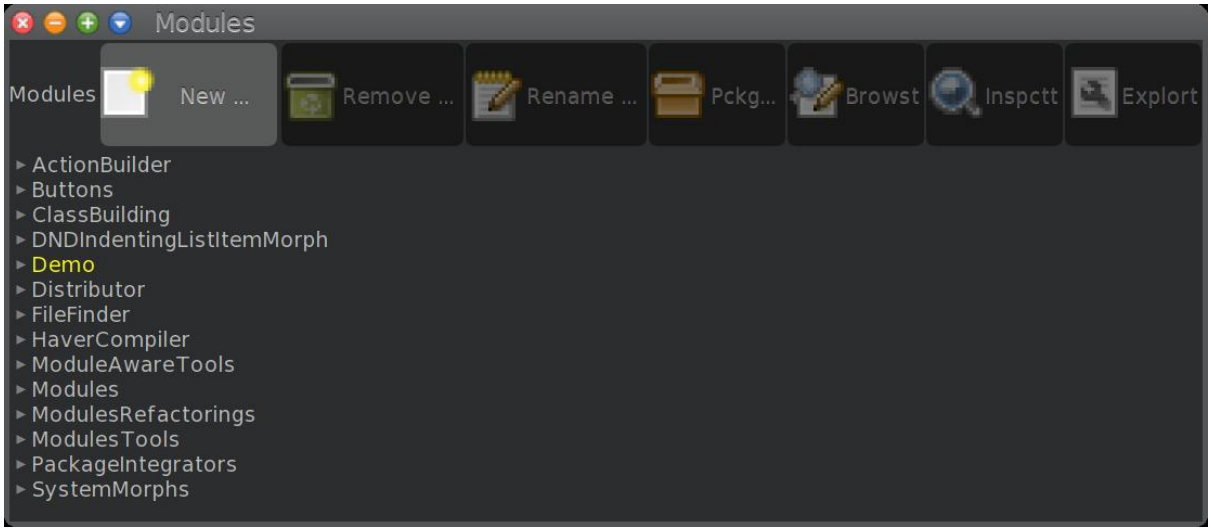


Fig. 2: The Module Browser (Video)

The modules browser on can create, rename, remove and browse modules. All the actions can either be performed with the usual pop-up menu or with the buttons at the top of the window.

Note: If the module name is highlighted with yellow the module has no package it can be stored in. In this case you can either define a package with same name as the module or assign a package name with *Package ...* button. This video shows how it's done.

The hierarchical list contains all modules defined in the image at the first level. The second level shows all the objects bound to names local to the module.

The *Inspect It ...* and the *Explore it ...* open an Inspector or an Explorer on the selected item:

When a module is selected the *Browse It ...* button opens a Browser to change the module's exports and imports.

4.2 The Module Browser

As mentioned in before the module browser can be opened by selecting a module in the modules browser and push the *Browse It ...* button:

The following sketches the various actions the module browser can perform:

A brief explanation of all the actions the module browser's actions on interfaces and exported symbols follows suit:

Package ... Set the package of the selected item as show in this video.

New ... Asks for a interface name and creates an interfaces with that name.

Remove ... Removes the selected module from the collection of modules.

Rename ... Asks for a new interface name and renames the interface.

Add ... Displays a menu of symbols bound in the module and lets you chose one to export. The same effect can also be achieved by dragging a symbol bound in the module to interface item.

Remove ... Removes the selected symbols from the interface

Edit ... Edits the name of the exported symbol.

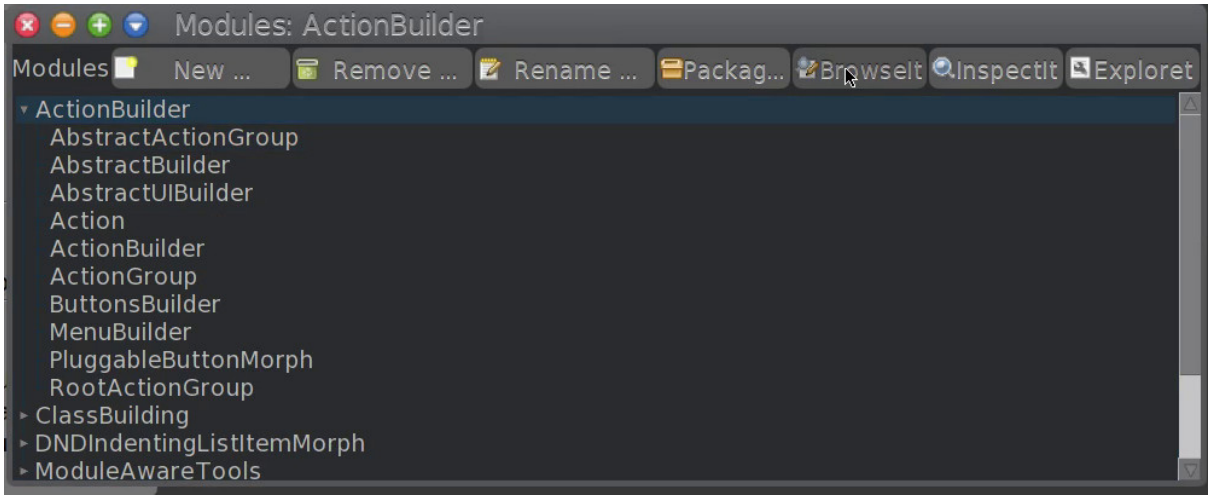


Fig. 3: Opening the Module Browser (Video)

The actions possible on import specifications are explained here:

Package ... Set the package of the selected item.

New ... Displays a menu of modules and the a menu of interfaces exported by the selected module. The same effect can be achieved by drag and drop.

Remove ... Removes the selected import from the collection of imports.

Add an exported symbol An exported symbol can only be added by drag and drop[1].

Import all This action removes all explicitly imported symbols and again imports all symbols the interface exports.

Exclude This imports all symbols exported by the interface excluding those explicitly mentioned.

Edit ... With this action you can rename an imported symbol.

Remove ... Removes the selected symbol from the collection of imported symbols.

This video shows the effect of the aforementioned actions:

4.3 The Default Module Browser

To make working with modules easier a browser that lets one select a default module where classes are defined in and global symbols are searched before they are looked up in *Smalltalk*.

This video should explain the semantics of the default module settings:

The default module browser is especially helpful, when one wants to file a non-module-aware package into a module.

Finally we can define *API* and *SPI* for the Spacewar! package:

[1] I wanted to avoid the three consecutive menus this would have needed. It's ugly to implement and unpleasant to operate.

Fig. 4: The Module Browser's Actions

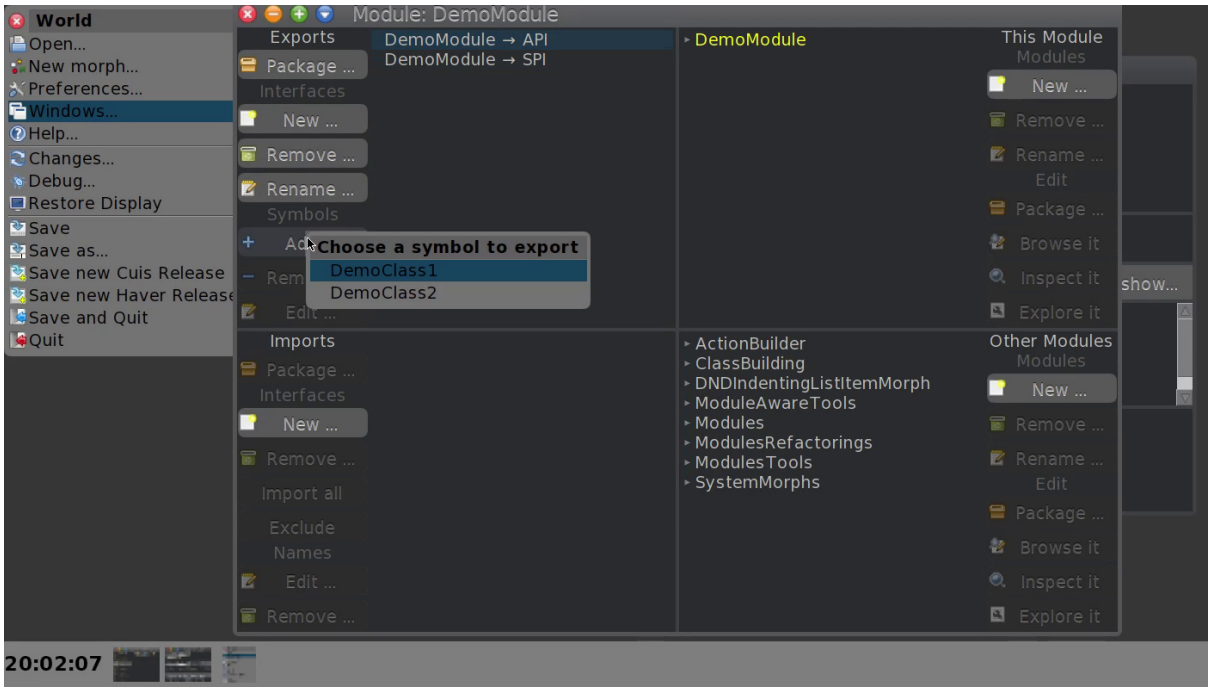


Fig. 5: The *Add* action (Video)

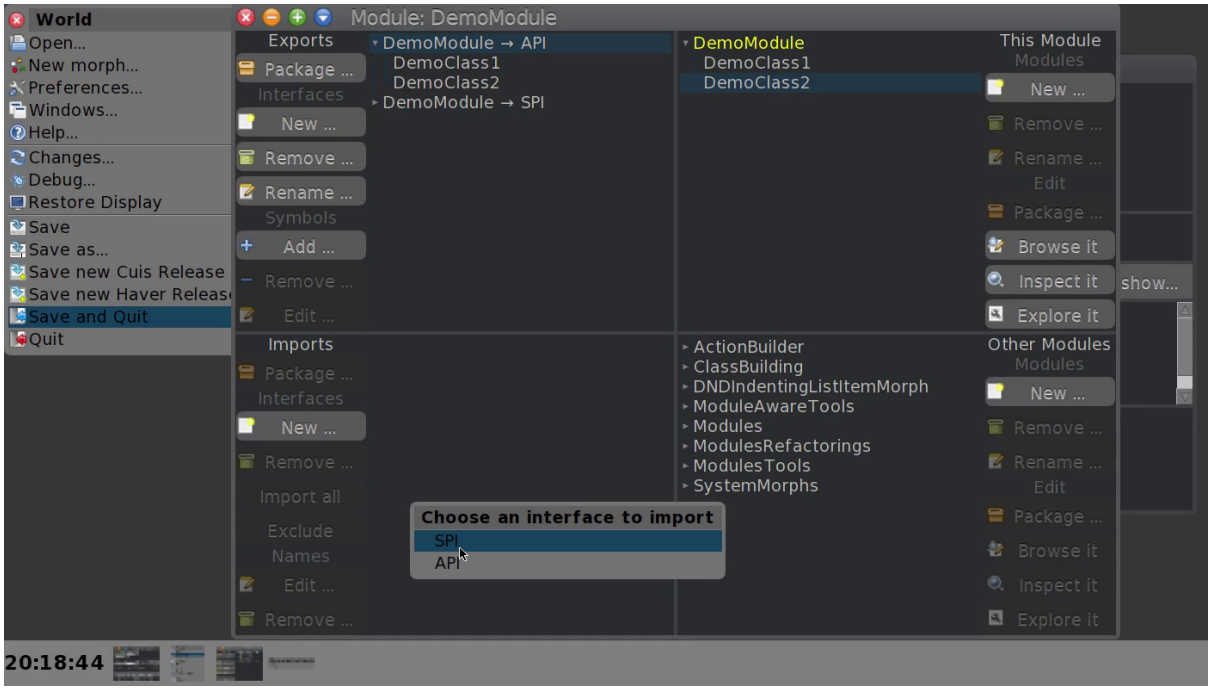


Fig. 6: Importing an interface (Video)

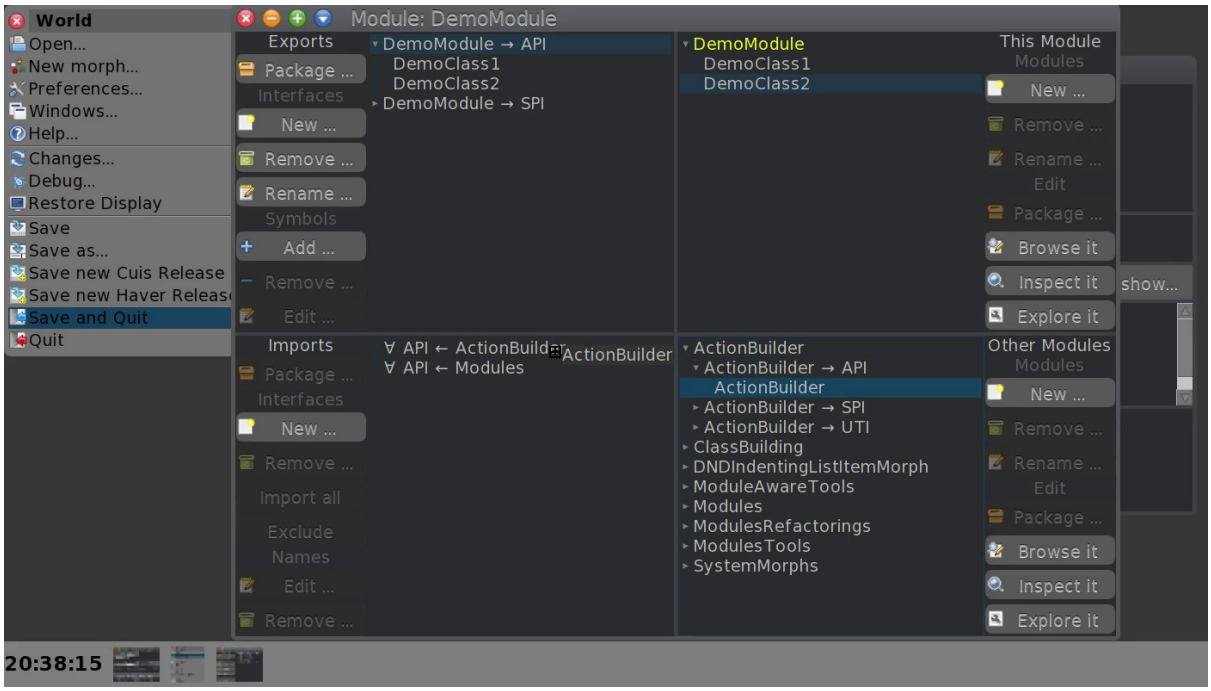


Fig. 7: Adding a specific symbol to an imported interface (Video)

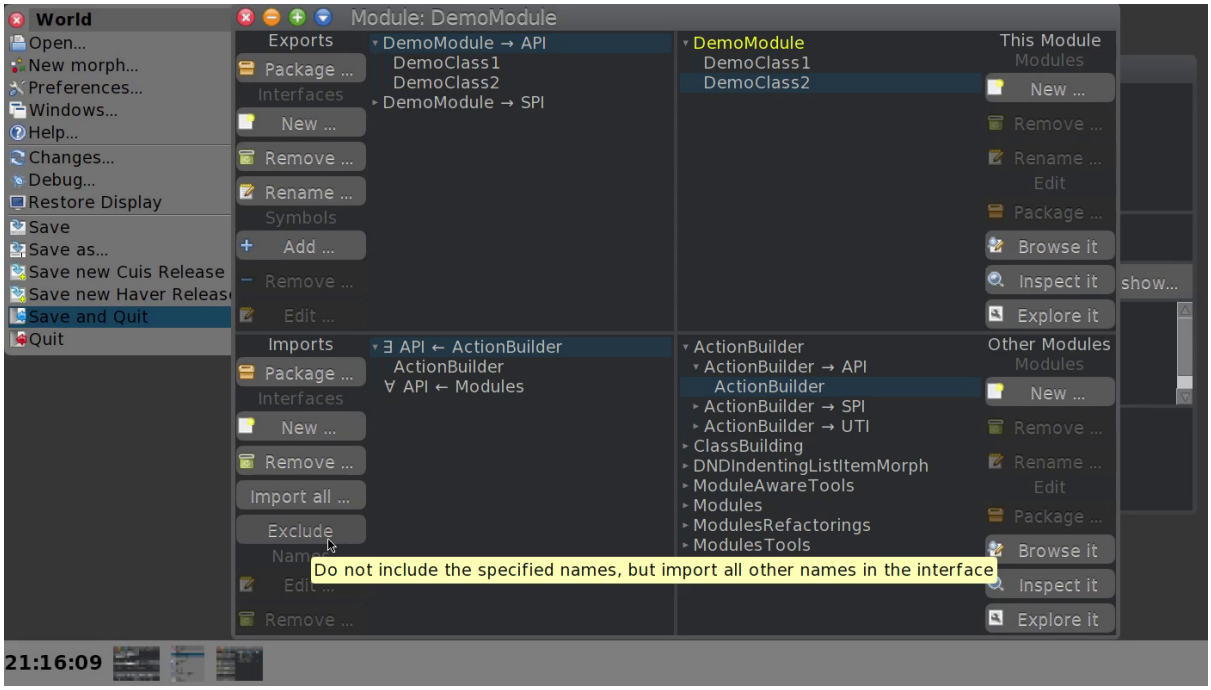


Fig. 8: Additional actions on imported symbols (Video)

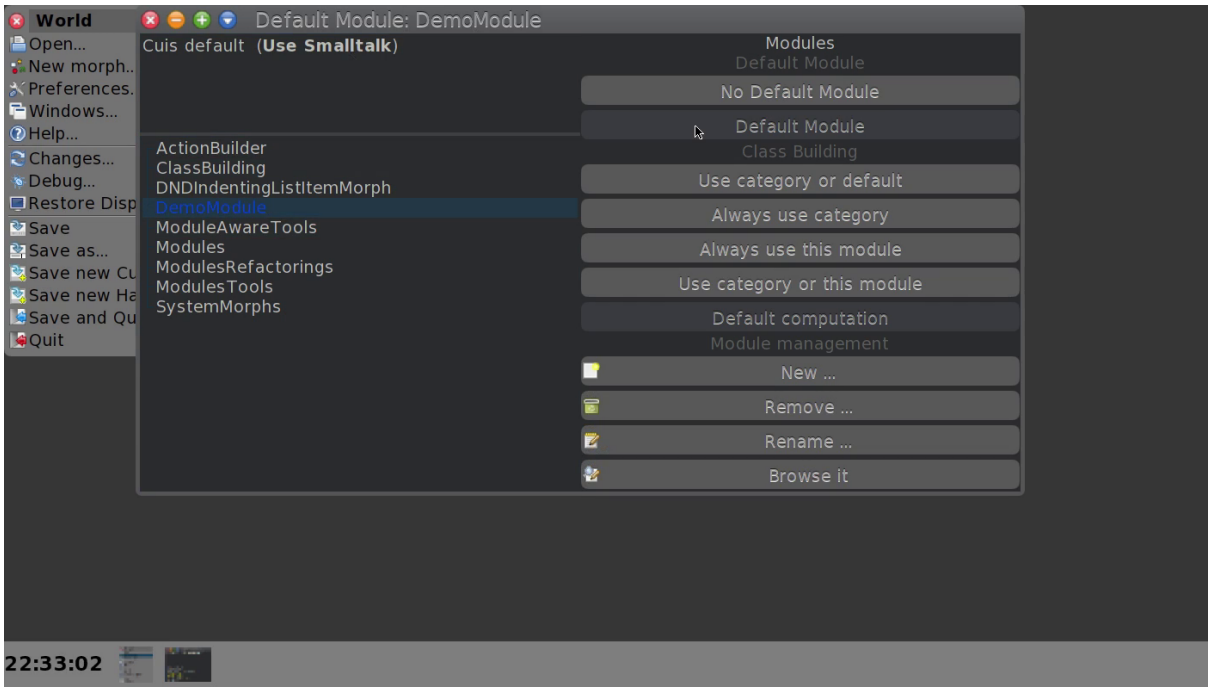


Fig. 9: A longer video explaining the Default Module Browser

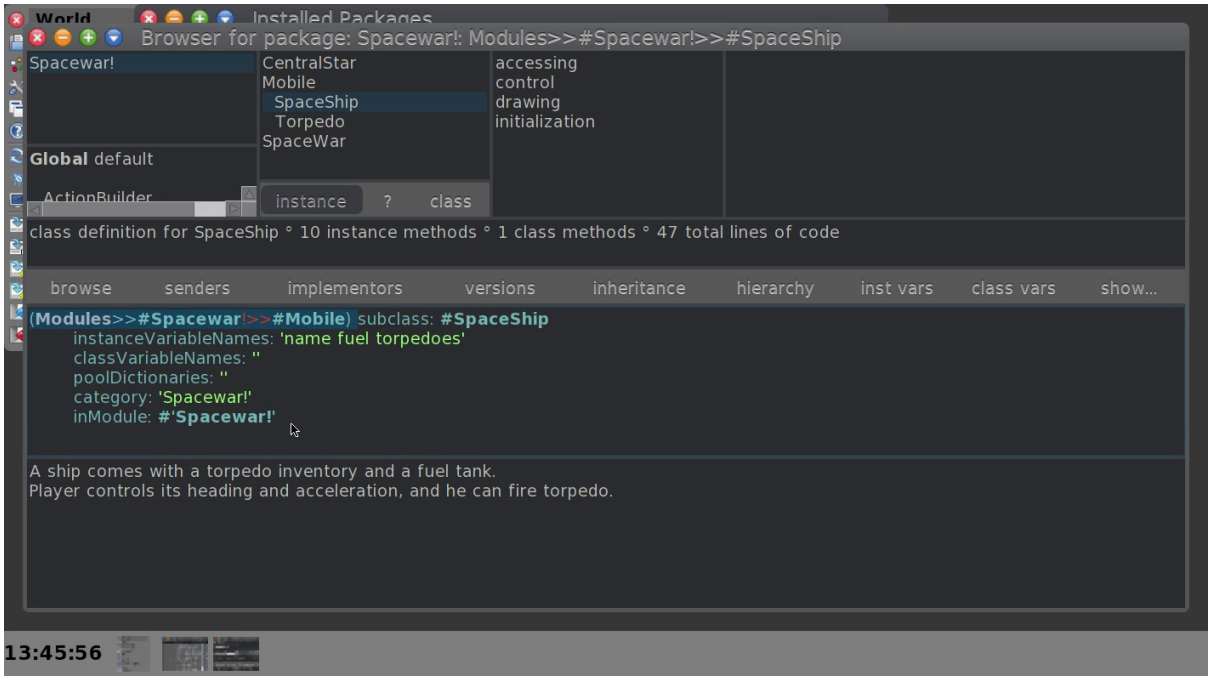


Fig. 10: Convert the Spacewar! package into a Spacewar!-module and -package (Video).

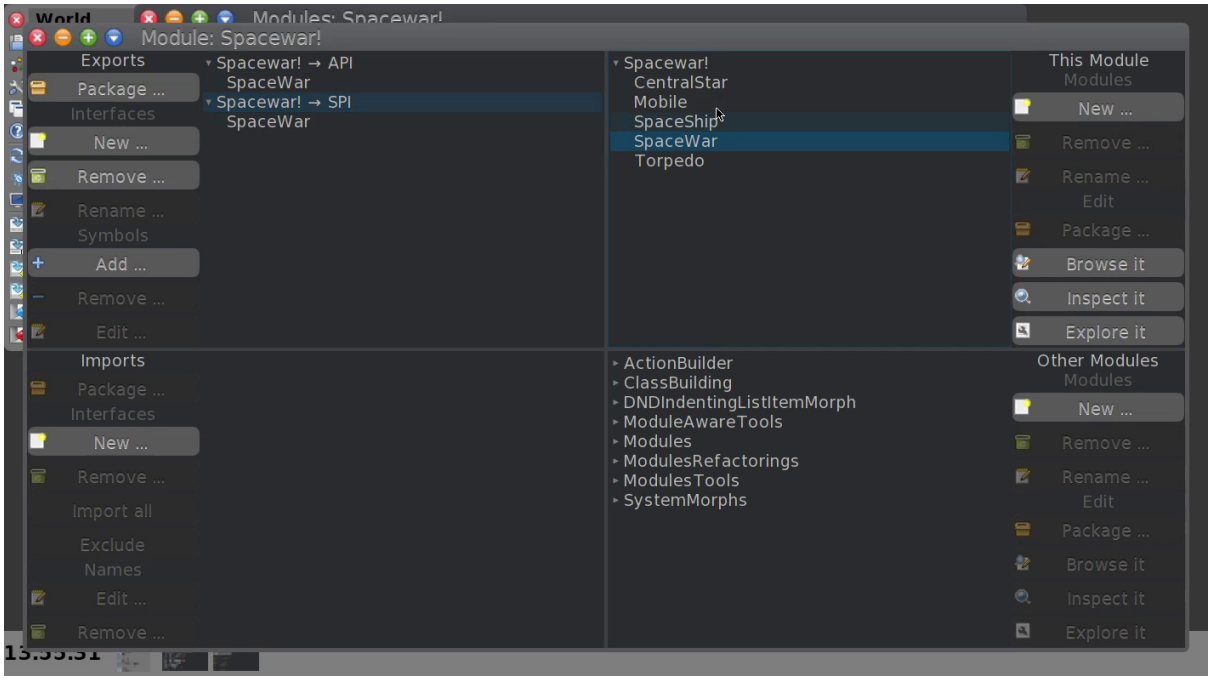


Fig. 11: Define interfaces for the Spacewar! module (Video)

4.4 Modifications to Existing Tools

The following tools were modified to make Smalltalk development with modules easier.

4.4.1 System and Hierarchy Browser

The System and the Hierarchy Browser windows were modified to contain a local default module browser, with the same functionality as the global default module browser.

If want to define Module named *Scratch* one can perform the steps shown in the video below:

4.4.2 Workspaces

While objects – especially classes – bound in modules can be accessed with the >>, the Workspace window menu gained a modules sub menu.

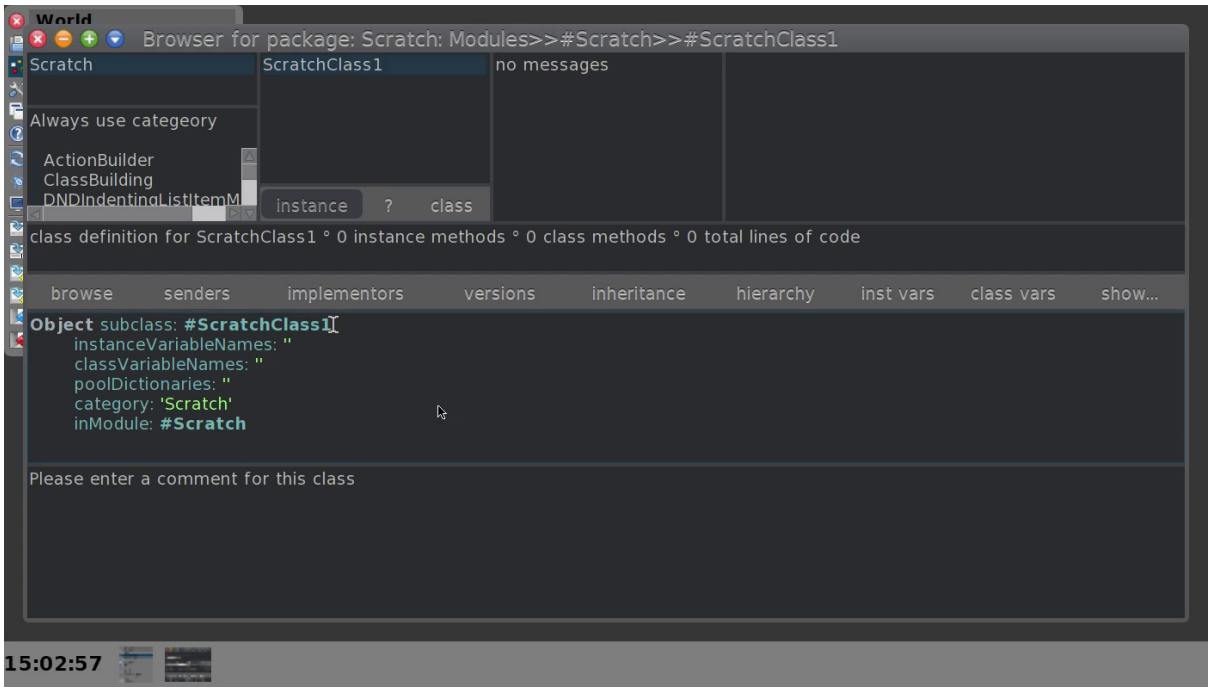


Fig. 12: Define a package, a category, a module and two classes in four simple steps.

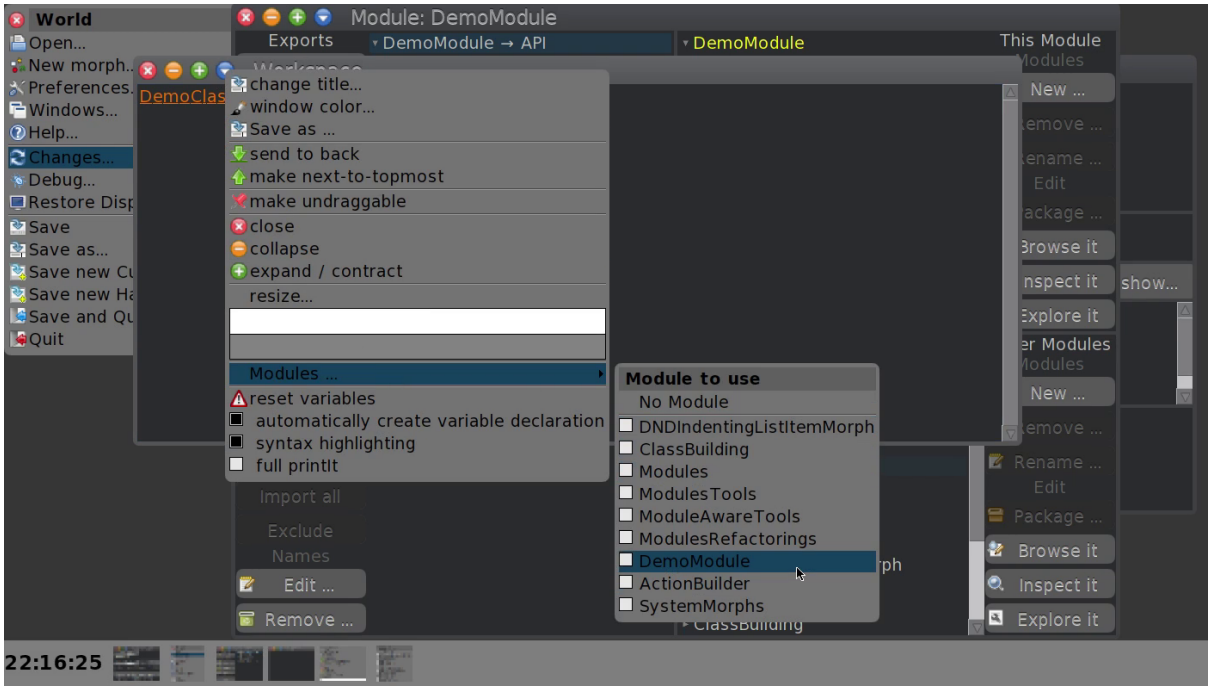


Fig. 13: The Workspace Modules Menu (Video)

CHAPTER

FIVE

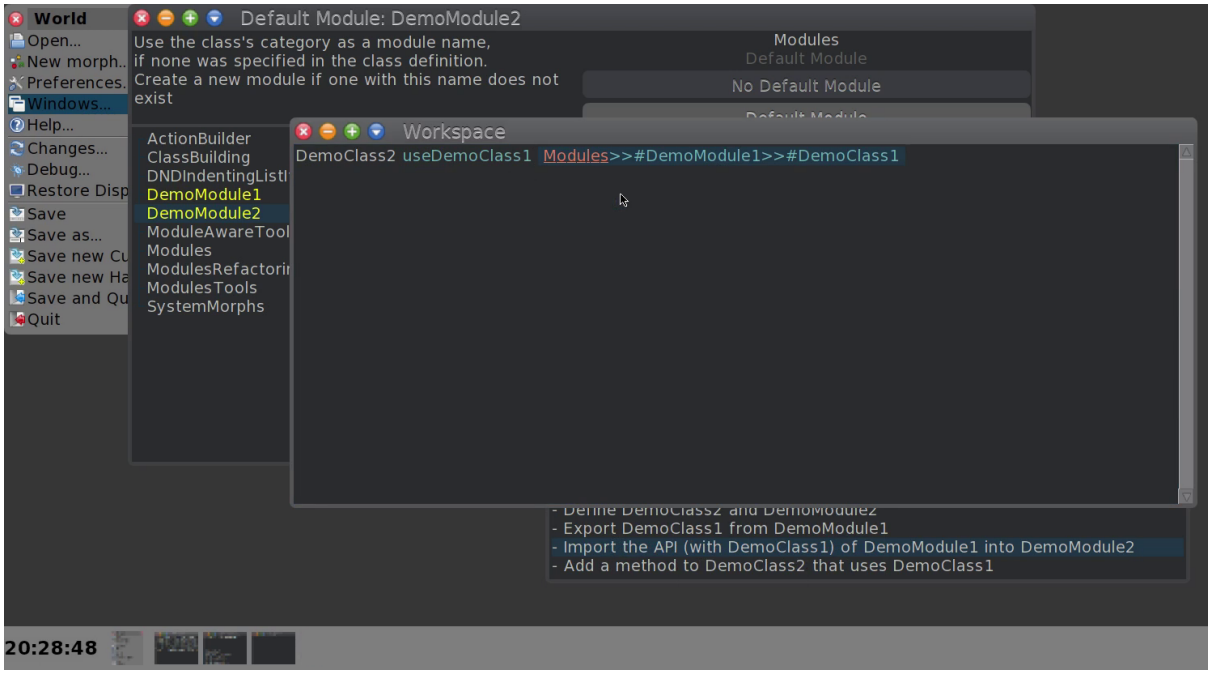
TUTORIALS

This section contains tutorials that (should) guid the user through various use cases of Haver's module system.

5.1 Exporting and Importing a Class

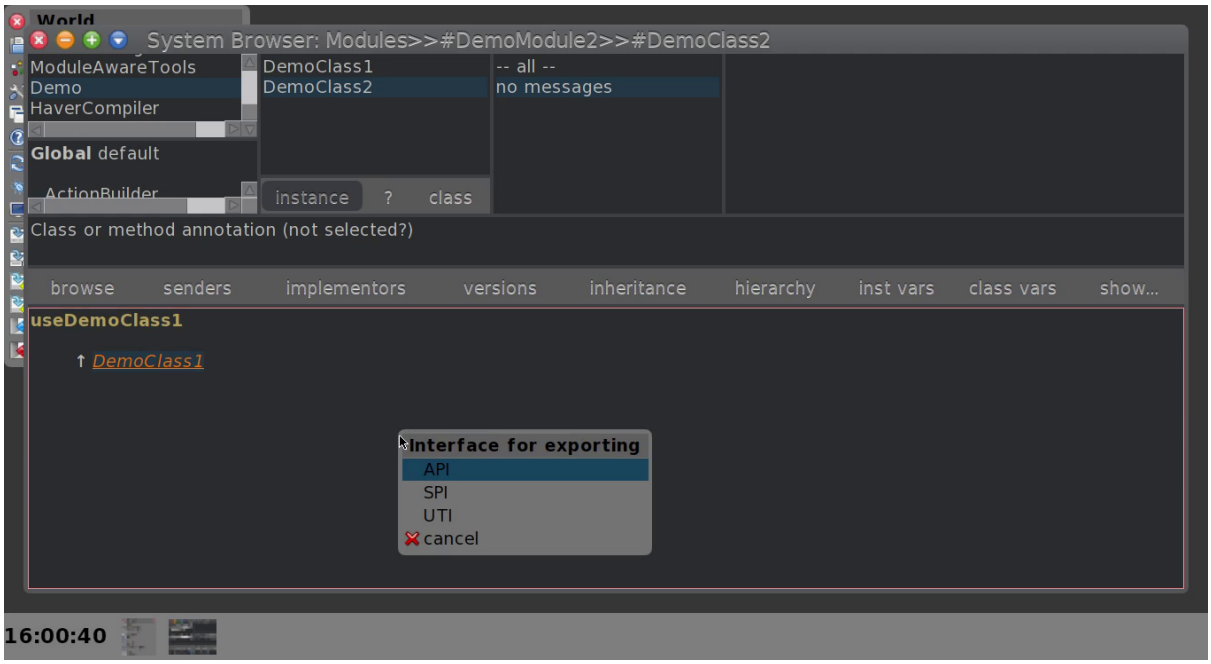
In this tutorial we will define two classes in two different modules:

1. Define the two classes and implicitly both modules.
2. Export one class in its modules application programming interface and import it in the module.
3. Use the one class in a method of the other class.



5.2 Using the Export/Import Wizard

In this tutorial we will define two classes in two different modules and let the wizard export and import one of the classes into the other module.



CHAPTER

SIX

UI TOOLS

Haver provides some additional tools to create user interfaces. These tool were written to implement the module related browsers.

TBD

CHAPTER

SEVEN

VIRTUAL MACHINE

The Linux virtual machine delivered with distributions of Haver are a custom build ones, which make the *Windows*- or *Super*-Key as ay *Cmd*-Key. Probably this is not too useful, because the same key is simulated by pressing *Ctrl* and *Alt* together.

I also changed some optional performance enhancements, such as X11-shared memory and X11 asynchronous updates the defaults:

```
5.0-202202190810-HVR_MVP_ALPHA4_WINKEY_2022-04-21  Thu Apr 21 14:29:24 CEST 2022␣
↪clang [Production Spur 64-bit x86_64 VM]
CoInterpreter VMMaker.oscog-eem.3183 uuid: c5c103b4-2c5c-44e7-8e32-030e79600ca6␣
↪Apr 21 2022
StackToRegisterMappingCogit VMMaker.oscog-mt.3179 uuid: c6fbcb07-2a19-ed4f-8b40-
↪9c119a70882a Apr 21 2022
VM: 202202190810-HVR_MVP_ALPHA4_WINKEY_2022-04-21 bear@speedy:gitwork/
↪opensmalltalk-vm
Date: Sat Feb 19 09:10:30 2022 CommitHash: 922833304
```

CHAPTER

EIGHT

DEFECTS

As we all know, no software is bug-free, especially when it contains more than three line of code.

8.1 Defects Fixed

f49a8518-e49e-4235-821a-68b0d2e9ba38 Patch the SystemDictionary>>#renameClass: Did it in another way, perhaps this method must be patched, too.

OPEN 2020-12-05 12:13:50 | BUSY 2020-12-05 18:06:41 | TESTING 2020-12-05 18:47:29 DONE before 2022-04-29

ea672ea8-1eb6-4d9f-829e-5539d3fc406b The versions browser does not diff, reverting versions does not work.

OPEN 2020-12-06 14:20:21 | TESTING 2020-12-06 15:09:06 | WATCHING 2020-12-09 17:09:58 | DONE 2020-12-21 18:31:50

57400aef-d0ac-483a-a9ff-5b3d1b331921 Reexporting imported symbols will not work. This statement is subject to testing, it was derived solely on theoretical considerations.

The current implementation is untested and crappy, it will not work (2021-01-05).

1) Will remove the imported symbols from the list of bound symbols. (DONE 2021-02-18 13:10:08)

2) Will enable drag and drop for imported symbols. (DONE 2021-02-18 20:31:56)

OPEN 2020-12-06 15:07:50 | BUSY 2021-01-03 14:53:26 | TESTING 2021-01-03 17:27:26 | BUSY 2021-02-18 12:56:57 | TESTING 2021-02-18 20:32:15 DONE before 2022-04-29

030be14a-5be0-43e7-bd8b-630c3dc28b49 Investigate why using a newly created class in a modules or environment in existing classes' code requires recompiling that class. (Might be our missing undeclared handling). Maybe creating new associations messing up #declare:using: is root of this evil. #declare:using: sends #add: which copies the association it adds. Redefining global classes does not work. It should trigger a recompilation. 2022-04-29: It is definitly the missing undeclared handling. I added a workaround that just recompiles everything in module.

OPEN 2020-12-07 12:35:51 | DIAGNOSIS 2020-12-10 14:22:07 BUSY 2020-12-13 17:41:33 | DIAGNOSIS 2020-12-21 18:31:00 | WORKAROUND before 2022-04-29

fdfb56ab-678f-4893-b242-3f26fd4b95c5

Defining a class in an environment with the same name as a global class defined in Smalltalk leads class organizer instance-variables in the browser being nil. See the screenshot below for details.

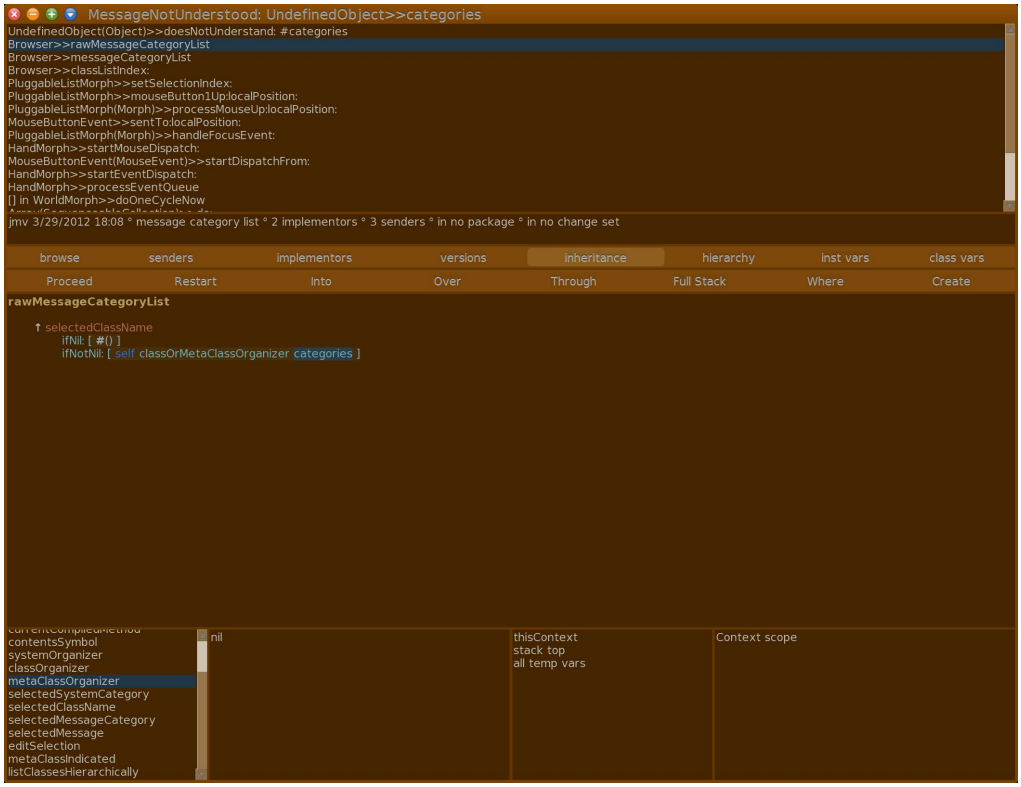
Browser>>#selectedClassName: answers nil for the global class. Using the environment-aware class list for filtering does not work. Obviously we have one case (Action) that is not part of the system organisation. If we have two classes in the same category; one global, another one in an environment, only one defined last is stored in the system organisation.

Categorizer>>#classify:under:suppressIfDefault: uses *#>=* to find the insertion point. This string comparison is by no means environment aware.

Now even Environments are messed up. Of course they are: The environment-awareness test-case resets the symbol-manager.

SystemDictionary>>#classNamed: contained dubious code.

OPEN 2020-12-07 16:35:29 | DIAGNOSIS 2020-12-07 22:16:43 STILL OPEN 2020-12-09 11:03:32 | WATCHING 2020-12-09 17:01:11 DONE 2022-04-29



31c6b8bb-13ea-4d67-ba86-d5f5904a8346 Filling in a package with modules over an existing package leaves one with a debugger.

OPEN 2020-12-09 | DONE 2021-03-23

f6795b8d-0e56-4cb5-b5ce-f7cd66b38d3c Removing a module does not remove it's classes from the system. Keeping the environments in computations leeds to spurious references. (WRONG 2021-02-27 19:03:14). Sometimes the class building stuff picks the wrong module. Got the reason: Classes created by the prameterized class builder are bound in *Smalltalk* instead of their proper environment (2021-02-27 19:44:08).

OPEN 2021-02-27 | BUSY 2021-02-27 16:15:13 | DIAGNOSIS 2021-02-27 19:44:08 | TESTING 2021-02-27 19:53:36

ce53a2c6-e7d1-4d74-bef7-ff8ce7b771a4 Using exactly a module *Always use <Module>* does not work in a browser. The same is true for *Use category or <Module>*. The global default module setting has no influence. The use category stuff does not even create a class, but works for the global setting. In general the global looks OK.

OPEN 2021-03-09 | WATCHING 2021-03-23

290b20af-7f6a-4503-af35-51e31689dafc File outs of interface components miss the package name.

OPEN 2021-04-14 | DONE before 2021-04-28

970e8f95-82e7-427f-b47d-d570e4022847 Inspecting the contents of a code file creates the modules contained in that file. Probably this true for opening a code browser on the package.

Yup it does!

Fix: Delete the spurious module.

See: https://todo.sr.ht/~cy-de-fect/HaverOnCuis/3

OPEN 2020-12-09 | DONE 2022-01-20

8.2 Know Limitations

96007d90-9859-4176-9688-6d2a59ca188e Sometimes imports of class or classes bound in a module become undefined; there name is bound to *nil*. Usually this can be solved be recompiling the class referenced. For the time being I don't know the exact reason.

Best theory so far: The class builder does not handle renaming classes correctly wrt. environments.

See: https://todo.sr.ht/~cy-de-fect/HaverOnCuis/1

Fix: Recompile the class being referenced. 2022-04-29: This workaround is now applied automatically. Undefined handling is still missing.

OPEN 2021-04-28

6501e4a4-76b0-4104-a721-9ce895e6892c On rare occasions import specifications do not point to the export specification they import. Instead they point to the module name symbol. Usually this means, that your image is broken and needs to be recreated.

No valid theory so far.

See: https://todo.sr.ht/~cy-de-fect/HaverOnCuis/2

Fix: None so far, keep backups :-]

OPEN 2021-04-28

7f2bf0b4-8442-4013-a147-fe229ac19608 Change set management is only half implemented. Filing out change-sets with module code may not work, because the author found that issue not too important.

Fix: Safe your module in package.

See: https://todo.sr.ht/~cy-de-fect/HaverOnCuis/4

OPEN 2021-04-28

d9dbfc51-c866-4ffc-bfd0-de3ff36a2352 Haver's version of the plugable button morph does not rotate with the canvas used by vector graphics.

See: https://todo.sr.ht/~cy-de-fect/HaverOnCuis/14

OPEN 2022-04-25

8.3 Bug Reporting

An issue tracker can be found at sourcehut: https://todo.sr.ht/~cy-de-fect/HaverOnCuis

CHAPTER

NINE

FUTURE

This sections contains my sometimes sundry about Haver's future.

9.1 App Image Creation

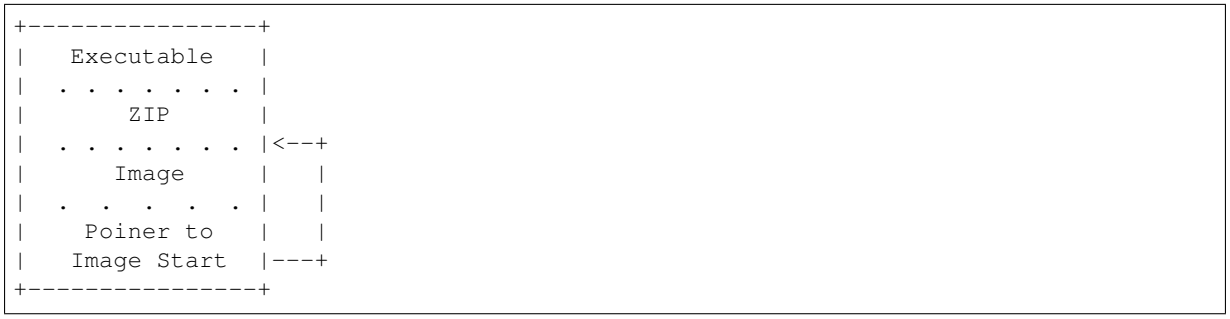
Add support to create minimal images for application delivery by porting the SystemTracer to Haver. I hope it is not slow for serious application creation.

9.2 Single File Application Deployment

One way to deploy Smalltalk, Cuis or Haver application is to modify the virtual machine in the following way:

- Only use internal plugins. Modify the display and audio-plugins and the VM's C code in a way that does not need to be compiled as shared objects or dynamic link libraries.
- Modify the image loading code in a way, that image can be appended to virtual machine's executable.
- Additional files, like font or icon-files, can either be incorporated into the image or stored in a ZIP-file that is also append to the image.

The memory layout is roughly like this:



To my understanding ZIP-files are parsed by searching for their central directory, which points to single compressed files. Any cruft should be ignored.

On the other hand, we also can devise a simple linked list format of our own:

(continues on next page)

(continued from previous page)

```
     .       .      :
     .       .      :
     .       .      :
                    :
|  . . . . . . . |  |
|   PTR to R 2   |--+
|  .  .  .  .  . |
|   Resource 1   |<-|
|  . . . . . . . |  |
|   PTR to R 1   |--+
|  .  .  .  .  . |
|      Image     |<-+
|  . . . . . . . |  |
|   PTR to Image |--+
+----------------+
```

9.3 Application and Package Distribution

An AppStore like application distribution application should also be very useful. Obviously there have been such ideas around for squeak.

I like to call this "App Store" `Haver Silo`. I should have the following features:

- Distribution of Cuis/Haver updates.
- Distribution of Apps,w hich are just Cuis or Haver images.
- Distribution of packages.
- Distributed hash table based implementation with UPD and builtin firewall traversal[1].
- An integrated application to start other applications.

9.4 Really Far Reaching Ideas

All of these ideas need quite a bunch of developers and of course serious funding.

9.4.1 Implement Python inside Haver

With the help of parso it should be easy to create an abstract syntax-tree for Python code with Haver and compile it to byte code. This can be bootstrapped by inter-process communication between a Python interpreter running parso and a Haver image. Of course this will need a Python run-time-system implemented in Smalltalk, which is a lot of work, but doable.

I will enable Haver to use the huge amount of Python software. Combined with Smalltalk's fix and continue capability it would make a great Python IDE.

[1] I am aware this will not be usable behind the firewall of a big organisation, but that is not the target audience.

9.4.2 Implement JavaScript inside Haver

The same approach as sketched above can be chosen to implement JavaScript on top of Smalltalk. Instead of using parso babel should be used. Alas I am by no way a JavaScript expert.

9.4.3 Implement a Sista Aware Compiler

Implement a new compiler that uses SSA throughout and targets the sista version of the opensmalltalkvm.

9.4.4 A Web Browser in Haver

Once we have JavaScript and Sista in place we can implement a complete web-browser. While this is a realy huge project, I can image there is demand for a hackable browser, that is not based on Google or Mozilla code.

CHAPTER

TEN

RELEASES AND PLANNING

A short sketch of the current release plan.

10.1 Minimum Viable Product

The minimum viable product will consist of two releases.

10.2 Alpha

Currently we have public alpha release. This release should provide everything to work comfortable with modules.

`Alpha 0` This version is already out in the wild.

`Alpha 1` `Alpha 1` was oppublished on 2021-05-10.

Features to be – or already – implemented are:

1. Added a refactoring that enables the user to copy a class from one module to another or to copy a global class into a module.
2. Added access-level. Not sure if these will stay.
3. Added a wizard to help users deal with undefined global, e.g. captial names. The wizard tries to import the right classes from other modules with the users' help.
4. Explained the semantics of modules in more detail.
5. Implement SqueakJS at http://haver.klix.ch as a spike. I would be cool if users can test Haver in their browsers. Will be done once this release is out.
6. Check whether the latest VM compiles on Linux, if yes merge the *VectorEnginePlugin* changes into a new branch and rebuild everything. This will probably postpone the release date to 2021-05-14 (one day). No joy on this one, include order was fixed in the Squeak-VMMaker, but not in the Cuis-VMMaker.

`Alpha 2` Will be released on 2021-05-31.

1. Invested a day into other OODBMS options. Philip was not successfull in porting Magma. I came up with a simple object store called PlanE.
2. Started with a (video) tutorial about a simple TODO-list. I am not satisfied with my videos, the lack explanatory text and maybe audio.

`Alpha 3` Will be released when I found some means for better demos.

1. Use CodeRockets's projects to implement demo environments, that are better than videos. *This was not implemented, due to lack of community interest in |Haver|.*

2. Add support for private- and internal extension methods to the compiler. Split the HaverCompiler- and the Modules- packages into a Compiler and a Modules package proper and an access-level package.
3. Implement integration packages that can be used to develop changes to Cuis in a package that is integrated later, removing all extension methods and re-categorizing new classes. *This was implemented in the Alpha 4 release.*
4. Create Haver's image upon first start on the users machine. This way we only need to distribute one image. *This is implemented in the Alpha 4 release.*
5. Document the maturity of the packages included, especially mark the work in progress packages as such. *Postponed to Alpha 5.*

`Alpha 4` Image management.

1. Create an image management and update application. *This will be postponed to the Alpha 5 release and only be implemented if there is community demand.*
2. Design and implement wrapper packages that are loaded before a Cuis package is loaded. This package will define exactly one Module that binds every global definition of the package filed in. Use special meta-classes. that file-out stuff just like a Cuis class. *This feature will not be implemented.* I will implement an "export to Cuis" feature.

`Alpha 5`

1. Application delivery. *Please note, that these features will only be implemented, if there is sufficient community demand.*
    1. Create an image management and update application.
    2. Find some means for application delivery.
        1. Give the old Squeak SystemTracer a shot. Invest up to two days in getting it running. If this is successful invest another day to get rid of the *ModulesTools*-module as a POC. If anything of this fails we will deploy bigger images.
        2. Check whether we can generate an Inno Setup based installer. I did this once for a Python project, it worked well.
        3. Create self-mounting fuse-based zip-file. I already did some experiments with archivemount. A zip file can be appended to executable and the resulting can be made mounting itself. Haver can be started from the mounted file-system, but has problems writing it's change log. With some fixes to the image and archivemount 's main this should work **Big advantage: Users can generate one-file-apps with additional tools.** (Look Ma, no compiler, no AppImageKit :).
2. Better Preferences handling.

Other things to be done in parallel:

1. Ask the Cuis-community, whether they would support either
    - an Haver-based AppStore – `HaverSilo` – for images and packages. This will be based on some P2P-protocol.
    - or if they would rather like some portable means to distribute single file applications.

    Especially ask whether they would support a crowd-funded project, that provides long-term-support. After all the list of dead Smalltalk projects is quit large. I am still under the impression that Haver is not mature and complete enough to ask.

Anything MacOS related will require external funding. I only have two broken Macs and can not afford a new one.

10.3 Beta

There will be a beta release. It will include a tutorial with to create a simple calculator application.

10.4 Release Candidate

The release candidate will be geared towards development of simple standalone application. Haver should help a single programmer, or a small teams of programmers, to develop applications.

Therefore it should have the following additional features:

- A (simple) object oriented database. Currently there is a string preference for porting Magma to Cuis/Haver.
- Some means to distribute applications for Linux. Hopefully more than just zip files.
- Maybe some means to create windows installers.

Alternatively, if I can obtain (crowd) funding for the single file idea application idea that one may be included.

CHAPTER

ELEVEN

ACKNOWLEDGEMENTS

Without Juan Vuletich's efforts to make Cuis a simple and understandable system, I would not have stood a change to implement such a complex piece of software as Haver.

Thank you Juan!

Ken Dickey's PackageEnvironments provided an encouraging starting point for Haver. Ken also provided valuable input.

Thanks Ken!

CHAPTER

TWELVE

COPYRIGHT

CHAPTER

THIRTEEN

LICENSE

Since some parts of Haver are modifications to Cuis we will use the same license as Cuis

```
MIT License

Copyright (©) Xerox Corp. 1981, 1982
Copyright (©) Apple Computer, Inc. 1985-1996
Copyright (©) Contributors to Squeak Project. 1997-2021
Copyright (©) Contributors to Cuis Smalltalk Project. 2009-2021
Copyright (©) Gerald Klix. 2020-2021.

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

Other parts like the

- *ActionBuilder*
- *SystemMorph*
- *ModulesTools*

package use Cuis, but do not modify it. Therefore these parts are under the GNU Affero General Public License. Other licenses may be obtained upon request.

CHAPTER

FOURTEEN

IMPRINT

Gerald Klix
Rheinstraße 19
79801 Hohentengen am Hochrhein

E-Mail: *haver@klix.ch*

CHAPTER

FIFTEEN

SEARCH THE DOCUMENTATION

- search