HaverOnCuis Documentation

Release 0.0.0

Gerald Klix

Apr 28, 2021

CONTENTS

Contents:

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.zip** for 64-bit Intel x86 Linux system
- **Haver-unix-linux-gnu-aarch64-32bit.zip** for Raspberry Pies with the latest 32-bit userspace and 64-bit kernel.
- **Haver-unix-linux-gnueabihf-armv7l-32bit.zip** for Raspberry Pies with an old 32-bit userspace and kernel.
- **Haver-Win32-6.2-X64-64bit.zip** for 64-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-linux-gnu-x86_64-64bit.zip
```

```
cuis
```

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

```
haver
```

Will start the original Haver image, with all updates installed[3].

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)

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

Development takes place on sourcehut:

Development https://sr.ht/~cy-de-fect/HaverOnCuis/

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

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

IRC irc://freenet.net/#Haver

IRC channel called *#Haver* at freenode. My nickname is `CyDefect`, usually I am available there.

[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.

1.5 Releases

A short sketch of the current release plan.

1.5.1 Minimum Viable Product

The minimum viable product will consist of two releases.

Alpha

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

Beta

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

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.
- Some means to distribute applications for Linux. Hopefully more than just zip files.

CHAPTER

TWO

THEMES

Haver adds two new themes to the standard themes.

A dark theme called *HaverDarkTheme*:

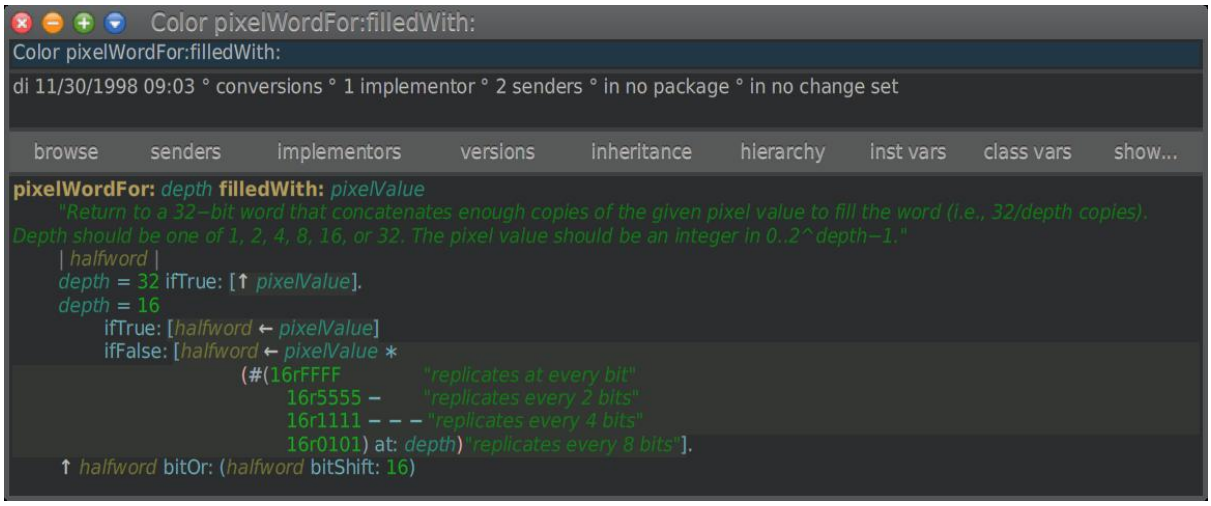


Fig. 1: Haver ‘s dark theme.

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

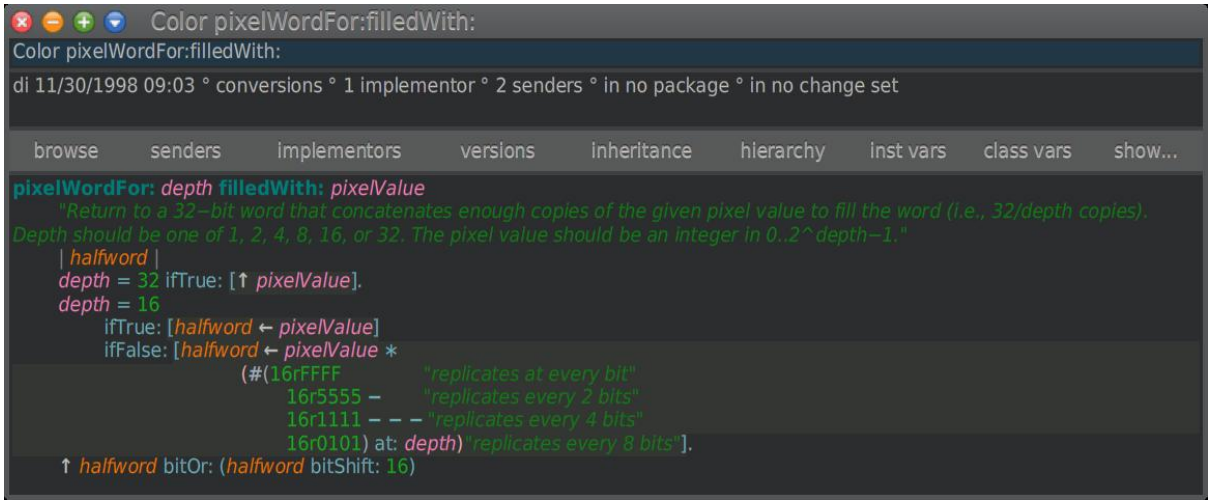


Fig. 2: Haver ‘s dark theme with distinctive syntax highlighting colors.

[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.

Themes can be selected with the *Preferences* menu – select *World* → *Preferences* → *Themes* and answer any questions with "No":

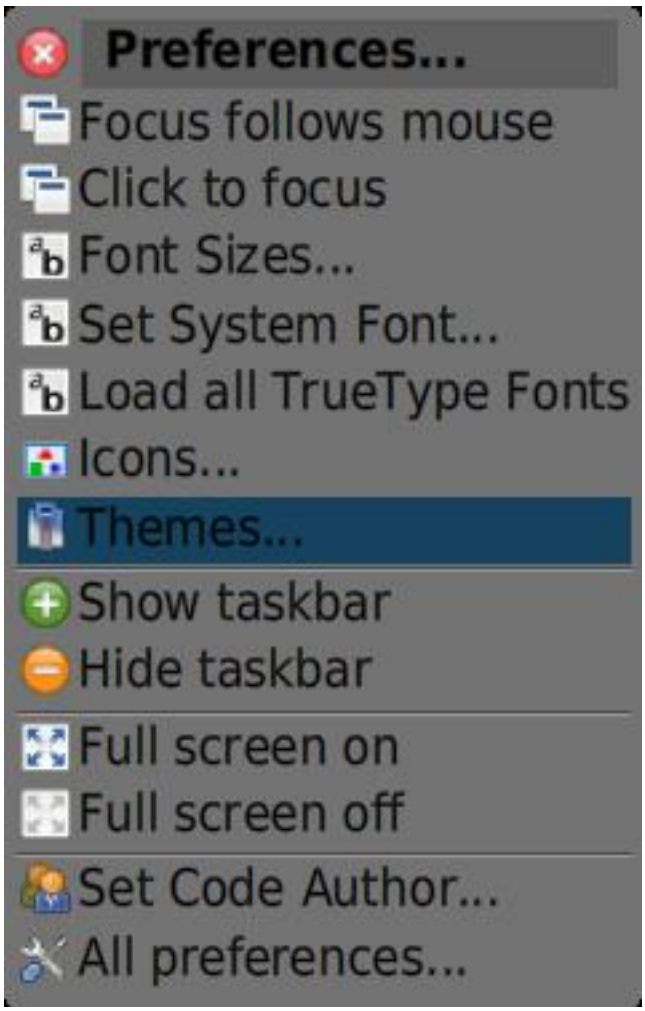


Fig. 3: Cuis' preferences menu.

CHAPTER

THREE

THE SEMANTICS OF ENVIRONMENTS AND MODULES

This section describes the semantics of modules.

3.1 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*.

3.2 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*.

3.2.1 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.

3.2.2 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.

3.2.3 Renaming

Symbols can be renamed upon export and import.

3.2.4 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.

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*:

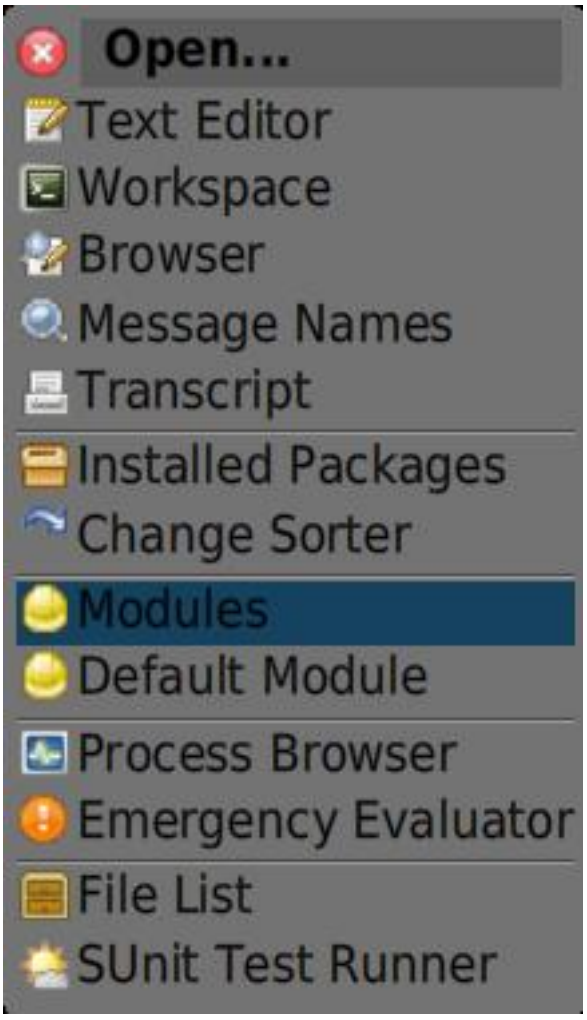


Fig. 1: Haver 's Open menu.

4.1 The Modules Browser

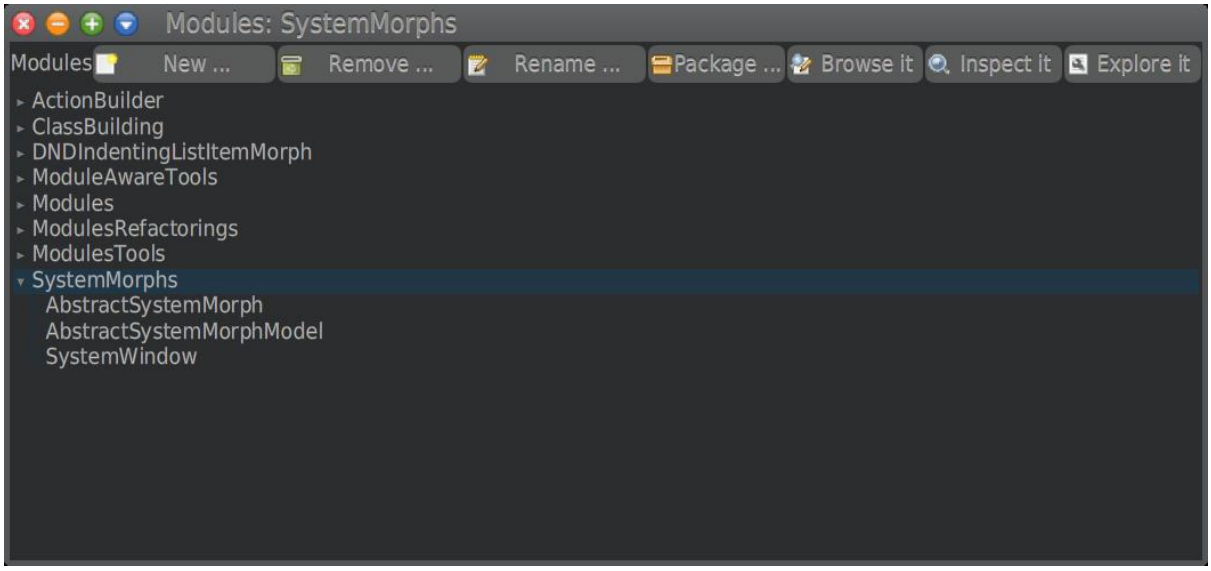


Fig. 2: The Module Browser (Video)

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

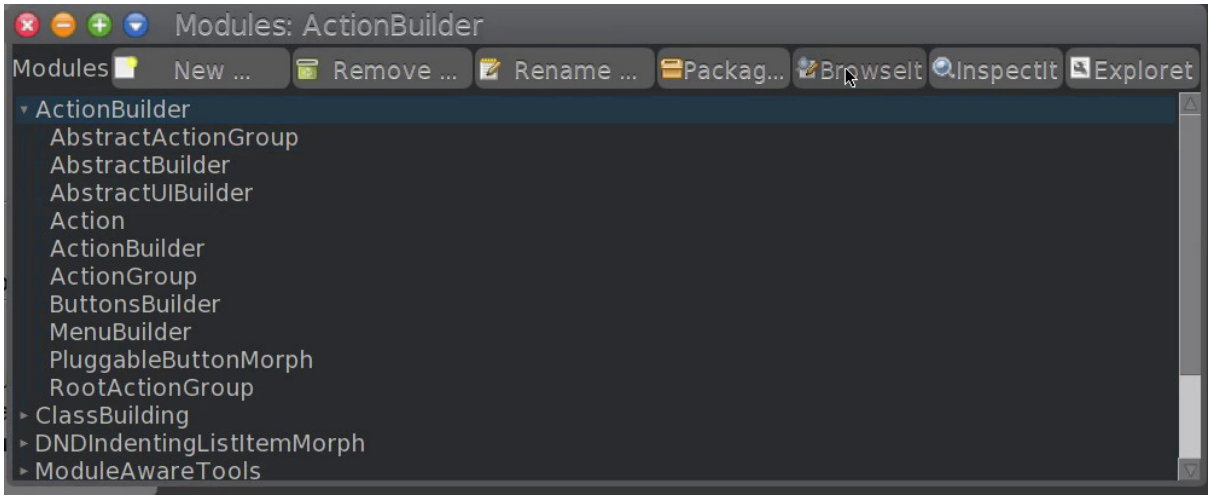


Fig. 3: Opening the Module Browser (Video)

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

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

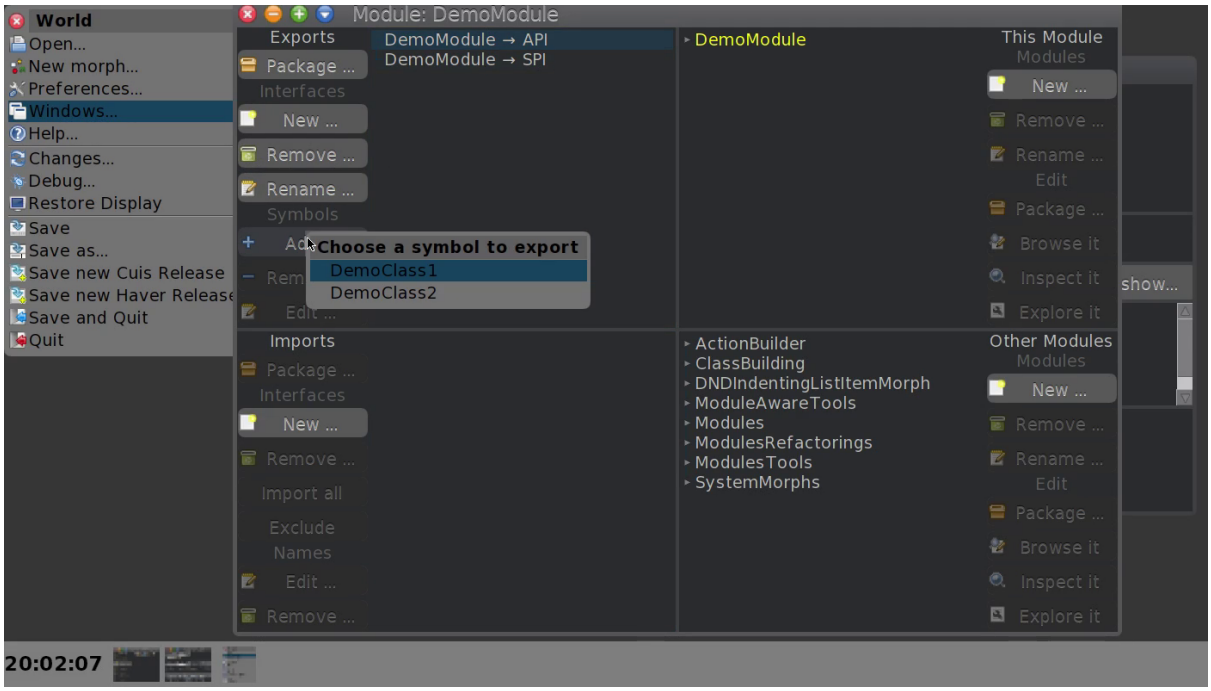


Fig. 5: The *Add* action (Video)

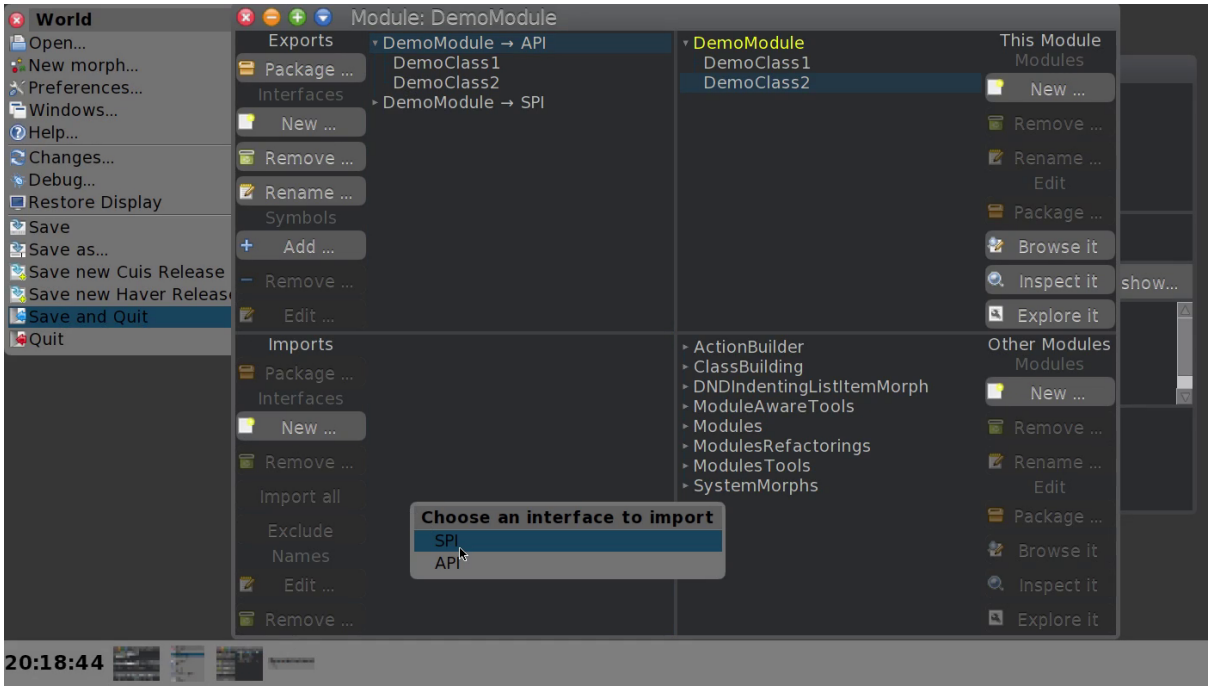


Fig. 6: Importing an interface (Video)

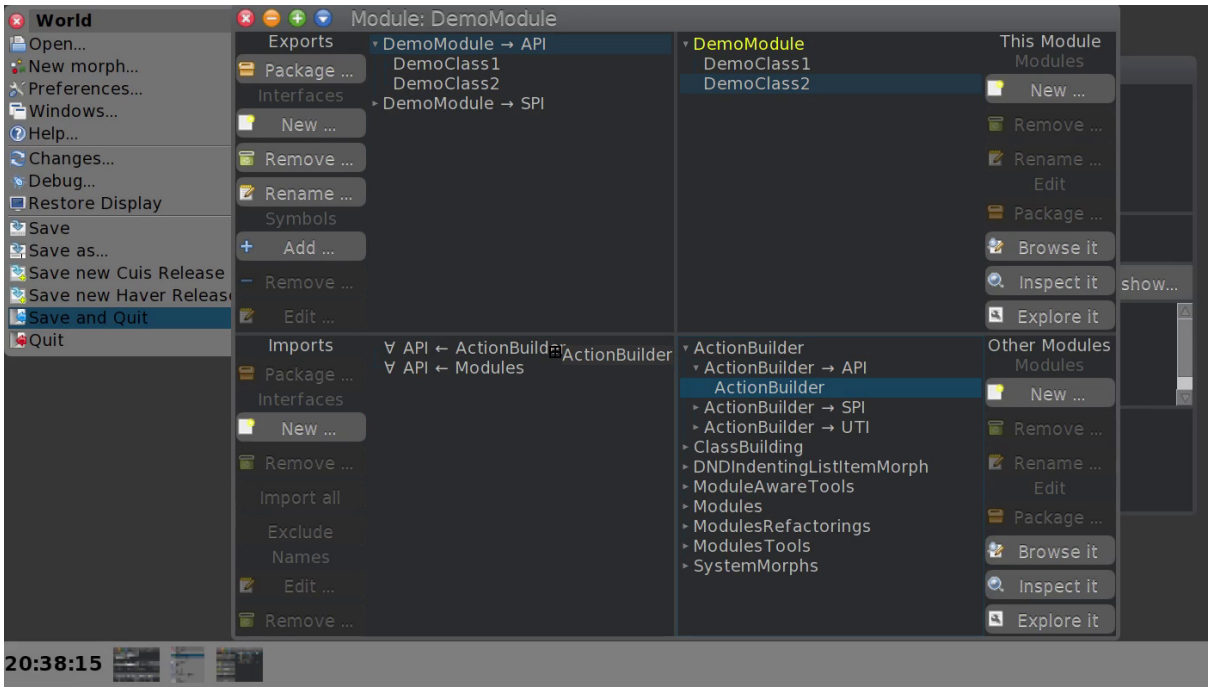


Fig. 7: Adding a specific symbol to an imported interface (Video)

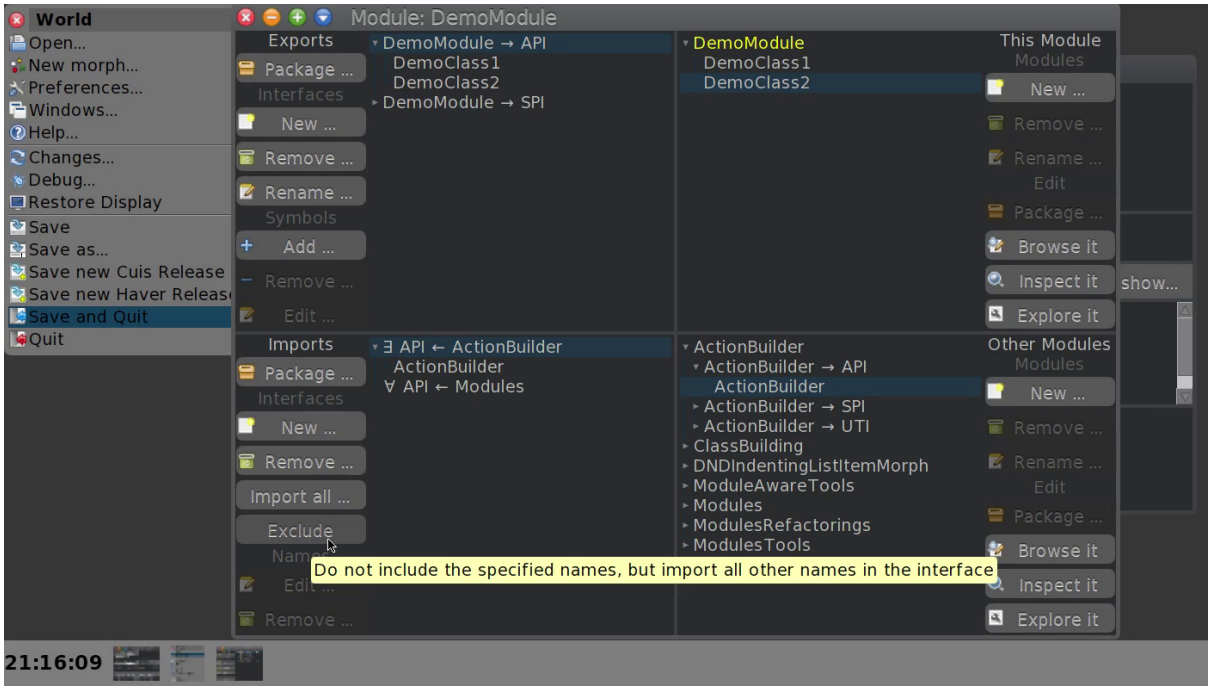


Fig. 8: Additional actions on imported symbols (Video)

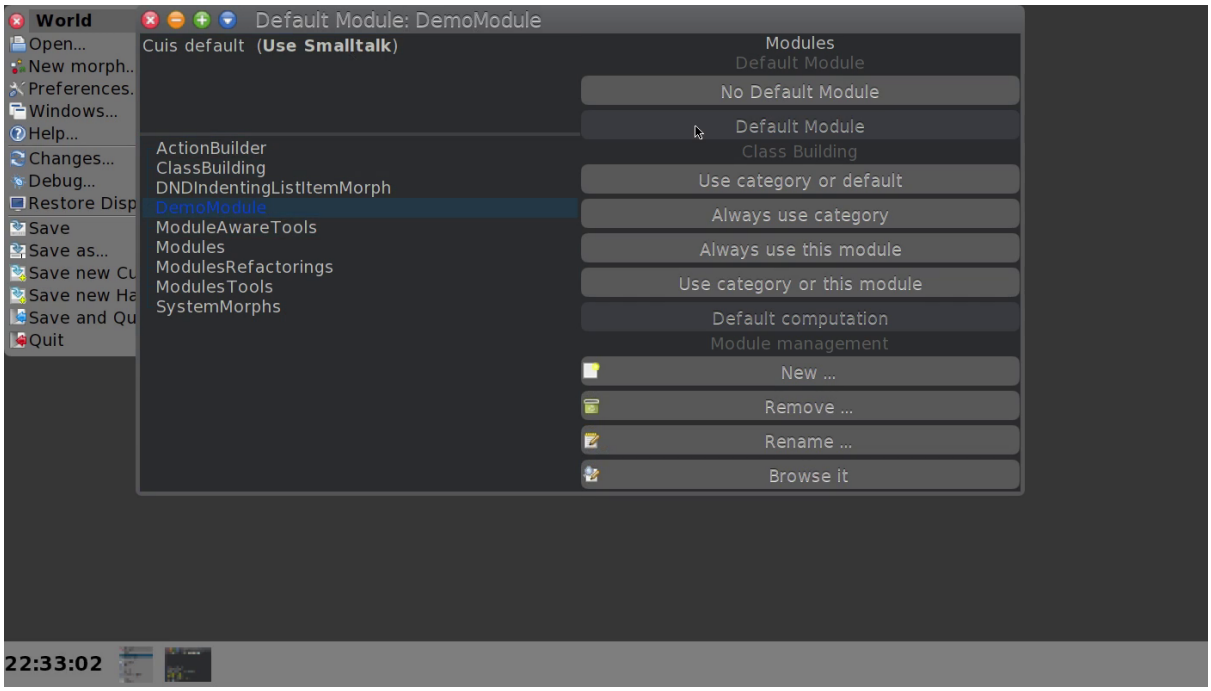


Fig. 9: A longer video explaining the Default Module Browser

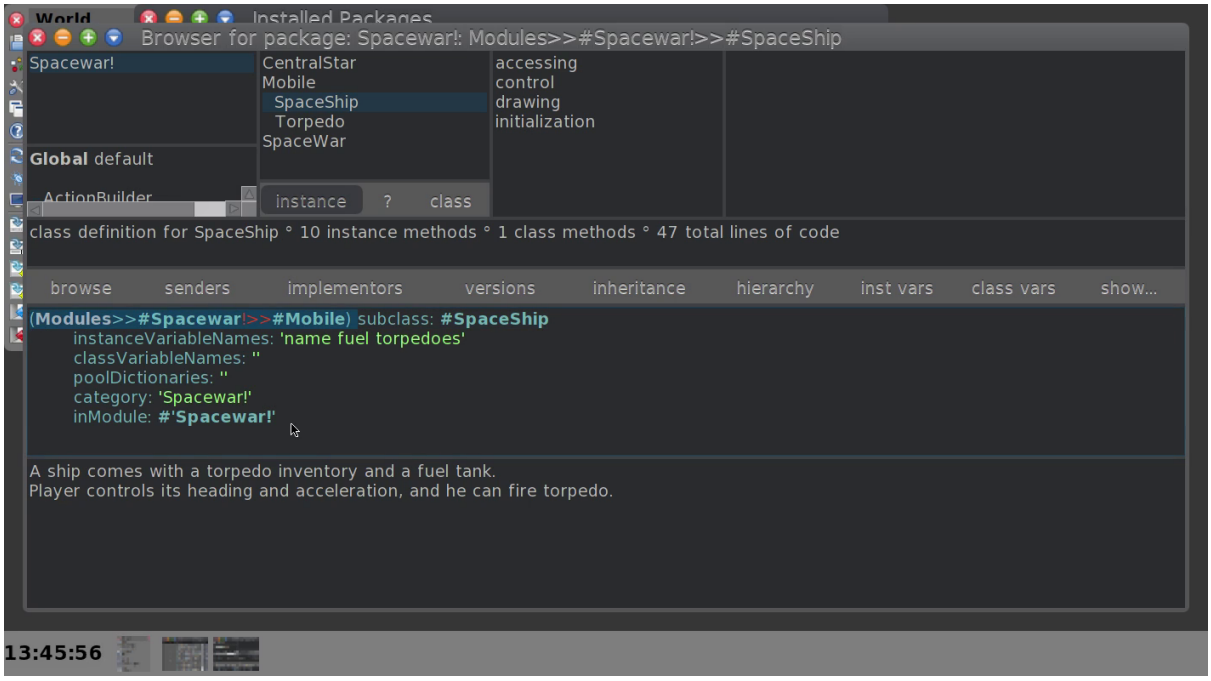


Fig. 10: Convert the Spacewar! package into a Spacewar!-module and -package (Video).

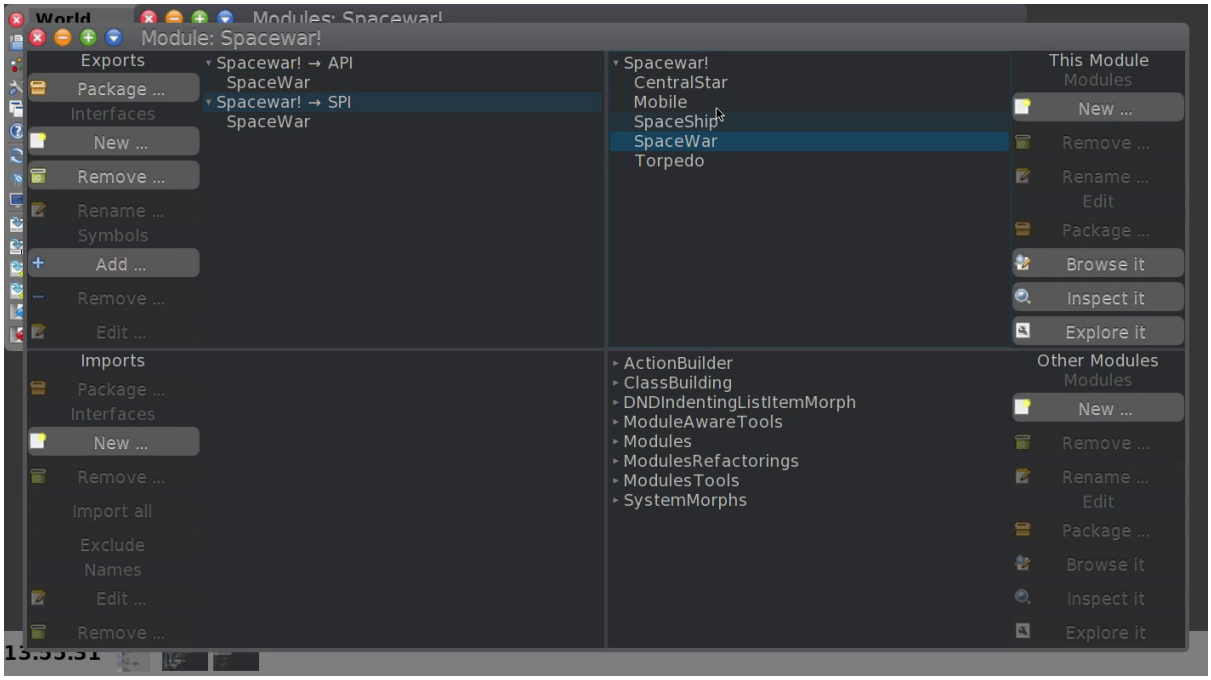


Fig. 11: Define interfaces for the Spacewar! module (Video)

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.

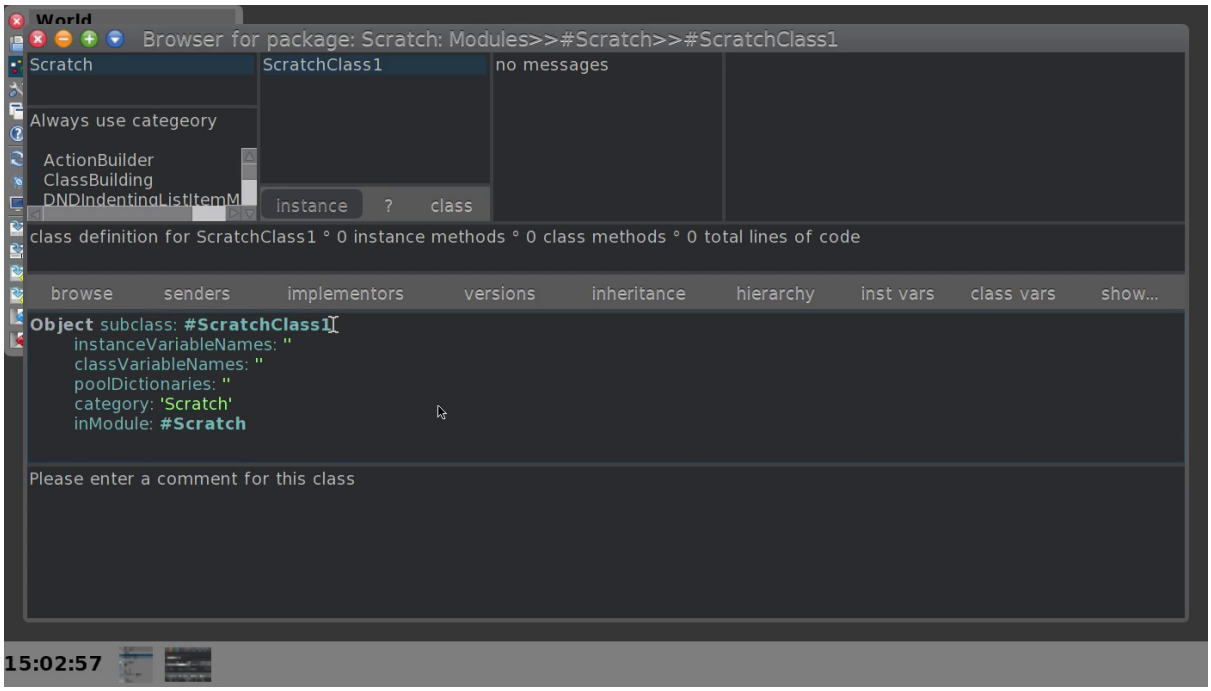


Fig. 12: Define a package, a category, a module and two classes in four simple steps.

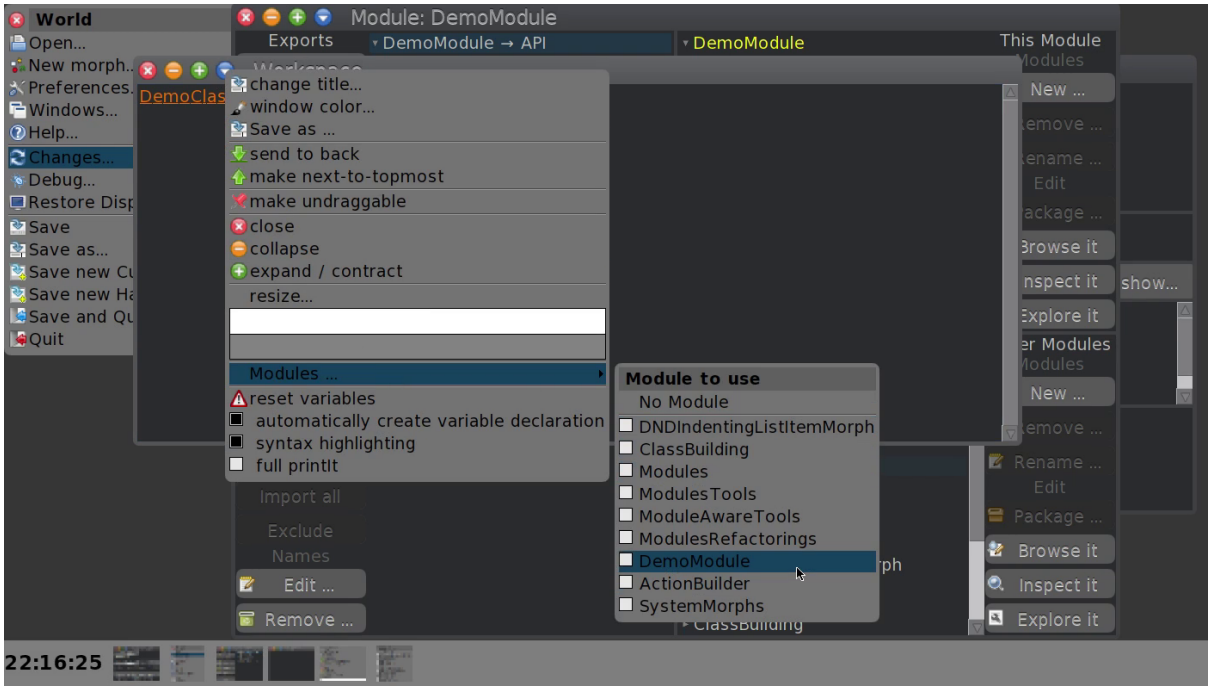


Fig. 13: The Workspace Modules Menu (Video)

CHAPTER

FIVE

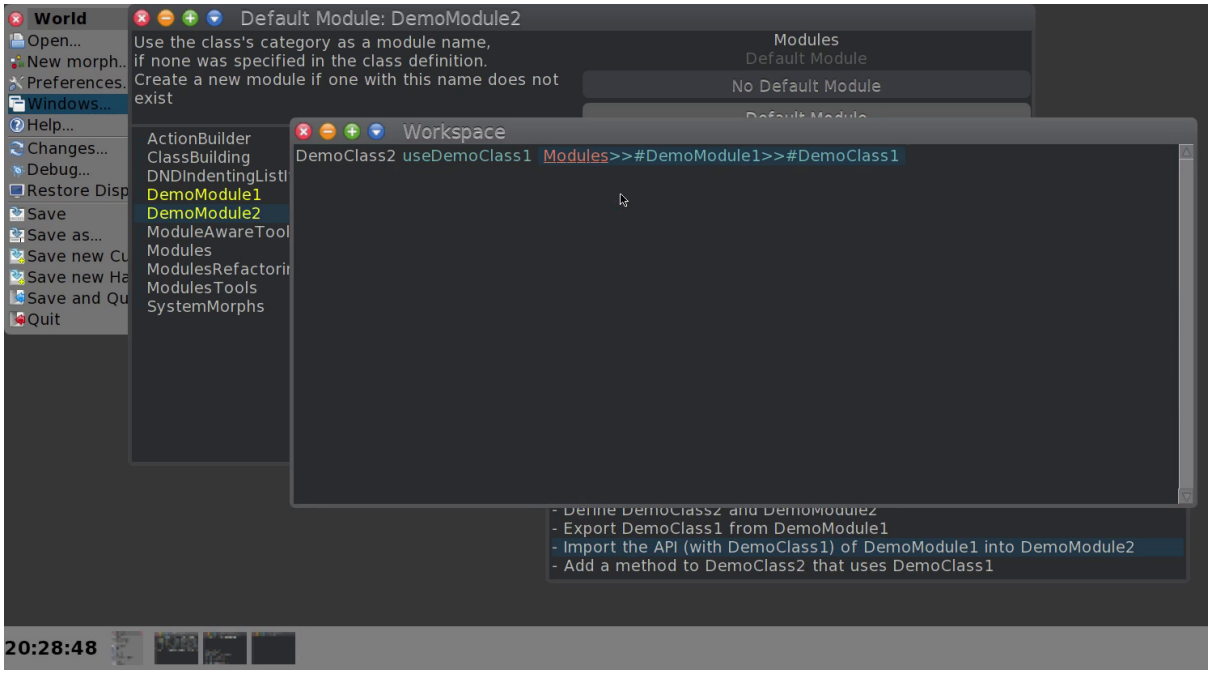
TUTORIALS

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.



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 virtual machine delivered all with distributions of Haver are a custom build ones, which include Juan's awesome VectorEnginePlugin . Their build information looks something like this:

```
'VM: 202104160938-HVR_MVP_ALPHA1 bear@speedy:gitwork/opensmalltalk-vm Date: Fri␣
↪Apr 16 11:38:38 2021 CommitHash: 0e06535be Plugins: 202104160938-HVR_MVP_ALPHA1␣
↪bear@speedy:gitwork/opensmalltalk-vm'
```

CHAPTER

EIGHT

DEFECTS

As we all know, no saftware 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

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

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.

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

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.

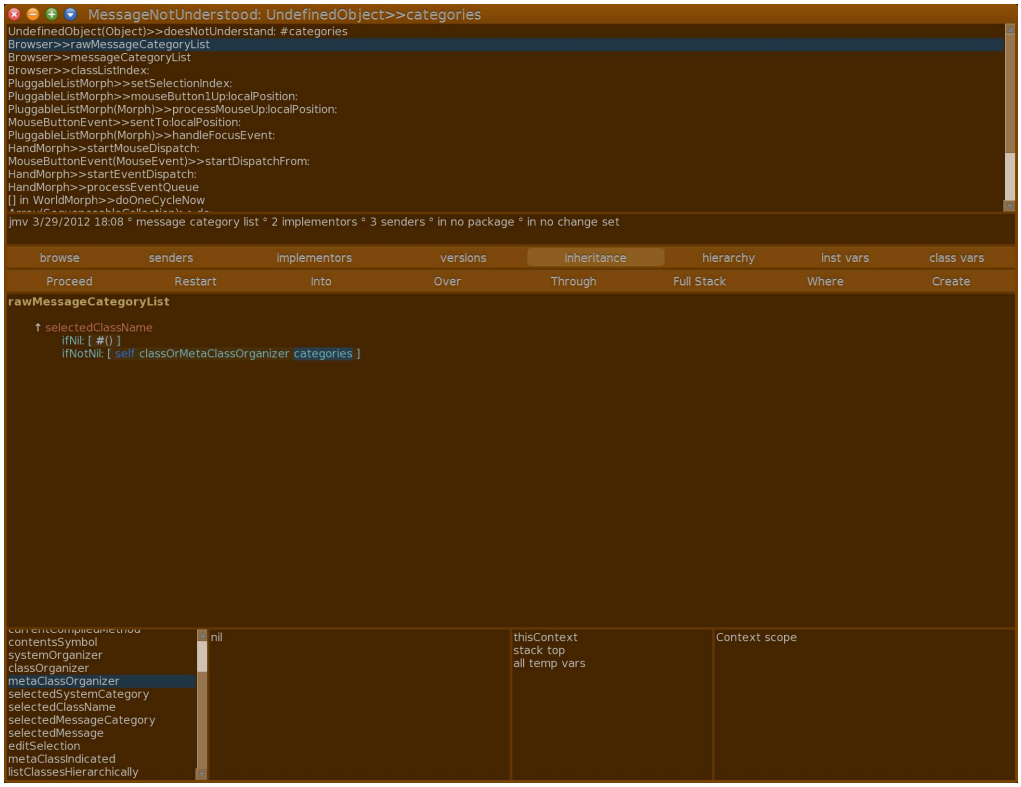
Browser>>#selectedClassName: answers nil for the global class. Using the environment-aware class list for filtering does not work. Obviously we have one case (Action) that is not part of the system organisation. If we have two classes in the same category; one global, another one in an environment, only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



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

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.

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

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

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

8.3 Bug Reporting

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

CHAPTER

NINE

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

TEN

COPYRIGHT

CHAPTER

ELEVEN

LICENSE

Since some parts of Haver are modifications to Cuis we will use the same license as Cuis:: None

> 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

TWELVE

IMPRINT

Gerald Klix
Rheinstraße 19
79801 Hohentengen am Hochrhein

E-Mail: *haver@klix.ch*

CHAPTER

THIRTEEN

SEARCH THE DOCUMENTATION

- search