Cuis Manual

January 20, 2019

Contents

1 Introduction

Introduction

This is a manual about Cuis Smalltalk.

NOTE: Most of its content was taken from Cuis documentation at:
https://github.com/Cuis-Smalltalk/Cuis-Smalltalk-Dev/tree/master/Documentation

2 Cuis Smalltalk

Cuis Smalltalk

Cuis is a free Smalltalk-80 environment originally derived from Squeak with a specific set of goals: being simple and powerful. It is also portable to any platform, fast and efficient. This means it is a great tool for running on any hardware, ranging from RasPis and the like, and phones, up to cloud servers, and everything in between, including regular laptops and PCs.

Cuis is

Small
Clean
Appropriable

Like Squeak, Cuis is also:

Open Source
Self contained
Multiplatform

Like other Smalltalk-80 environments (including Squeak, Pharo and others), Cuis is also:

A complete development environment written in itself
A pure, dynamic Object Oriented language

Cuis shares the OpenSmalltalk Virtual Machine with Squeak, Pharo and Newspeak. What sets Cuis apart from the other members of the Squeak family is the focus on Smalltalk-80 and an active attitude towards system complexity:

Unbound complexity growth, together with development strategies focused only in the short term, are the worst long term enemies of all software systems. As systems grow older, they usually become more complex. New features are added as layers on top of whatever is below, sometimes without really understanding it, and almost always without modifying it. Complexity and size grow without control. Evolution slows down. Understanding the system becomes harder every day. Bugs are harder to fix. Codebases become huge for no clear reason. At some point, the system can't evolve anymore and becomes "legacy code".

Complexity puts a limit to the level of understanding of the system a person might reach, and therefore limits the things that can be done with it. Dan Ingalls says all this in "Design Principles Behind Smalltalk". Even if you have already done so, please go and read it again!

This presentation by Rich Hickey, "Simple made Easy" is also an excellent reflection on these values.

We follow a set of ideas that started with Jean Piaget's Constructivism, and were further developed in Seymour Papert's Mathland. These lead to Alan Kay's Learning Research Group's Personal Computer for Children of All Ages, Personal Dynamic Media, i.e. the Dynabook and to Smalltalk-80. To us, a Smalltalk system is a Dynabook. A place to experiment and learn, and a medium to express and register the knlowledge we acquire. We understand software development as the activity of learning and documenting knowledge, for us and others to use, and also to be run on a computer. The fact that the computer run is useful, is a

consequence of the knowldege being sound and relevant. (Just making it work is **not** the important part!)

Cuis Smalltalk is our attempt at this challenge. Furthermore, we believe we are doing something else that no other Smalltalk, commercial or open source, does. We attempt to give the true Smalltalk-80 experience, and keep Smalltalk-80 not as legacy software historic significance, but as a live, evolving system. We feel we are the keepers of the Smalltalk-80 heritage, and enablers of the Dynabook experience.

Cuis is continuously evolving towards simplicity. Each release is better (i.e. simpler) than the previous one. At the same time, features are enhanced, and any reported bugs fixed. We also adopt recent enhancements from Squeak and share our work with the wider Squeak and Smalltalk community.

2.1 About the name Cuis

About the name Cuis

Cuis is the common name of a small animal that lives in Argentina's countryside. Cuis Smalltalk was originally forked from Squeak Smalltalk and many of us are also active in the Squeak community. So, picking the onomatopoeia of the voice of a mouse for the name makes sense. As the project was started in Buenos Aires, 'Cuis' (essentially 'Squeak' in Rioplatense Spanish) was the obvious choice.

2.2 Learning about Cuis Smalltalk

Learning about Cuis Smalltalk

If you are learning Smalltalk, the Cuis community can help you. Check the "Learning Cuis Smalltalk" repository. It includes several great tutorials. Also, the TerseGuide.pck.st package (in the /Packages folder in this repo) is useful both as a guide and a reference.

Additionally, there are many tutorials and references for Smalltalk in the web. They apply quite well to Cuis, especially those written originally for Smalltalk-80 or Squeak. These books "Smalltalk-80 the language and its implementation" and "Inside Smalltalk volume I" are great introductory texts, and they are also the reference for the language and basic class library. Both are freely available.

The user interface enables you to access most of the code and conduct Smalltalk experiments on your own. You can review its features at "Quick Tour of the UI".

2.3 Contributing to Cuis

Contributing to Cuis

Cuis is maintained on `https://github.com/Cuis-Smalltalk/Cuis-Smalltalk-Dev`. The main meeting point for Cuis users and developers is the mail list `http://cuis-smalltalk.org/mailman/listinfo/cuis-dev_cuis-smalltalk`.org . You can browswe the archives for a glimpse of our discussions.

If you want to contribute code to the project, please read `ManagingyourcodeinCuis`, about developing packages for Cuis, and `CuisandGithub`. While Cuis should work equally well with any file-based DVCS, we

encourage the use of Git and GitHub.

In any case, we also accept contributions as ChangeSet files in email. Any contribution must be under the MIT license.

To contribute code, please use an image with all included packages already loaded, using updated versions, especially, of any affected packages. This will ensure we don't break them while we evolve Cuis.
Here is a Smalltalk script to load all packages currently included:

```
Feature require: 'Core-Packages'
```

2.4 License

Cuis License

Cuis is distributed subject to the MIT License. See the LICENSE file. Any contribution submitted for incorporation into or for distribution with Cuis shall be presumed subject to the same license.

3 Managing your code in Cuis

Managing your code in Cuis

Cuis includes tools and procedures for managing Smalltalk code. Code that is not part of the Cuis Core image itself, like applications, frameworks and libraries, should be stored in Packages. New code that are meant as patches, fixes or additions; that could eventually become part of Cuis itself, is not part of any *Package*, and is therefore automatically stored in ChangeSets.

3.1 Packages

Packages

Let's start with *Packages* . The Package implementation in Cuis is based on PackageInfo, the standard way to specify packages in Squeak and its derivatives, and used, for example, by Monticello. It uses Package names to specify prefixes for Class and Method categories. Classes and Methods whose categories match a Package's prefixes belong in that Package. More details about how PackageInfo decides what code belongs in a package are available at `http://wiki.squeak.org/squeak/3329` .

To install packages *(.pck.st files)* in Cuis, use the

FileListWindow openFileList

, navigate to the appropriate directory (on disk, or in a GitHub repository, etc), select the package file and click on [**Install Package**].

Cuis includes a tool to manage installed *Packages*. It is at

CodePackageListWindow openPackageList

. To create a new package (instead of installing an existing one from a file), click on [**Create Package**] This creates a new package, and associates with it all the existing code in the image that matches the package name.

The operations available on installed or newly created packages are:

[**Save**] Saves a package on the file system. Overwrites any existing version. It is good to save the package from time to time, to reduce the risk of losing code.

[**Delete**] Removes the Package instance from the image. Does not remove any code. This means, effectively, to merge back the code into Cuis.

[**Browse unsaved Changes**] This opens a ChangeSorter on the ChangeSet that captures all the changes done to the Package since it was last saved. Therefore it shows the work done on the package that would be lost if the package is not saved.

[**Browse package code**] This opens a Class Browser that only shows the code that belongs in the package.

This is useful for working on a package, or studying it.

[**Add requirement**] This opens a select list of loaded packages. Each package provides a *Feature*. You can CANCEL, require the current Cuis base version (at a minimum) or require any of the packages on the list. Required packages will be loaded before the selected package (

```
Feature require: #'your-package'.
```

). When a package is selected, the lower browser pane shows its requirents, which may be deleted. Don't forget to *Save* your package after adding or deleting requirements!

The tool shows, for each Package, the name, whether it is dirty (has unsaved changes) and the file it was installed from / saved to.

Handling Packages like this, Cuis behaves as a sort of document editor (like, for example a regular text editor) whose documents are *Package* files *(.pck.st)*. Cuis doesn't handle Package versions, ancestries, etc. If versioning of Packages is desired, the best is to use a versioning file repository, such as Git or Mercurial. The recommendation is to use a GitHub repository with a name beginning with 'Cuis-Smalltalk-', so it will be easy for anybody to find it. Cuis *Package* files are uncompressed, use Lf (ASCII 10) as newLine, and are encoded in ISO 8859-15. This means they are Git friendly, and Git/GitHub can diff and merge them, and browse them with syntax highlighting.

This is not unlike using Git or GitHub with a file-based development environment such as Eclipse or a text editor. Like Cuis, these tools don't do version handling themselves, they just load and save files; and let Git do its magic.

3.2 Changes to the Cuis base image

Changes to the Cuis base image

The way *ChangeSets* are created and managed in Cuis is different from Squeak. This was done to make ChangeSets a good way to manage changes to the base Cuis Core image, while keeping code in Packages out of the way, so they don't get mixed together.

What is not in a Package belongs (at least temporarily) to the Cuis Core image. Such code is automatically captured in a *ChangeSet*. The ChangeSet for Core changes is created automatically and named like *"1243-CuisCore-JuanVuletich-2012Apr03-22h50m"*. The number at the beginning is the next number for the Cuis update stream, and is provided only as a suggestion. The "CuisCore" part is to reveal that the code belongs in the base image and not in some package. Then we have author name and date / time of creation. These *ChangeSets* are created automatically. There is no longer a way to manually create them, or make them "current" or "active". It is best to rename them, replacing *'CuisCore'* with some meaningful name. These *ChangeSets* will not capture any code that belongs in a Package.

Opening a

ChangeSorterWindow openChangeSorter

will show the CuisCore change set. This is useful, for example, to check that no code that was intended for a Package ends here by mistake (because of the wrong class or method category). But it is also useful when doing changes to the base system. Now, we can do changes both to the base system and to a number of packages, all in the same session, without having to be careful about selecting the proper change set before saving a method: The code is automatically added to the proper *Package* or *ChangeSet*, simply following the class or method category. Gone are the days of messed up change sets and lost code!

When the changes to the base system are complete, it is a good time to review the CuisCore change set and, maybe remove from it changes that we don't want to keep (for example, experiments, halts, etc). Then, just do right click / File out and remove. This saves the *ChangeSet* on disk. It also removes it from the **ChangeSorter** (but it doesn't remove any code). This is good, because the next changes done will end in a new CuisCore change set, and there's no risk of having undesired changes in the old one. As changes to the base image progress, and several CuisCore *ChangeSets* are saved to disk, these numbered files are created in sequence. They will be ready to be loaded back in proper order in a fresh Cuis image, or to be sent to Cuis maintainers for integration in the update stream and in next releases of Cuis.

3.3 Loading ChangeSet files into Cuis

Loading ChangeSet files into Cuis

There are two ways to load *ChangeSet* files (.cs): [**FileIn**] and [**Install**].

[**FileIn**] loads the code without creating a new *ChangeSet* object. This means that changes that belong in the base image (and not in a package) will be added to the current ChangeSet for Cuis core changes, as if they were done by the user. This is appropriate when we are combining code from more than one source into a single ChangeSet. Any change that belongs in an installed package will be added to it, and the package will appear as dirty.

[**Install**] loads the code into a separate *ChangeSet* object (viewable in the

`ChangeSorterWindow` `openChangeSorter`

This is appropriate for loading Cuis updates, or other code that we are not authoring, as it doesn't add new items (class or method definitions) to the current ChangeSet for our changes to Cuis. Usually any ChangeSets should be installed before doing changes to the image. The reason is that an installed ChangeSet could overwrite changes done by you, or packages you have installed. If this is the case, the affected packages would appear as dirty, and your change set would include any installed changes (that don't belong in a package). Be careful when saving packages or change sets if this was the case!

4 Additional Packages for Cuis

Additional Packages for Cuis

The Cuis base image includes only kernel functionailty, very basic libraries, and development tools. Optional functionality, that can be loaded as needed, is stored in separate code Packages. The Cuis community developes and maintains several dozens of such Packages.

Some of them are included in the main Cuis GitHub repository, at https://github.com/Cuis-Smalltalk/Cuis-Smalltalk-Dev in the 'Packages' folder. Loading 'Core-Packages.pck.st' loads them all:

```
Feature require: 'Core-Packages'
```

5 Cuis and Github

Using Git and GitHub to host and manage Cuis code

Cuis includes tools and procedures for managing Smalltalk code. Central to this is the management of Packages and Package Files (.pck). But Cuis doesn't do version control. Instead, we suggest using external VCS tools. In particular, we're using GitHub, and the first project we're hosting there is StyledTextEditor.

The guiding principle is to **not duplicate concepts and behavior**. As we're using an external tool (Git) for version control, then we use it as it meant to be used. Most people use Git for version control and a file based IDE such as Eclipse for development. Such IDEs don't do version control themselves. It is done by Git. Do the same: do not include package version control in Cuis. This is a departure from the Monticello /Git integration (smallsource and MonticelloFileTree) by Otto Behrens, Dale Henrichs, etc.

We use GitHub to host, version, diff and merge external packages (.pck files), i.e. code that is maintained independently and outside Cuis.

Package files need to be simple text files. Cuis encoding for latin alphabet (ISO 8859-15) is handled without problems by GitHub. Cuis uses the LF (ascii code 10) newline convention, as preferred in GitHub. This allows Git/GitHub to diff versions, and merge branches.

Each GitHub repository has one set of users and permissions. Each GitHub repository has one state (Git commits repositories, not individual files). Branch and merges are done on the whole repository and not on individual files. Therefore, we need a separate GitHub repository for each project, i.e., for each package or set of closely related packages that are always loaded and maintained together as a whole.

Development process for External Packages

This is the suggested procedure for developing external packages. Usually do this every day.

Start with a standard (i.e. fresh) Cuis image. Never save the image.

Set up Git repositories for external packages (if not already done)

Install packages from Git repositories.

Develop. Modify and/or create packages.

Save own packages (to Git repositories).

Git add / commit / push as appropriate.

Fileout changes that are not part of any package. These are automatically captured in numbered changesets, separated from changes to packages.

Exit the image. Usually without saving.

6 Index