



# SourceIDE: A Semi-live Cross-development IDE for Cola

Scott Wallace

This material is based upon work supported in part by the National Science Foundation under Grant No. 0639876. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

VPRI Research Note RN-2008-002

## SourceIDE: A Semi-live Cross-development IDE for Cola

by Scott Wallace     2 September 2008

This material is based upon work supported by the National Science Foundation under Grant No. 0639876.

This paper describes the first phase of a long-term effort that will culminate in the availability of native, incremental, “live,” interactive development environments for the “Cola” family of languages.

However, a native Cola IDE sufficiently capable that it can attract real users away from their familiar, highly-evolved IDE’s, replete with feature-rich text-editors, windowing systems, etc., such as Emacs and Smalltalk, is probably still a long way off.

In the meantime, cross-development will continue to be necessary. Currently, most Cola development is done using the Unix command shell as the development environment and using Emacs as the code editor -- basically a C-style batch development process, in which the programmer edits plain-text source files in a text-editor, uses full-text search as his main query tool, assembles code in trees of directories and source files, submits compile requests from the command line, manages everything via “make” files, and so forth.

For some kinds of Cola development (such as constructing user interfaces,) and at least for some kinds of users (including the current writer,) the availability of much higher-level development-support tools would make a large, indeed decisive, difference in Cola empowerment. But this is a chicken-and-egg situation. Developing a high-quality, feature-rich, user-friendly, interactive development environment is, at the best of times, very difficult. Few tasks would benefit more from being developed within an already-existing very rich, live IDE than developing an IDE.

The work described here, called “SourceIDE” (i.e., an IDE for developing code resident in external source files,) done between November 2007 and January 2008, explores one possible approach to breaking out of the chicken-and-egg deadlock.

The primary intention of the SourceIDE is to bring many "live environment" features, traditionally only found in self-hosting dynamic environments such as Smalltalk, into a source-code editing environment intended for use with external source files organized into a source tree.

In Smalltalk, the code tools operate on the actual code base of the living system. In contrast, in the SourceIDE, the code tools operate on an external code base as described in external source files. Within the constraint of that difference, every effort is made in the SourceIDE to enable and encourage the same dynamic-query-based, development style so successfully used in Smalltalk.

By "source tree" here we mean an interlocking set of ascii source-code files, linked by compiler directives such as "\$include <filename>" and "\$import <filename>," distributed across a number of directories. Hundreds of files, in dozens of directories, may be involved in a single tree.

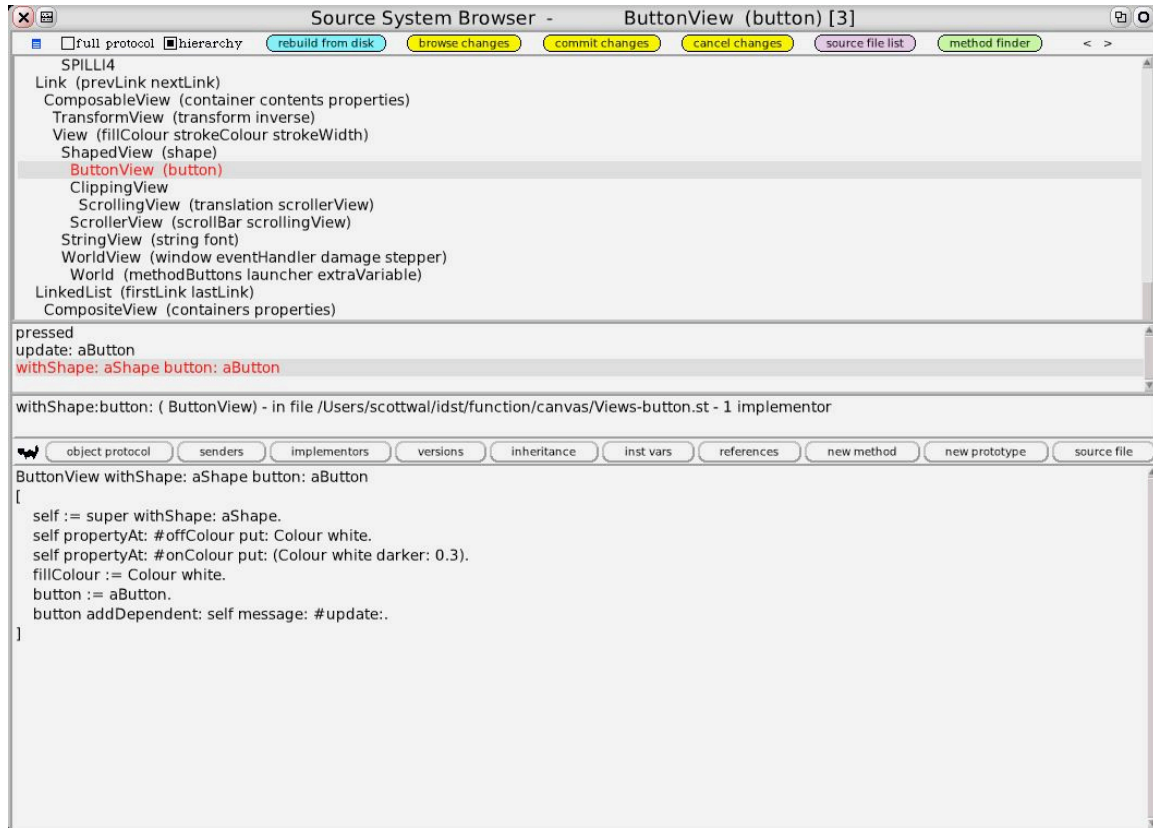
In the SourceIDE, we provide a Squeak-hosted "semi-live environment" for viewing, traversing, querying, and editing the code represented by a given source-tree. The IDE constructs an internal "mental-map" of all the objects, methods, and other structures defined by a source-tree, and uses that map to provide various views of the system, embodied in a bevy of query tools to assist in comprehending and traversing the code, and, pervasively, a source-code editing environment that includes:

- \* Ability to view and assess the system at any point along the source-editing development line.
- \* Ability to "commit" an arbitrary set of changes back to disk.
- \* Ability to browse and edit changes made since the latest "commit".
- \* Ability to view and edit code both as individual units (e.g. methods) and on an entire-file basis.
- \* Selective roll-back both at the individual method or class level, and at the entire-source-file level.
- \* Ability to pursue chains of enquiry (senders, implementors, references, and the like) at any point in the development process.

The seven principal tools provided by the Winter 2007-8 version of the SourceIDE are:

## 1. Source System Browser

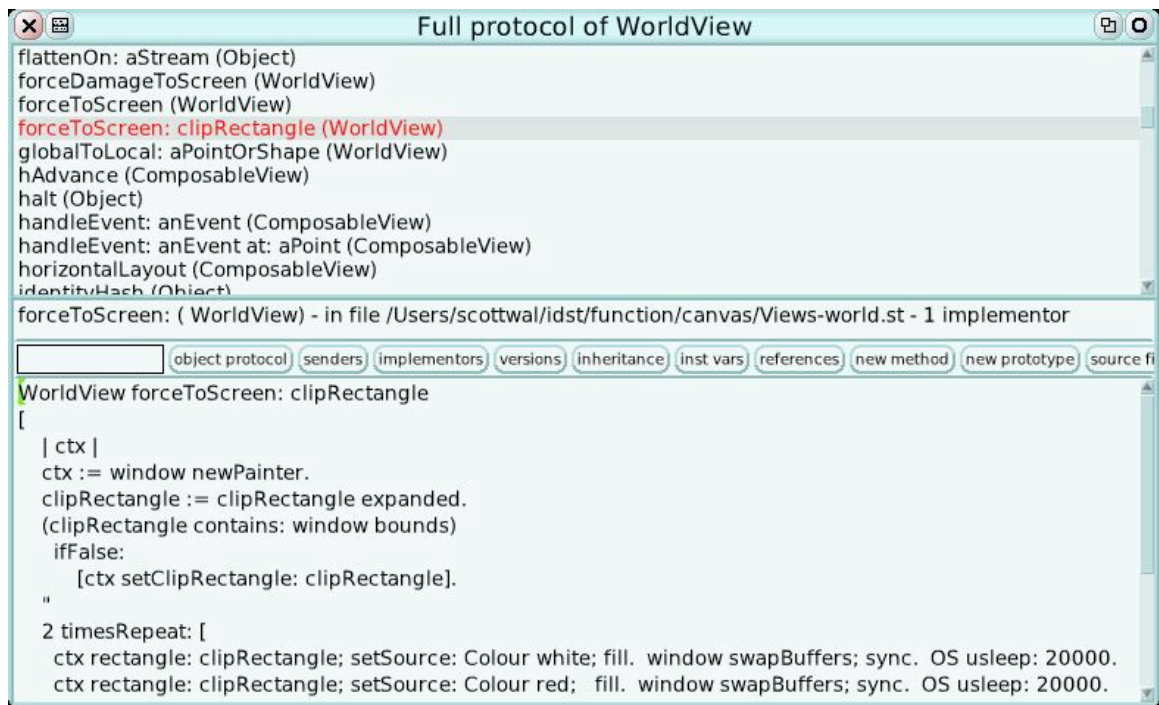
This serves as the nerve center for access to all the tools in the IDE.



For a given source tree, the “System Browser” provides access to all the object type-definitions, and all the code of each type represented. Checkboxes at top allow the alternatives of viewing by hierarchy (as illustrated below) or viewing alphabetically by type-name, and also the alternatives of listing under each type only the methods explicitly defined by it or listing its "full protocol" (including inherited methods.) An "annotation pane" provides useful collateral information about the currently-selected method, and various buttons and menu-items in the tool allow for numerous queries to be undertaken.

In the above example, method #withShape:button: of the prototype named ButtonView is selected; its source code is seen, and can be edited, in the bottom pane. The buttons in a row above the code pane offer access to a variety of other tools which pertain to the selected prototype and method.

## 2. Individual Object Lexicon

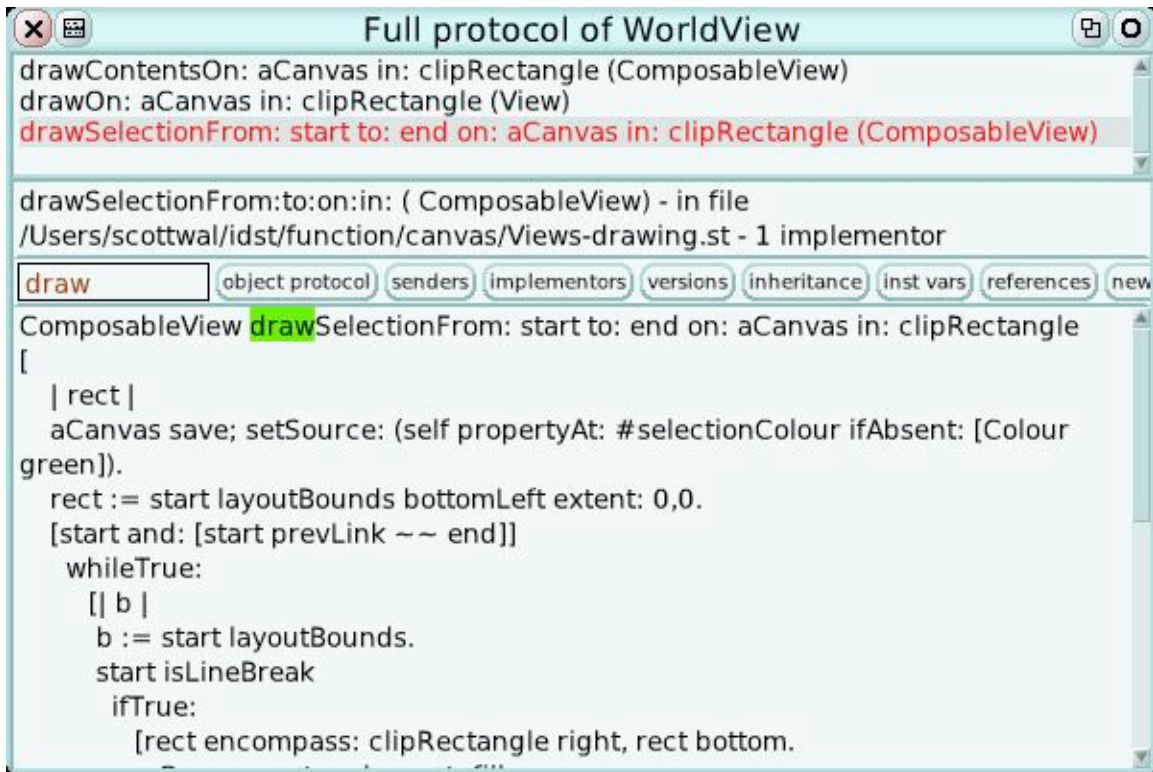


This is a comprehensive look inside an object, showing all its code (including inherited methods,) all of which can be edited within the tool.

The example above shows a Lexicon open on a WorldView object. WorldView inherits from ComposableView, which inherits from Object, and one can see some methods of all these prototypes in the example.

The centerpiece of this tool is its "search" capability, allowing the user to find methods, within the protocol of a given object, by name / keyword.

In the example below, the user has typed "draw" into the Search pane, and in consequence only the three methods in the object's protocol whose selectors contain the substring "draw" are displayed:



### 3. Message-Set list

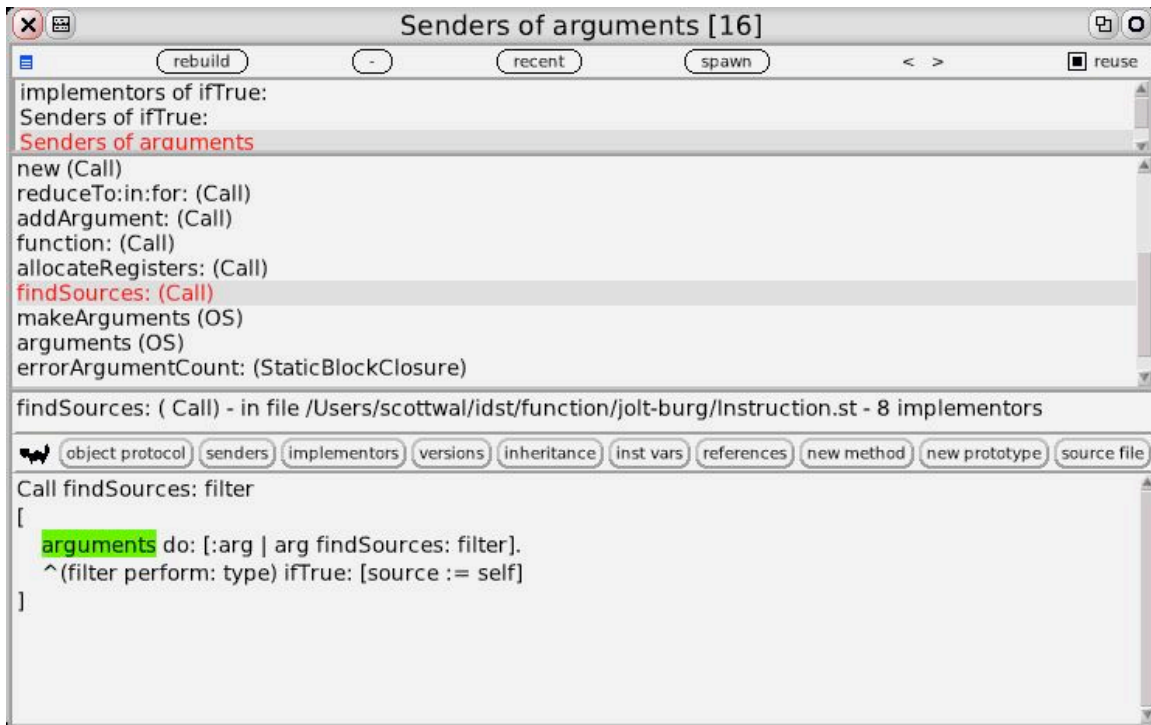
A pervasive characteristic of effective code development in live IDE's is that most of the plausible queries that the programmer wishes to make during development can be posed from within the tool currently being used, and (ideally, in many cases at least) the results of most such queries can be viewed within the tool from which the query was launched, thus holding down what might otherwise be wild proliferation of windows.

Some of the queries that can be invoked at any point from within the SourceIDE are

- browse all methods that reference a given instance variable of a given object type.
- browse all methods that implement a given message.

- browse all methods that send a given message.
- browse all methods that reference a given global.
- browse all methods whose selectors contain a given string pattern.
- browse all methods that reference a given literal.
- browse the inheritance hierarchy of a given method.
- browse all methods that have been changed since the last commit.

Most such queries can be initiated via buttons in the control panes of the various tools; others are available only from menus. In any case, most queries result in results being displayed in a highly flexible tool called a “Message Set List”:

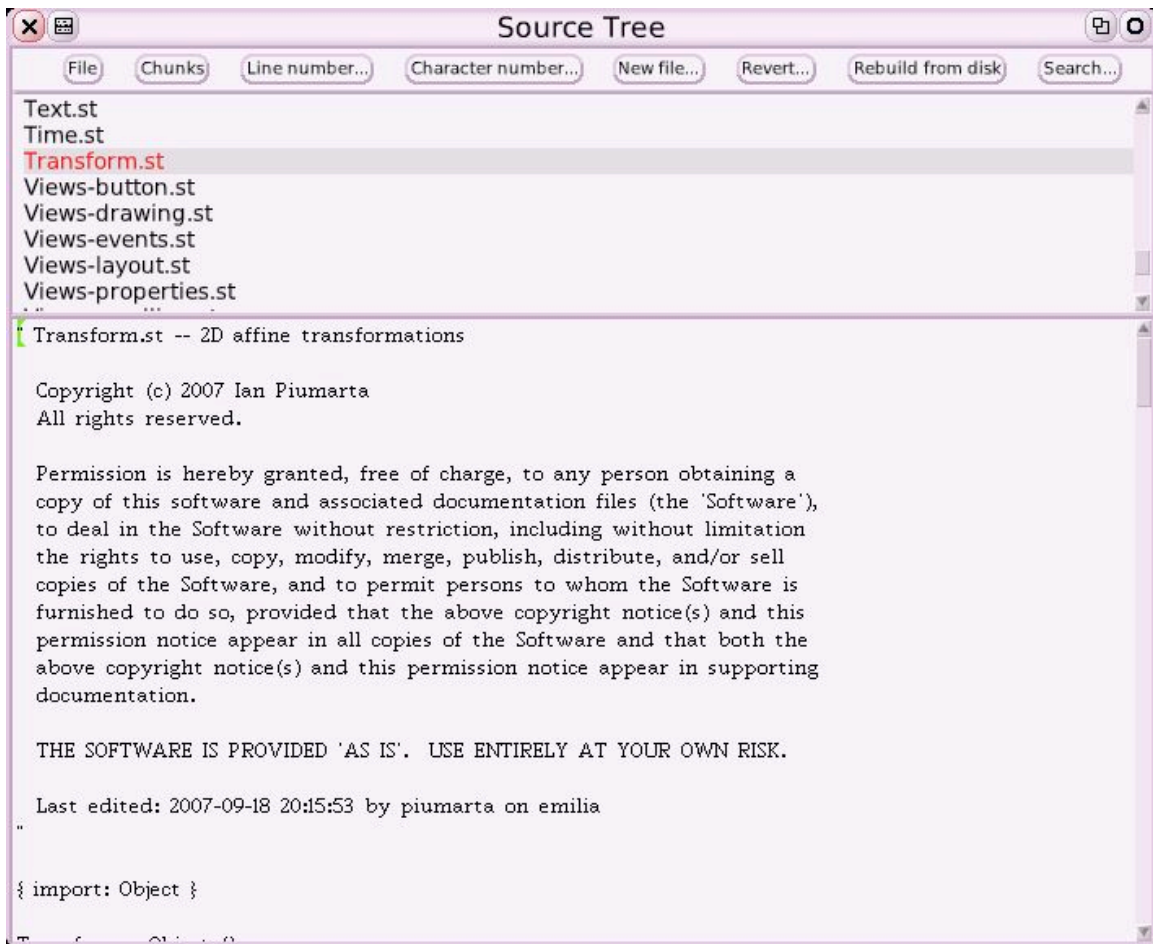


Message-set lists provide a uniform, compact mechanism for browsing the results of a series of (usually related) queries within the bounds of the same tool. Thus, for example, in the above figure the results of three different queries are all seen in the tool.

The < and > buttons in the control panel at the top of the tool allow the user to retrace his browsing history within the tool, and the "reuse" checkbox allows the user to decide whether further queries should be serviced within the tool or whether new windows should

be used for them. Unless “reuse” is turned off, subsequent queries from anywhere within the tool will result in new message-sets being added to the same tool, thus reducing the use of windows overall.

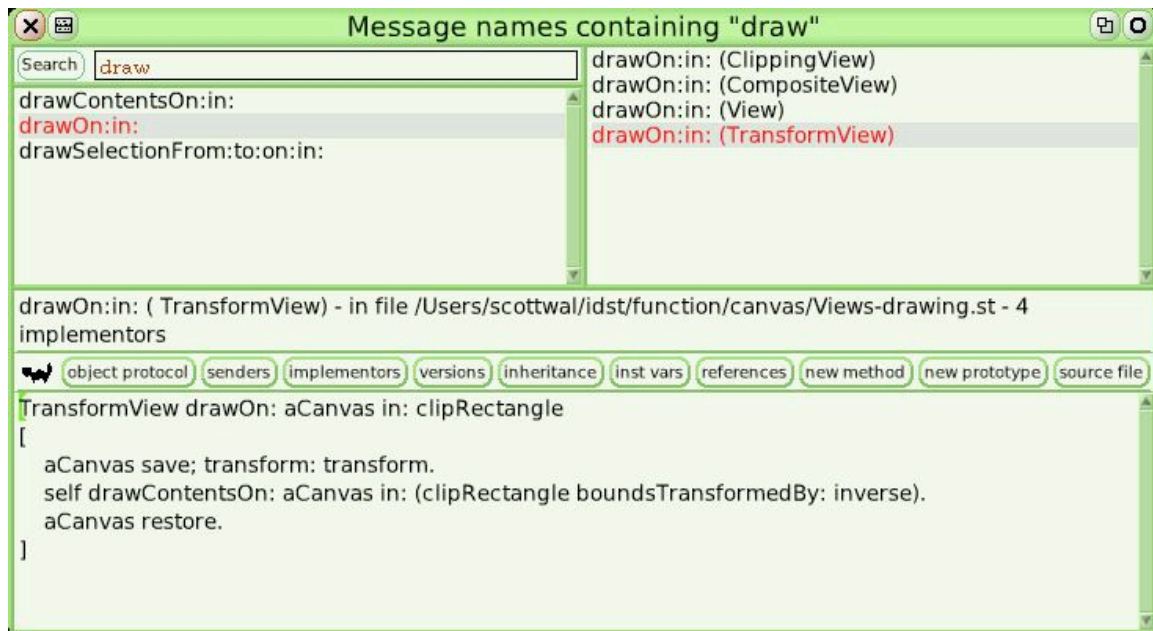
#### 4. Flattened File-List



This tool presents all the files that participate in the source tree in a single, flat list, and allows the user to see and change the contents of the files at any time, using the highly-evolved Squeak text-editing tools. Automatic versioning is provided, and the "Revert..." button allows for selective rollback.

The “Search” feature in the flattened file-list is not yet functional, but the intention is to provide an ingratiating alternative to “grep” for finding all occurrences of a given string anywhere in the source tree.

## 5. Searching for selectors: "Message Names"



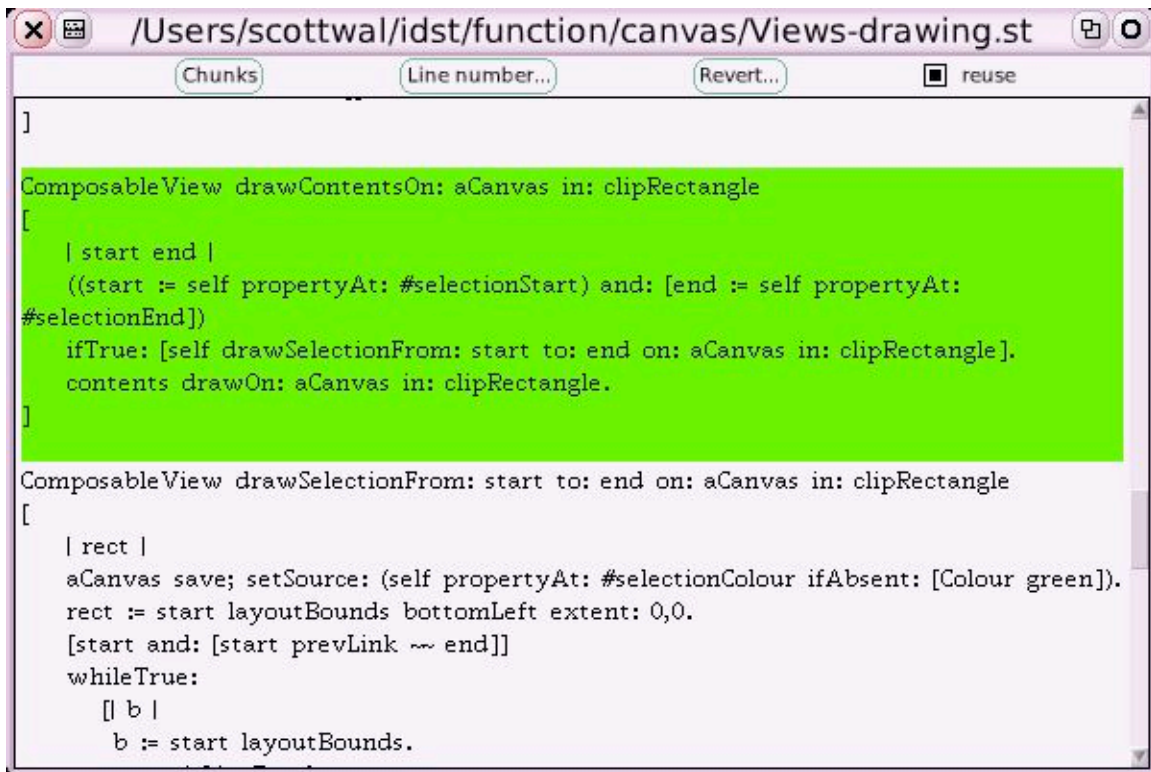
This tool allows the entire system to be searched for methods whose names (selectors) match any given string pattern; the retrieved methods can be viewed and edited in-place within the tool, and all the usual queries can be initiated within the tool as well.

In the example above, the user has searched for methods whose selectors contain the fragment "draw". Three selectors were found. One of these, #drawOn:in:, has been clicked on by the user, revealing three object types which implement that method. One of these implementations, that of TransformView, has been selected, and thus the source code we see in the bottom pane is the code for TransformView's implementation of #drawOn:in:.

## 6. Directly editing a source file

All of the code tools described above have a "source file" button at the right edge of the tool pane found directly above the code pane. When a method is selected in the tool, clicking on the "source file" button will serve to open up an editing window on the disk file which contains the method definition, scrolled to the position in the file in which the method definition occurs.

This supports a style of use in which the user wants to leverage off the ide's higher-level tools such as senders, implementors, and variable-references, but wishes to do code editing by "directly editing ascii files" rather than by incrementally editing individual methods in the source-ide.



The screenshot shows a text editor window titled "/Users/scottwal/idst/function/canvas/Views-drawing.st". The window has a menu bar with "Chunks", "Line number...", and "Revert..." buttons, and a "reuse" checkbox. The code is as follows:

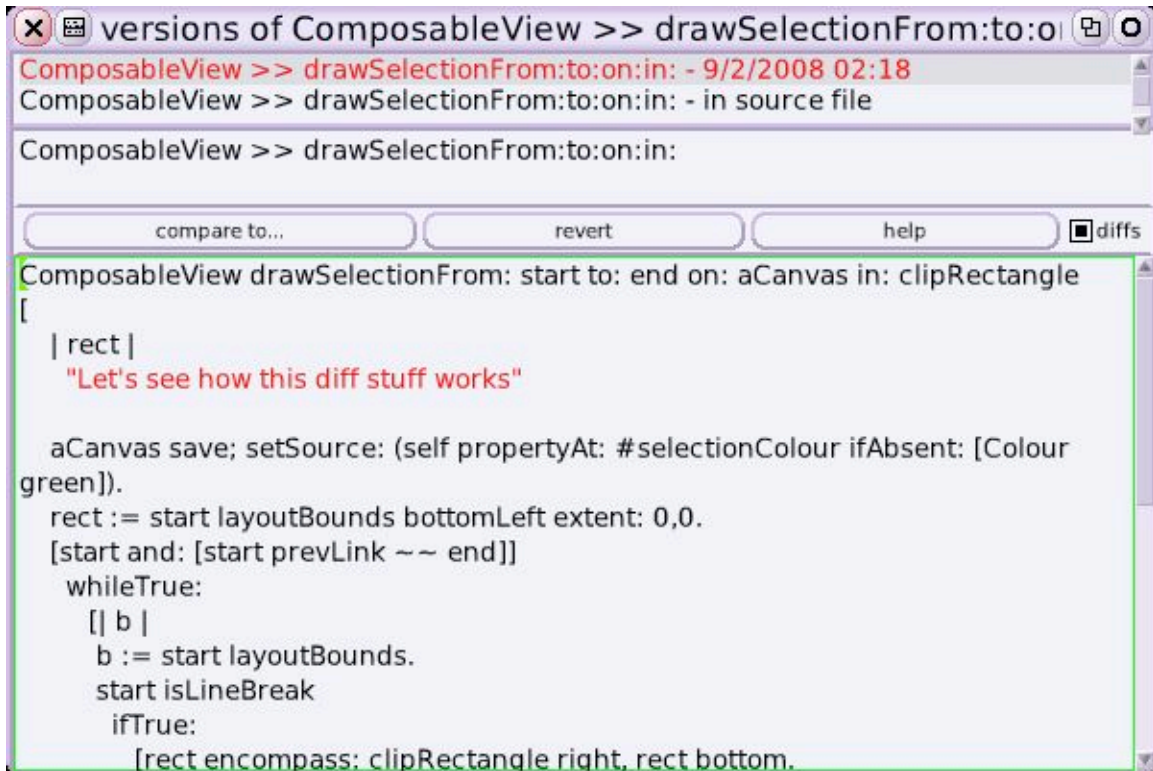
```
]
ComposableView drawContentsOn: aCanvas in: clipRectangle
[
  | start end |
  ((start := self propertyAt: #selectionStart) and: [end := self propertyAt:
#selectionEnd])
  ifTrue: [self drawSelectionFrom: start to: end on: aCanvas in: clipRectangle].
  contents drawOn: aCanvas in: clipRectangle.
]
ComposableView drawSelectionFrom: start to: end on: aCanvas in: clipRectangle
[
  | rect |
  aCanvas save; setSource: (self propertyAt: #selectionColour ifAbsent: [Colour green]).
  rect := start layoutBounds bottomLeft extent: 0,0.
  [start and: [start prevLink ~ end]]
  whileTrue:
    [| b |
     b := start layoutBounds.
```

The example above was created in response to the user's clicking on the "Source File" button in another code tool when the tool was pointed at method `ComposableView>>drawContentsOn:in:`. As will be seen, a window is opened on the file in question, and the window is scrolled to the appropriate place to reveal the method in question, and the actual source code is highlighted.

The user is now free, if she chooses, to edit the contents of the file directly, as if operating in any other text editor. This will however invalidate the "mental map" that the SourceIDE has of the code-base resident on disk, which therefore will need to be reinternalized after changes are made directly to a file in this manner.

## 7. Versions

The “versions” feature mirrors a feature found in Smalltalk systems. When new code is submitted for a given method, the code for the previous version of the method is remembered, and a tool is made available which offers access to every version ever submitted for the method, with various “diff” viewing available to assist in understand what has changed.



In the example above, a new version of the method `ComposableView >> drawSelectionFrom:to:to:in:` was submitted on 9/2/2008. The code pane shows the new code, with diffs from the prior version highlighted. Note the “revert” button.

## Next steps

Ultimately, it is anticipated that most Cola development will be done within Cola-created environments, akin to the “live” environments found in modern Smalltalk systems -- using native reflective tools to construct, traverse, extend, and debug code.

However, some bootstrapping stages of building Cola systems will always continue to require “cross-development,” in the sense that their source code will need to be written and edited in an environment that is not identical to the run-time environment being described by the source code.

Thus, even in an imagined image-based mainstream Cola future, there will always be the need for external management of source files. Someday IDEs written within Cola will suffice for such work.

But in the meantime, the SourceIDE will remain a useful tool for people not in the embrace of Unix and Emacs.

Unfortunately, the SourceIDE as currently constituted is something of a prisoner of its own provenance. Which is to say, it was possible to develop a quite comprehensive IDE in a short period of time only because the author was able to leverage, hugely, off of extensive higher-level support already present in Smalltalk. But none of that support exists in the current Cola code base, and attempting to port it all, wholesale, to Cola would be a massive, unwieldy, and inappropriate effort. So when some of the features of the SourceIDE ultimately appear natively in the Cola code-base, it is likely going to be due to fresh invention rather than to a straightforward port.

Meanwhile, however, it is anticipated that the SourceIDE will receive its first serious use, and make its first serious contribution, when the step of porting a simplified IDE written in so-called “Lesserphic” from Squeak to Pepsi is undertaken – a step currently planned for the final quarter of 2008.