Syntax-Elements for Smalltalk

Feb. 24. 2009

A Scratch-like GUI for Smalltalk-80
by Jens Mönig (jens@moenig.org)

Elements is a new graphical user interface for the Smalltalk-80 programming language inspired by MIT's Scratch (http://scratch.mit.edu). Like Scratch, *Elements* offers draggable "bricks" of code, which can be accumulated and assembled LEGO-wise into complex programming constructs, rather than entering text through a keyboard. The *Elements* project wants to find out, if and how Scratch's design can be applied not only to educational micro-worlds, but also to a full-fledged professional, object-oriented programming environment. As a proof-of-concept study I'm supplying both a minimal programming environment to start with, as well as a version of the Scratch Source Code Squeak environment which strives to be completely malleable by means of these *Elements*, thereby implementing Scratch quasi in itself.

Few pieces of software have inspired me the way Scratch has, with the possible exception of the Smalltalk programming language and the Morphic user-interface paradigm. In fact, I have come to love Scratch so much that I want everything to look and feel like Scratch. If such blissful narrow-mindedness turns you off, don't even bother reading any further . I also enjoy the nerdy delight coming out of manipulating something by means of itself. By letting *Elements* plug into Squeak's compiler/decompiler structure *Elements* can be modified and enhanced through itself.

The *Elements* Language

There are seven basic syntax elements:

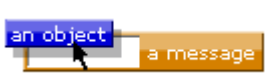
a literal
a closure
a step
a primitive
an answer

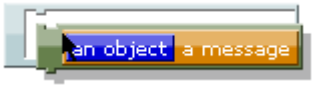
The first three rectangular elements resemble variations of *objects*. The last three puzzle-piece shaped elements embody *sequence*. The element in the middle – a closure – acts as adaptor piece between an internal sequence and an outside object behavior.

Note that elements' labels do not follow Smalltalk's camel casing. Instead, all Smalltalk expressions are automatically converted to multi-word labels and can also be user-edited as such.

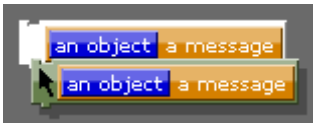
Clustering

Elements can be by combined by nesting:





and stacking:

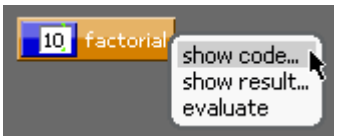


In addition, multiple messages sent to the same receiver can also be cascaded by dropping them onto the message which is to be evaluated last:

Elements with temporary variables and parameters feature their own local variable-palettes, from which new variable instances can be dragged off:

Direct Interaction

Right-clicking on an element offers several choices:



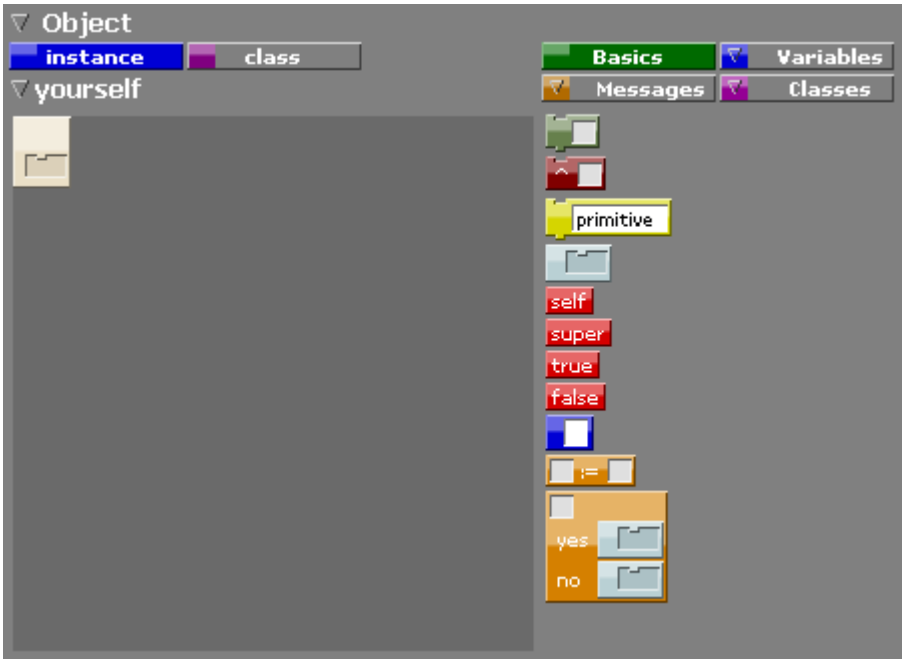
"show code" translates the current element including all of its sub-elements into Smalltalk code and pops up a dialog box displaying this code:

"show result" temporarily compiles and evaluates the current element including all of its sub-elements and pops up a dialog box displaying the result. This menu item corresponds to the "showIt" command in Smalltalk:

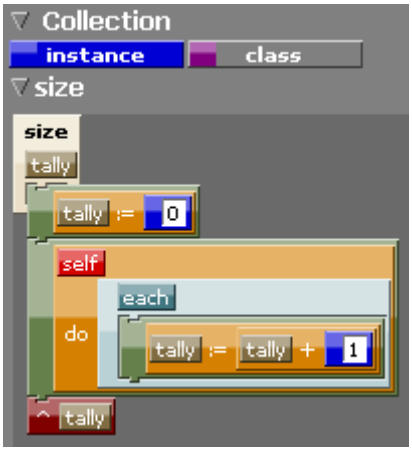
"evaluate" also compiles and executes the current element, but does not display anything. It resembles Smalltalk's "doIt" command, and is best used on expressions resulting in their own visual feedback:

The *Elements* Window

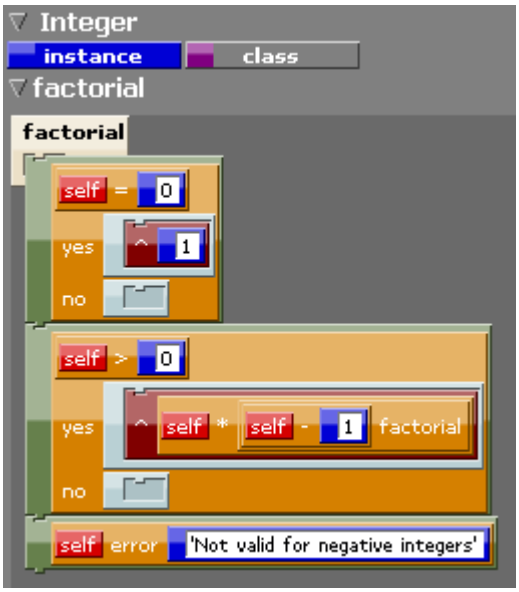
The *Elements* Window is the equivalent of a Smalltalk System Browser.



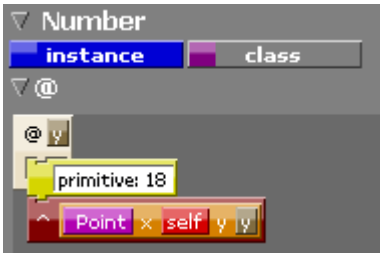
It consists of three parts: The header (top left), the palette (far right) and the scripting canvas (darker background). Method code can be browsed as elements on the scripting canvas by selecting a class and then choosing the desired method in the header:



Notice that Smalltalk's #ifTrue:ifFalse messages including all variants are always translated into universal yes/no elements:



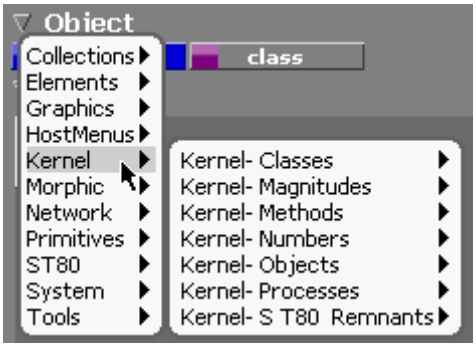
Primitive elements are followed by their fallback code:



Just looking at all the existing elements-code in the image should give the user a good impression how to use these elements.

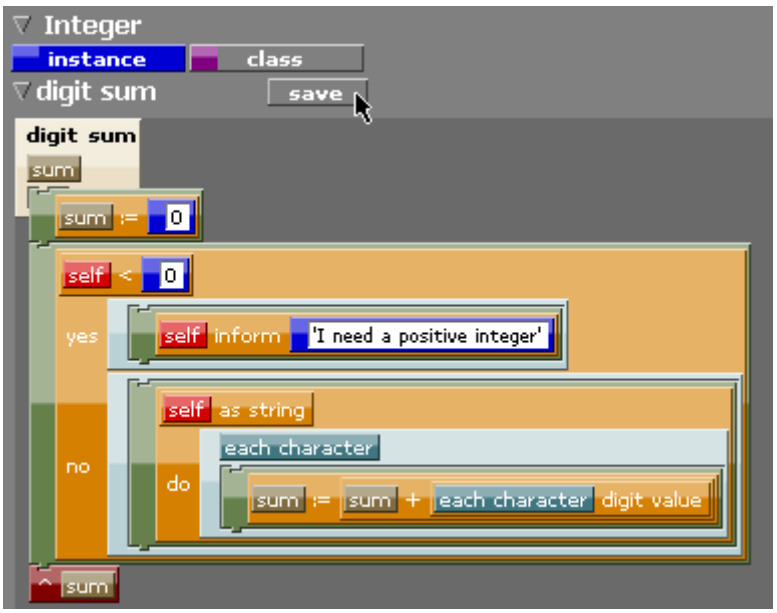
Creating a new Method

Creating a new Method starts by selecting an existing class in the header, and by specifying the "side" (instance / class) for it:



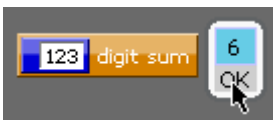
The method template in the scripting canvas can then be renamed into the desired label by right-clicking on its top region. In case one or more temporary variables are needed, templates for these can the added to the method's local palette also by right-clicking on the top-area:

Now the method's code body is ready to be constructed from elements dragged out of the palette. Pressing the "save" button compiles the element cluster into bytecode and stores it in the image:



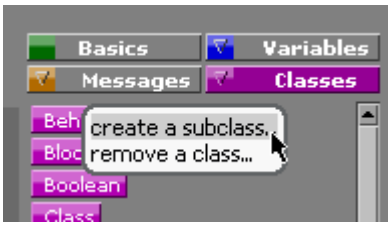
(Notice: You might be prompted by Squeak for your initials the first time you press the "save" button).

Afterwards the new method can be tested using "direct interaction":



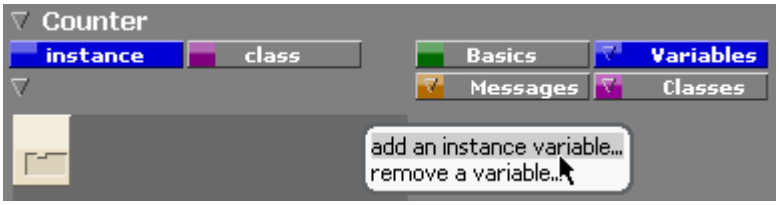
Creating a new Class

Creating a new Class starts by selecting it's superclass in the window header, by selecting the "Classes" tab in the palette, right-clicking onto the classes palette and entering a name for the new subclass:



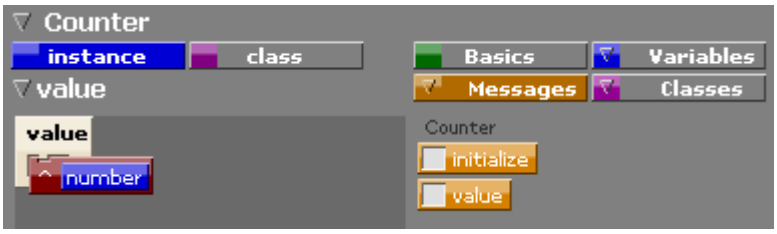
(in this example the superclass for the new class named "Counter" should be "Object")

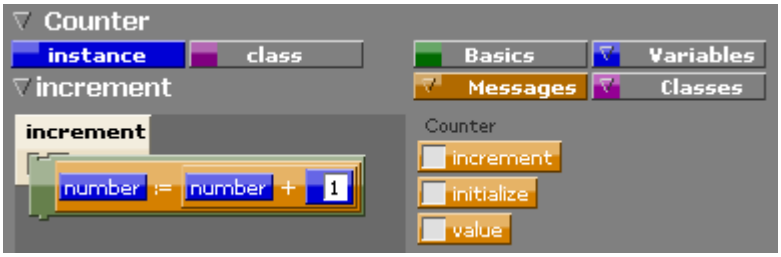
Once a new (empty) class has been created, instance variables can be added to it, again by selecting the "Variables" tab in the palette, and by entering one or more instance variables:



(Class variables can be added or removed in the same way by first selecting "class variables" from the tab's drop down menu).

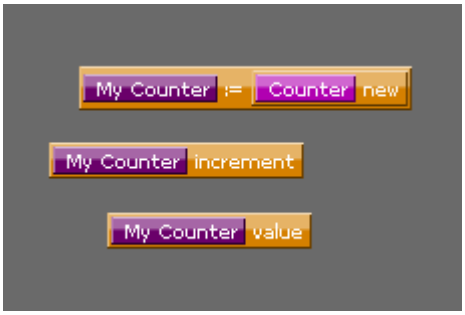
Afterwards methods can be added to the class which use instance/class variables:





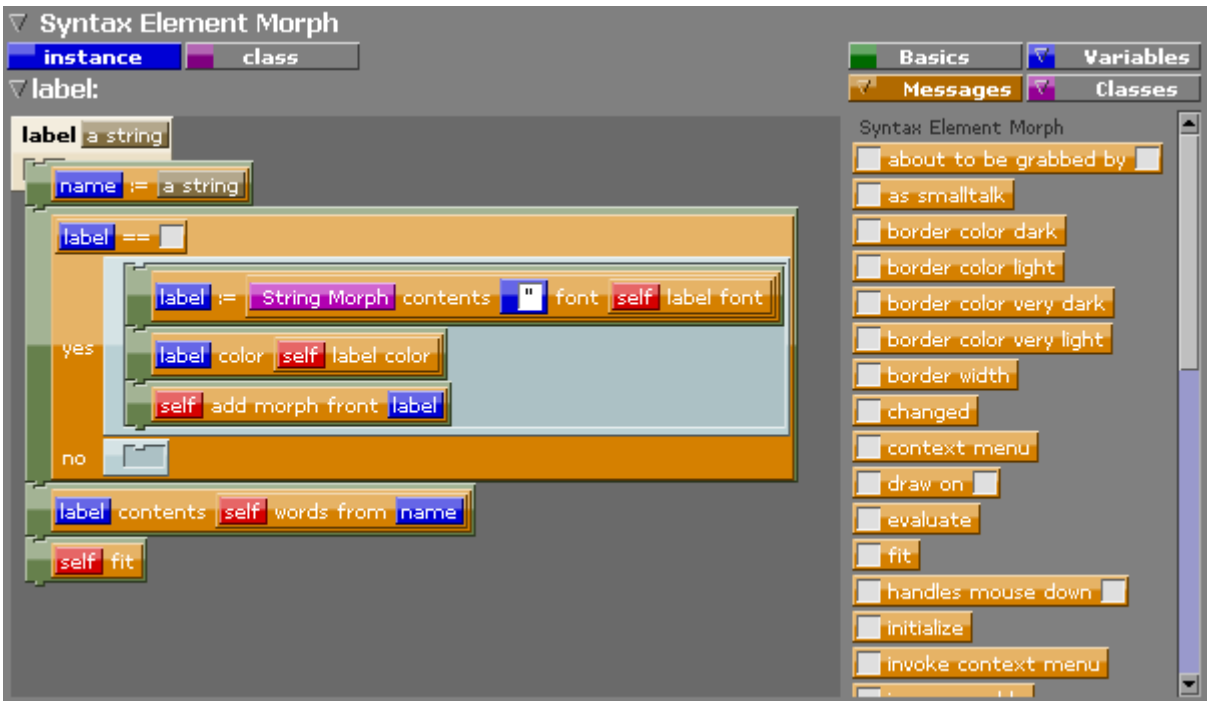
One way to test a new class is to create a global variable:

and to again use „direct interaction“ on test-case elements:



Self Reflective

Another fun thing to try is browsing the *Elements*-code itself within the *Elements*-Window, and finding out how it is integrated into Squeak:



Completion Status

This first version of *Elements* is a tentative experimental prototype, comparable to a novel's first draft. I'd like to use this prototype to gather experience and feedback for a more refined and mature design, and for a better and more stable GUI. Most, if not all of Smalltalk's compiler nodes can be decompiled into syntax elements, with the exception of certain Squeak-specific constructs, such as BraceNodes (Arrays within {}).

Keeping a distinct changeset for *Elements* turns out to be increasingly tricky when debugging the prototype. However, I would like to eventually port these syntax elements to other Smalltalk/Squeak versions as well (Etoys, Croquet,VisualWorks), and maybe even find yet other ways to facilitate programming with elements in Smalltalk. Among these are ideas to externalize and link-back image segments, but that's another project…

Credits

Elements is completely inspired by and based in essential parts on the fantastic and brilliant work of the MIT Media-Lab's Scratch-team, encouraged by their willingness to share ideas, designs, code, and enthusiasm. Thank you, John, Evelyn, Mitchel, Natalie, Andrés and Eric!