The Smalltalk Report

The International Newsletter for Smalltalk Programmers

June 1992 Volume 1 Number 8

GNU Smalltalk

By Steve Byrne

Contents:

Features/Articles

1 GNU Smalltalk
by Steve Byrne

5 SmallDraw—Release 4 graphics and MVC, Part 2
by Dan Benson

Columns

13 *Smalltalk Idioms:* The dreaded super
by Kent Beck

17 *GUIs:* Drag/Drop in Smalltalk/V PM
by Greg Hendley and Eric Smith

19 *The Best of comp.lang.smalltalk:* Encapsulation and information hiding
by Alan Knight

21 *Product Review:* VOSS—Virtual Object Storage System
reviewed by Wayne Beaton

Departments

23 *What They're Saying About Smalltalk*

26 *Product Announcements*

Although GNU Smalltalk is in use literally all around the world, knowledge of its existence is limited to the fortunate few who enjoy some sort of access to the Internet and Usenet network news. This article is an attempt to remedy this unfortunate situation.

I'll provide a brief introduction to what GNU Smalltalk is, where it is currently, where it's going in the future, and how to obtain it.

WHAT IS GNU SMALLTALK?

GNU Smalltalk is an implementation of the Smalltalk-80 programming language, as described in the book *Smalltalk-80: The Language and Its Implementation*[1] (a.k.a. "the Blue Book"). It includes most of the standard Smalltalk kernel class definitions, as well as a number of extensions to support system-level programming. It is part of the Free Software Foundation's GNU project, whose goal is to provide high quality, high functionality, freely available implementations of standard UNIX utilities, programming languages, and even the UNIX operating system itself. While not in the public domain, GNU programs (including, of course, GNU Smalltalk) are freely available in source code form, and run on a wide variety of UNIX-and non-UNIX-based platforms.

GNU Smalltalk has been generally available for over two years, and has seen many improvements and refinements over its lifetime. It is a continually evolving system, as we'll see later in this article.

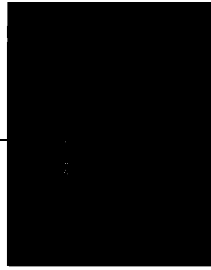
DESIGN GOALS

GNU Smalltalk is designed around several major goals:

- Compatibility support. Smalltalk is designed to support the Smalltalk-80 language as described in the Blue Book. This includes providing the same language syntax, the same kernel classes, and even the same byte code definitions for compiled methods.
- Portability. To allow the maximum number of people to be able to use the GNU Smalltalk environment, it must be highly portable. This means making as few assumptions about the environment as possible and isolating the few remaining system dependencies to particular modules.
- Ability to do system-level programming. To be truly useful, a software development system must be able to span a wide range of capabilities from expressing high level abstractions to doing system-level programming. The Smalltalk language itself takes care of supporting the creation of high-level abstractions. GNU Smalltalk currently has several features, and will have more in the future, which allow developers access to lower-level or system-level facilities, and allow it to operate with as much flexibility as native C programs have.
- Graphical user interface independence. Although providing a graphical user interface is one of Smalltalk's traditional fortes, it does limit its usability to environments which provide a bitmapped display. Often, access to a graphical display device is impossible,

continued on page 3...

John Pugh



Paul White

EDITORS' CORNER

"Smalltalk is looming larger" is the title of David Taylor's column in the May/June issue of our sister publication, Object Magazine. David is a well-respected consultant in the OOP world, and he expresses the opinion, which we voiced in a recent editorial, that Smalltalk is being adopted by many large companies for mainstream business applications. Certainly, our own experience as educators and consultants bears this out: Smalltalk, while retaining its traditional scientific and engineering base, is moving quickly into areas such as banking, insurance, and enterprise modelling. Although it would not be accurate to say that Smalltalk is becoming the dominant choice of industry (not yet, anyway), it is clear that more and more organizations see it as the best option for future application development.

In our lead article this issue, Steve Byrne describes the GNU Smalltalk effort. GNU is a project of the Free Software Foundation whose stated goal is to provide high-quality, high-functionality, freely available implementations of utilities and programming languages for the UNIX world. They have been successsful in the C world—GNU C, for example, is widely used by universities. The price is definitely right! GNU Smalltalk does not yet pose much of a threat to the major Smalltalk vendors. At present, the effort is entirely volunteer-driven. Perhaps Steve's article will prompt some of our readers to become involved. If you ever wanted to write your own garbage collector or interface to X, this may be your opportunity!

In the second of his series of three tutorial articles on Objectworks/Smalltalk, Dan Benson from the University of Washington continues his discussion of the development of SmallDraw, a structured-graphics editor. This month, he extends the abilities of SmallDraw to permit selection, translation, scaling, and modification of the visual attributes of objects in a drawing.

Kent Beck addresses the "dreaded super" in the latest of his Smalltalk idiom columns. We agree with Kent's assertion that super is confusing to and often misunderstood by many beginning Smalltalk programmers. Indeed, a favorite test question of ours is Where does the search for a method begin when you send a message to super? Since super is really a pseudonym for self, beginners usually say it starts in the superclass of the reciever (self). The correct answer (of course) is that the search begins in the superclass of the class in which the executing method was found. Kent describes a variety of idioms involving super and demonstrates how he extended the Smalltalk system to provide support for searching the image for uses of super.

Also in this issue, Greg Hendley and Eric Smith describe the use of the Drag and Drop facility in Smnalltalk/V PM. In his bulletin board round-up, Alan Knight describes an interesting dialog on USENET concerning the relationship between information hiding and encapsulation, and explains how Smalltalk programmers can contribute code to the Smalltalk archives. Finally, Wayne Beaton reviews the VOSS/Windows product from Logic Arts. VOSS (Virtual Object Storage System) is a persistent object storage system for Smalltalk V/Windows.

—The Editors

The Smalltalk Report (ISSN# 1056-7976) is published 9 times a year, every month except for the Mar/Apr, July/Aug, and Nov/Dec combined issues. Published by SIGS Publications Group, 588 Broadway, New York, NY 10012 (212)274-0640. © Copyright 1992 by SIGS Publications, Inc. All rights reserved. Reproduction of this material by electronic transmission, Xerox or any other method will be treated as a willful violation of the US Copyright Law and is flatly prohibited. Material may be reproduced with express permission from the publishers. Mailed First Class. Subscription rates 1 year, (9 issues) domestic, $65, Foreign and Canada, $90, Single copy price, $8.00. POSTMASTER: Send address changes and subscription orders to: THE SMALLTALK REPORT, Subscriber Services, Dept. SML, P.O. Box 3000, Denville, NJ 07834. Submit articles to the Editors at 91 Second Avenue, Ottawa, Ontario K1S 2H4, Canada.

continued from page 1...

due to lack of physical proximity (i.e., dialing up using a terminal or terminal emulator), or other reasons. GNU Smalltalk explicitly does not require the presence of a graphical display device; rather, it provides a full-featured development environment layered on top of the GNU Emacs editor.

- Central algorithm repository. Although this last goal is not so much of a design goal, it was always intended that as GNU Smalltalk progressed, support for more and more objects which provide reusable implementations of standard algorithms à la Sedgewick[2] would be provided.

CURRENT FEATURES

The current version of GNU Smalltalk has implementations of all the kernel classes and almost all the methods described in the Blue Book. It also provides a number of highly useful extensions for interacting with C subroutines, editing Smalltalk method definitions, and creating X Window–based graphical user interfaces. The highlights are detailed in the next few paragraphs.

To assist with interfacing Smalltalk to C code, GNU Smalltalk provides a simple mechanism for Smalltalk to invoke C functions and receive returned results. This simple capability allows most operations available in typical C runtime libraries to be available to the Smalltalk programmer as Smalltalk methods. The mechanism takes care of converting between Smalltalk datatypes and C datatypes, so the effort involved in using this facility is minimal.

GNU Smalltalk comes with extensions to the GNU Emacs editor to support the editing of Smalltalk programs. It includes program formatting assistance and provides many of the operations that are found in commercial Smalltalk operations, such as evaluate the current region as a Smalltalk expression, compile this method, etc. It also provides a mode where Smalltalk may be interactively invoked within an editor window, with the full range of text editing capabilities available.

GNU Smalltalk also comes with a simple interface to the X Window System, by providing direct access to the X client protocol. It provides an object-oriented encapsulation of most of the X protocol operations, and comes with some simple examples of usage. This facility is greatly enhanced in the next release, as outlined below.

GNU Smalltalk currently runs on most 32-bit UNIX platforms. Thanks to its focus on portability, it typically ports to new platforms in a matter of hours or less. It has also been ported to VMS, and to some 16-bit machines, such as Atari.

FUTURE DIRECTIONS

The plans for GNU Smalltalk's future fall into two categories: those scheduled for the next minor release, and those for the next major release. We'll first describe the new features the next minor release will have, and then describe the major improvements for the next major release.

NEAR-TERM FEATURES

The next release of GNU Smalltalk (which will be version 1.2), will have numerous significant features added to it. The most significant ones are described briefly here.

GNU Smalltalk Version 1.2 will have several new capabilities for doing more system-level programming and general interfacing to existing C libraries. It will allow Smalltalk programs to directly access and manipulate C variables and structures. An interface for C programs to directly invoke Smalltalk methods is provided. Smalltalk isn't even required to be the main program: the C programmer can request that initialization of Smalltalk be performed with certain command-line parameters at a particular point in the execution of the program, or can choose to accept the standard defaults and just begin invoking Smalltalk methods at will and let Smalltalk auto-initialize itself. To better support the needs of system-level programming, GNU Smalltalk Version 1.2 will also provide direct support for the handling of interrupts, both system and user generated.

Version 1.2 will give developers more direct control over memory usage. The amount of memory GNU Smalltalk requires as a comfortable minimum for object storage has been significantly reduced. Should a particular application require more memory as it runs, GNU Smalltalk will automatically increase the amount of space dedicated to object memory, at a programmer-definable rate when memory usage exceeds a programmer-defined memory usage threshold.

The existing interface to the X Window System has been completely rewritten and provides complete access to the X Window protocol. It also includes support for the X authorization mechanism, so developers need not take any special action (such as using the *xhost* command) to use this facility. It will feature a simple, extensible set of graphical user interface components, similar to those provided by the OPEN LOOK or Motif toolkits, as well as support for the more traditional Smalltalk rendering model. And finally, support for the majority of Inter-Client Communication Conventions Manual (ICCCM) standard window properties will be provided.

As the GNU Smalltalk graphical user interface support becomes more full-featured and mature, it becomes ever more critical that the performance of the system be as high as possible to support the demands of interactive user interfaces. Version 1.2 incorporates substantial performance improvements. The garbage collector has been streamlined. The system creates only real method contexts when necessary and uses a cache of pseudo method contexts to dramatically decrease method invocation time. A number of the busiest method selectors have had their invocation overhead decreased. The decrease in overall system memory requirements also helps to improve performance on virtual memory systems with limited physical memory.

One of the features which helps make software development in Smalltalk more efficient and productive is the class browser. The next release will provide support for an Emacs-based class browser, and possibly a graphical user interface based class browser as well. The Emacs Smalltalk editing mode also includes support for method tracing and debugging, as well as things like locating all the methods present in the system that have a particular selector and allowing the user to quickly browse the method definitions for each.

Other new features in Version 1.2 include the ability to dynamically load and access C functions and variables while GNU Smalltalk is running. This is facilitated by using the GNU dynamic loading library (a separate GNU project utility library). Although it may sound simple, this is an extremely powerful capability to have in an interactive system.

Also new with Version 1.2, Smalltalk method definitions may be conditionally compiled based upon boolean expressions, which can include tests on the presence of the exact same set of preprocessor symbols that C programs enjoy, through the new Features facility. This allows for machine-architecture and operating-system–specific versions of methods to be created and to have the appropriate definitions selected based upon the execution environment.

Finally, Version 1.2 will include support for operations on large integers. It is also likely that an Emacs-based hypertext Smalltalk method reference manual will be available in the 1.2 timeframe.

THE NEXT MAJOR RELEASE

The goal of producing a highly portable system means that the number of operating-system and machine-architecture dependencies must be minimized. The direct implication of this is, of course, no machine specific code generator: the current implementation uses a simple byte code compiler and interpreter to handle the execution of Smalltalk code. While this has major portability advantages, it also means that the code executes much more slowly than optimized machine code.

The next major release of GNU Smalltalk will address the performance issue in a number of different ways. The first will be through the implementation of a generational garbage collector. Smalltalk memory use tends to be relatively bimodal. There is a relatively unchanging component in the form of method definitions, and kernel method definitions, in particular, and there is a much more dynamic and short-lived component in the form of transitory objects such as method and block contexts and intermediate computational results. A generational garbage collector is ideally suited to this environment, as it provides support for several different memory areas with different object longevity characteristics and a tenuring policy that can move newly created objects into less dynamic memory areas as they survive successive garbage collection passes.

The second approach to improving performance of the system involves switching from interpreting byte codes to compiling to machine code. The techniques for compiling Smalltalk to efficiently executing machine code are becoming relatively well understood. To achieve compilation in a portable way, the approach that's being taken is to use the GNU C compiler (GCC) back end to perform the actual machine specific code generation and optimization. The ability to have this kind of code sharing and code reuse is one of the major benefits of the Free Software Foundation's approach of building high quality tools that are freely available.

The GNU Smalltalk project is an all-volunteer effort. The rate at which GNU Smalltalk changes and improves is directly related to the number of people who contribute their time and talents to making GNU Smalltalk a world-class software development environment. The GNU Smalltalk project is open to anyone with an interest in Smalltalk who would like to contribute their time and energies to extending and improving the system. People from around the world have already contributed substantial work to the project, but we can always use more help.

FOR MORE INFORMATION

For more information, please write to me at the address below. GNU Smalltalk may be anonymously FTP'ed from a variety of Internet hosts, but its primary location is on prep.ai.mit.edu. It can also be obtained from the mail server at the University of Illinois at Urbana-Champaign. To have the mail server send you GNU smalltalk, send it a message like the following:

```
To: archive-server@st.cs.uiuc.edu
Subject:
path yourname@your.internet.email.address
archiver shar
encoder uuencode
help
encodedsend gnu_st/smalltalk-1.1.1.tar.Z
```

For more information about the GNU project and the Free Software Foundation in general, please send email to gnu@prep.ai.mit.edu or write to Free Software Foundation, 675 Massachusetts Avenue, Cambridge, MA 02139; 617.876.7739. ■

REFERENCES

1. Goldberg, A. and D. Robson. Smalltalk-80: *The Language and Its Implementation*, Addison-Wesley, Reading, MA, 1983.
2. Sedgewick, R. *Algorithms*, Addison-Wesley, Reading, MA, 1983.

Steve Byrne is a Staff Engineer at SunSoft, a Sun Microsystems Inc. company. His interests include programming language design and implementation, operating systems, software development environments, freely sharable software, and object-oriented programming in general. He can be reached at sbb@eng.sun.com or 5269 Carter Ave., San Jose, CA 95118.

SMALLDRAW—RELEASE 4 GRAPHICS AND MVC, PART 2

Dan Benson

In Part 1 of this series, we outlined the basic concepts of Release 4 graphics and MVC application construction and described the development of a structured-graphics editor called SmallDraw. The application consists of the three MVC components and a set of graphic objects that are able to display themselves on a graphics context. The editor is minimal in its abilities—new objects can be created interactively, colors and line widths can be specified, and the view can be scaled, but objects could not be modified once they were drawn. In part 2 of this series, SmallDraw will be enhanced to include object selection, translation, and scaling. With more changes taking place in the view, we'll also add improved updating techniques.

SELECTING OBJECTS

An editor is not very useful unless changes can be made to the drawing. A structured-graphics editor should provide facilities for moving and resizing objects as well as changing objects' visual attributes of inside color, border color, and line width. This requires a means of identifying and distinguishing objects in the drawing, which will be referred to as object selection. Several issues need to be addressed in selecting objects that affect each of the three MVC components and involve user interface conventions. The following sections will look at each of these issues: making a selection, keeping track of selected objects, distinguishing selected objects visually, and modifying selected objects.

MAKING A SELECTION

Following modern interface conventions, the most direct way of selecting a single object in a drawing is by clicking once on it with the cursor. One can think of "clicking" on an object as trying to touch it with a finger. To the user, the objects in the drawing appear to be hollow (no inside color) or solid. To select a hollow object one would have to touch an edge, whereas a solid object can be selected by touching it anywhere in its interior (or on its edge). Objects also appear to be stacked on one another, so if the user clicked on a stack of objects, the first one touched would be selected.

Selecting more than one object is done by rubber banding a rectangle around them so they are completely contained in the rectangle. The set of selected objects can be extended by selecting other objects while holding down the shift key. In other words, specific objects may be added or removed from the set of currently selected objects.

How is it determined whether or not an object has been touched by a mouse click or surrounded by a rectangle? The objects themselves can decide these two situations. The three possible tests are: an object is contained in a rectangle, an object's interior contains a point, and an object's edge is touched by a point. Since these tests should be performed as quickly as possible, it would be convenient for each object to retain a **boundingBox** instance variable. All objects must be able to perform these tests, so the following **SDGraphicObject** instance methods are defined:

```
isHollow
    ^self insideColor isNil

insideRectangle: aRectangle
    ^aRectangle contains: self boundingBox

containsPoint: aPoint
    ^(self boundingBox containsPoint: aPoint)
        ifTrue: [self isHollow
            ifTrue: [self edgeContainsPoint: aPoint]
            ifFalse: [(self interiorContainsPoint: aPoint)
                or: [self edgeContainsPoint: aPoint]]]
        ifFalse: [false]

edgeContainsPoint: aPoint
    self subclassResponsibility

interiorContainsPoint: aPoint
    self subclassResponsibility
```

Note that the last two methods are left up to the subclasses to implement, since **SDGraphicObjects** only know about **boundingBoxes**. To check for containment of a point, the object's **boundingBox** is first checked to eliminate the trivial cases. If the object is hollow, a check is made to see if the point is on or near one of its edges. For a solid object its interior and edges are checked.

The tests for rectangles and line segments are trivial, but are much more involved for polygons and ellipses. There are several well-known methods for determining whether a polygon contains a point. From my experience, the best overall performance is obtained using what is known as the "winding" method. The basic idea is to sum the angles from the point in question to each vertex in the polygon. If the point is inside the polygon, the sum of the angles will be 2π, otherwise the point is outside the polygon. However, rather than work with angles, the calculations can be simplified by assigning a num-

ber to each of the four quadrants. A point can then be asked for the quadrant number, relative to itself, that contains another given point. This then becomes aPoint instance method:

```
quadrantContaining: aPoint
    "Answer the number of the quadrant containing aPoint relative to
the receiver, where the quadrants are numbered as follows:

        1 | 0
        ———
        2 | 3

This convention is used for determining whether a point is in a polygon."

    ^aPoint x > x
        ifTrue: [aPoint y >= y
            ifTrue: [3]
            ifFalse: [0]]
        ifFalse: [aPoint y >= y
            ifTrue: [2]
            ifFalse: [1]]
```

Numbering the quadrants in this manner means that the quadrant numbers can simply be added as the list of vertices is traversed. Numbers are added when moving in a clockwise direction and subtracted moving counter-clockwise. Jumping diagonally adds or subtracts 2 from the running total. If the final winding number is not 0, the point is inside:

```
interiorContainsPoint: aPoint
"Answer whether the receiver contains aPoint on its boundary or in its
interior. Uses the winding technique. See the method
Point|quadrantContaining:."
    | wind lastPoint oldQuad newQuad |
    wind := 0.
    lastPoint := self vertices last.
    oldQuad := lastPoint quadrantContaining: aPoint.
    self vertices do: [:each |
      aPoint = each ifTrue: [^true].
      newQuad := each quadrantContaining: aPoint.
      oldQuad = newQuad
        ifFalse: [oldQuad+1\\4 = newQuad
          ifTrue: [wind := wind + 1]
          ifFalse: [newQuad+1\\4 = oldQuad
            ifTrue: [wind := wind - 1]
            ifFalse: [| a b |
                a := lastPoint y - each y.
                a := a * (aPoint x - lastPoint x).
                b := lastPoint x - each x.
                a := a + (b * lastPoint y).
                b := b * aPoint y.
                a > b
                  ifTrue: [wind := wind - 2]
                  ifFalse: [a = b ifTrue:[^true] ifFalse: [wind := wind + 2]]]]].
      oldQuad := newQuad.
      lastPoint := each].
    ^wind isZero not
```

The other check is whether a point is on or near the edge of a polygon. Clicking exactly on a line is often difficult so a tolerance can be specified for all SDGraphicObjects so that the user doesn't have to be too precise with the mouse:

```
tolerance
  "Answer the minimum distance that a point can be from an edge of
  the receiver to constitute a 'hit'."
```

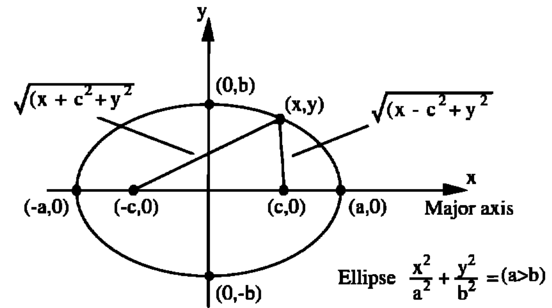


Figure 1. An ellipse centered at the origin with x as its major axis.

```
  ^(self lineWidth/2) truncated + 2

edgeContainsPoint: aPoint
  "Answer whether any one of the receiver's edges contains aPoint. This
  is true if aPoint is within a certain distance from an edge - see
  message: tolerance."
    | from delta |
    delta := self tolerance.
    from := self outline first.
    self outline do: [:pt |
        ((aPoint nearestIntegerPointOnLineFrom: from to: pt )
            dist: aPoint) <= delta
            ifTrue: [^true].
        from := pt].
    ^false
```

Performing the same tests for an ellipse requires a closer look at its definition (see Figure 1). If the ellipse happens to be a circle (its width is the same as its height), the tests are trivial. The focus points of an ellipse are determined using the major and minor axis radii (a and b, respectively) in the following equation:

$$c = \sqrt{(a^2 - b^2)}$$

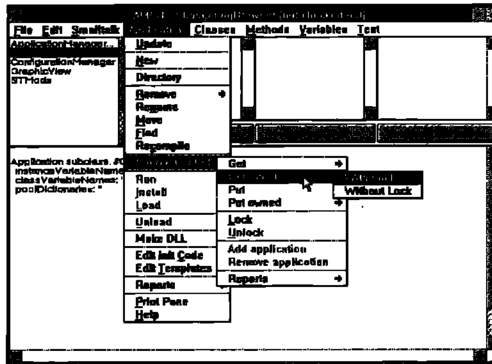
The sum of the distances between the two focus points and any point on the edge of the ellipse is constant, namely 2a. Therefore, the sum of the distances between the two focus points and any point inside the ellipse will be less than or equal to 2a:

```
interiorContainsPoint: aPoint
    ^self isCircle
        ifTrue: [(self center dist: aPoint) <= self xRadius]
        ifFalse: [| offset constantLength |
"Determine focus points on major axis."
            self width >= self height
                ifTrue: [offset := (self xRadius squared - self yRadius
                                squared) sqrt@0.
                        constantLength := self width]
                ifFalse: [offset := 0@(self yRadius squared - self xRadius
                    squared) sqrt.
                        constantLength := self height].
"Now, answer whether the sum of the distances between aPoint and the
two focus points is less than the constantLength."
            (((self center - offset) dist: aPoint) + ((self center + offset)
                dist: aPoint)) <= constantLength]
```

Checking the edge of an ellipse is similar, with a slight variation to check whether the sum of the distances between the focus points and the point in question is close to 2a:

ImageSoft

AM/ST™

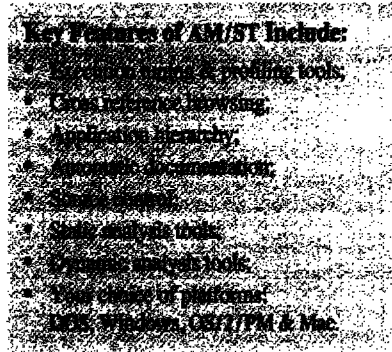


The original and still premier application manager for Smalltalk/V.™

AM/ST, developed by the SoftPert Systems Division of Coopers & Lybrand, enables the developer to manage large, complex, object-oriented applications. The AM/ST Application Browser provides multiple views of a developer's application.

AM/ST defines Smalltalk/V applications as logical groupings of classes and methods which can be managed in source files independent of the Smalltalk/V image. An application can be locked and modified by one developer, enabling other developers to browse the source code. The source code control system manages multiple revisions easily.

Coopers & Lybrand



ChangeBrowser. As an additional tool available for Smalltalk/V PM and Smalltalk/V Windows, ChangeBrowser supports browsing of the Smalltalk/V change log file or any file in Smalltalk/V chunk format.

The addition of AM/ST to the ImageSoft Family of software development tools enhances and solidifies ImageSoft's position as — "The World's Leading Publisher of Object-Oriented Software Development Tools."

1-800/245-8840

ImageSoft™
The World's Leading Publisher of Development Tools

All trademarks are the property of their respective owners. ImageSoft, Inc., 2 Haven Avenue, Port Washington, NY 11050 516/767-2233; Fax 516/767-9067; UUCP address: mcdhup!image!info

```
edgeContainsPoint: aPoint
    ^self isCircle
        ifTrue: [(self center dist: aPoint) <= (self xRadius + self tolerance)]
        ifFalse: [| offset constantLength |
"Determine focus points on major axis."
            self width >= self height
                ifTrue: [offset := (self xRadius squared - self yRadius
                            squared) sqrt@0.
                    constantLength := self width]
                ifFalse: [offset := 0@(self yRadius squared - self xRadius
                            squared) sqrt.
                    constantLength := self height].
"Now, answer whether the sum of the distances between aPoint and the
two focus points is close enough to the constantLength."
            ((((self center - offset) dist: aPoint) + ((self center + offset)
                    dist: aPoint)) - constantLength) abs < self tolerance]
```

KEEPING TRACK

Now that objects are able to determine whether they contain a point or are inside a rectangle, the application must keep track of which objects are selected. Since the SmallDraw model keeps track of all the objects in the drawing, it's reasonable to have it also keep track of which objects are selected. A simple way to do this is to associate a Boolean flag with each graphic object identifying it as selected or not. The SmallDraw model uses an OrderedCollection to store the graphic objects. The order is important because it determines the layering of objects. The ordered list can be retained, but instead of a list of graphic objects it will be a list of Associations where each graphic object is the key and the Boolean flag is the value. Selected objects would then be those Association keys whose values were true, as answered in the following SmallDraw instance method:

```
selectedObjects
    "Answer all currently selected objects."
    ^(self objects select: [:p | p value]) collect: [:a | a key]
```

The SmallDraw instance method to answer its display objects is now modified to be:

```
displayObjects
    "Answer the receiver's objects in order for display purposes."
    ^self objects reverse collect: [:a | a key]
```

Adding a new object is modified to add a new Association, selecting the newly added object at the same time:

```
self objects addFirst: (anObject -> true).
```

Finally, deselecting an object is done by setting its corresponding Association value to false. The following method deselects all currently selected objects:

```
deselectAll
    self objects do: [:p | p value: false]
```

IDENTIFYING SELECTED OBJECTS

In sticking with modern interface conventions, selected objects will be identified through the metaphor of "handles" that appear as small solid squares at an object's extremities. For the set of SmallDraw objects, the corners of the bounding box can serve as handle points. Whenever an SDGraphicObject changes its shape it should recalculate its boundingBox, and whenever its boundingBox changes so will its handles. Since handles are expected to be accessed often it will be best to store them as

an instance variable and be automatically updated, as in the following SDGraphicObject instance methods:

```
computeBoundingBox
    "Subclasses should set their own boundingBox using
setBoundingBox:"
    self subclassResponsibility

setBoundingBox: aRectangle
    "Set the receiver's boundingBox and update its handles."
    boundingBox := aRectangle.
    self setHandles

setHandles
    "For consistency, handles should be set in a clockwise direction."
    handles := (Array
        with: self boundingBox origin
        with: self boundingBox topRight
        with: self boundingBox corner
        with: self boundingBox bottomLeft)
```

There are cases where it is appropriate to display only two of the four handles, for instance, when an object is "squished," so that it is completely horizontal or vertical. Also, an SDLineSegment will always display two handles, one at each end point. The view and controller will want to access the handle points used for display purposes, so each object will be responsible for supplying appropriate display handles:

```
isSquished
    "Answer whether the receiver is either totally horizontal or totally
    vertical."
    ^(self boundingBox width isZero or: [self boundingBox height is Zero])
displayHandles
    "Answer the handles to use for displaying the receiver."
    ^self isSquished
        ifTrue: [Array with: self boundingBox origin
                with: self boundingBox corner]
        ifFalse: [self handles]
```

SDLineSegments will always return their end points for displaying their handles:

```
displayHandles
    ^Array with: self start with: self end
```

With this convention, handles for SmallDraw objects will appear as illustrated in Figure 2. Handles serve not only as a feedback mechanism to identify a selected object, but also as visual indicators for control points used in stretching or shrinking an object as if grabbing and pulling an actual handle. The view must be able to turn on and off handles very quickly and the controller must be aware of handles in order

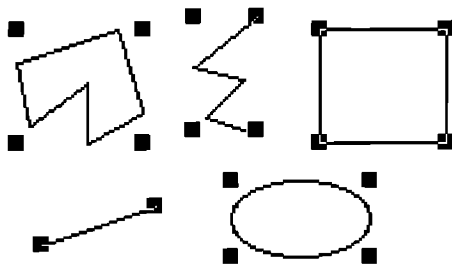


Figure 2. Graphic object handles.

to determine when one has been hit by the cursor and when modifying an object. The only components concerned with the size and shape of handles are the SmallDrawView and SmallDrawController. For this reason, a generic handle is defined as a SmallDrawView class variable, initialized as a rectangle with extent (6@6) centered at the origin:

```
initialize
    "Initialize the size of the handles with a Rectangle centered at
    the origin."
    DrawHandle := (3@3) negated corner: (3@3).
```

As a class variable, DrawHandle is accessible by all instances of SmallDrawView and can be easily resized for users who prefer a smaller or larger handle. Centering it at the origin allows it to be translated to an object's handle locations for rendering on screen. All the view has to do is to ask each object for its handle points, translate the generic handle to each location and display the handles.

Handles need to be displayed and erased quickly. This can be accomplished by taking advantage of Image methods that combine bits using RasterOp rules. Many thanks to Patrick McClaughry for the following technique that was borrowed from his version of HotDraw, written for Release 4 and available via ftp over the internet. The basic idea is to grab from the screen the small rectangular region where a handle will be displayed. The bits in the rectangle are reversed by copying the image onto itself using the **RasterOp reverse** rule (actually, **RasterOp reverse** does not work correctly on all platforms but apparently using the rule with integer value 10 does work). The resulting image is then displayed in the view. Repeating this bit operation again results in the original bits as if nothing had changed. In this way, handles can be turned on and off without having to redisplay any of the objects in the view. The following SmallDrawView method can be used to turn handles both on and off:

```
toggleHandlesFor: aCollectionOfObjects on: aGC
    aCollectionOfObjects do: [:o | o handles do: [:h | | rect image |
        rect := DrawHandle translateBy: h * self displayScale.
        (aGC clippingBounds intersects: rect)
            ifTrue: [rect := ((rect intersect: aGC clippingBounds)
                            translateBy: aGC translation) rounded.
                    (image := (aGC medium contentsOfArea: rect) first)
                        copy: (0 @ 0 extent: rect extent)
                        from: 0 @ 0

                        in: image
                        rule: 10;
                        displayOn: aGC
                        at: rect origin - aGC translation]]]
```

Note that it is necessary to clip each handle to the graphic context's clippingBounds because an error occurs when accessing bits outside of the clippingBounds.

MAKING SELECTIONS

From the discussion of MVC components in Part 1, it should be obvious that the SmallDrawController plays a role in selecting objects since it handles all user input. When the select button

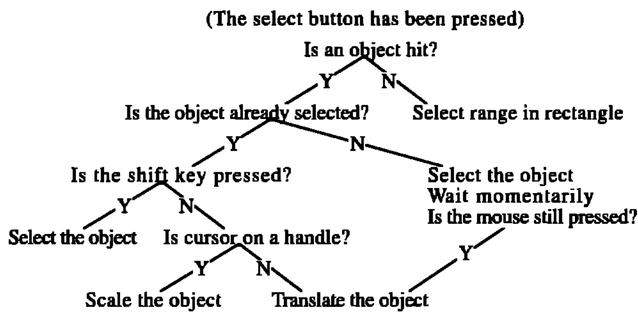


Figure 3. The controller's decision tree.

is pressed, the controller has to somehow determine whether the user wants to click on an object or make a range selection by drawing a rectangle around a group of objects. These two possibilities are easily distinguished since a rectangle can only be started if the cursor is not on an object. If the shift key is pressed while making a selection the current selection should be extended to either include or exclude the newly identified objects. Of the several possibilities that can occur when the mouse button is pressed there are four possible outcomes: a single object is selected, a range of objects is selected in a rectangle, an object is scaled, and an object is translated. The controller's decision tree is shown in Figure 3.

The first test is to see whether an object has been hit by the cursor. Selected objects take precedence if one of their handles is touched, so these are tested first. Otherwise, all of the model's objects are tested. In either case, the first object hit is immediately answered:

```
objectHitBy: aPoint
    "Answer the first objecthit, or nil."
    self model selectedObjects do: [:each |
        each displayHandles do: [:pt |
            (self view handle containsPoint: aPoint - pt)
                ifTrue: [^each]]].
    self model allObjects do: [:each |
        (each containsPoint: aPoint)
            ifTrue: [^each]].
    ^nil
```

Determining a range of objects inside a given rectangle is straightforward:

```
objectsInRectangle: aRectangle
    ^self model allObjects select: [:o | o insideRectangle: aRectangle]
```

The controller notifies its model of the selection, and whether or not it is an extended selection. If the selection is not to be extended, all objects are first deselected. Boolean flags identify selected objects, so the model simply negates the flags of those objects to be selected (or deselected, as the case may be):

```
selectRange:aCollectionOfObjects extend: aBoolean
    aBoolean ifFalse: [selfdeselectAll].
    aCollectionOfObjects do: [:obj | | oa |
        (oa := self objects detect: [:o | o key == obj] ifNone:[nil]) notNil
            ifTrue: [oa value: oa value not]].
```

MODIFYING SELECTIONS

Once a selection is made, the selected objects can be modified in a number of ways. The previous version of SmallDraw provided a means for changing the three default visual attributes (inside color, border color, line width) through the model's menu options. These three menu selector methods must be modified to account for selected objects. If any objects are currently selected, the changes should apply only to them. If there are no currently selected objects, the changes should apply only to the model's default attributes.

The other modifications that can take place are translation and scaling. Whenever the selected objects are translated or scaled, immediate feedback should be provided indicating the effects of the change. For instance, when moving an object, it should appear as if the object is actually being picked up and moved. The visual feedback will be an outline of the object as it follows the cursor to its final destination. Once the mouse button is released, the actual object will then be asked to move to the new location. If more than one object is selected, the bounding box of the entire selection will also be displayed so that it will be easy to see where the group of objects will end up.

In order for this to happen, every SDGraphicObject will have to supply an array of points describing its outline. This is a simple matter for the set of SmallDraw objects with the exception of SDEllipse. The points outlining an ellipse can be calculated based on its width and height. This time-consuming calculation can be improved by doing some of the work in advance. An array of points describing a unit circle centered at the origin will be precomputed and stored as an SDEllipse class variable:

```
initialize
    "Construct an array of points that describe a unit circle."
    UnitCircle := OrderedCollection new: 37.
    0 to: 2*Float pi by: Float pi / 18 do: [:a |
        UnitCircle add: (a cos @ a sin)].
    UnitCircle addLast: UnitCircle first.
    UnitCircle := UnitCircle asArray
```

An array of 37 points gives a reasonable approximation of a smooth outline for an ellipse. When asked for its outline, an ellipse merely has to translate and multiply the points contained in the class variable as follows:

```
outline
    "Answer an array of points that represent the receiver as an outline."
    ^UnitCircle collect: [:pt |
                    self center + (pt * (self xRadius@self yRadius))]
```

The visual feedback for scaling an object will be a rectangular outline using its handles or, in the case of an SDLineSeg-

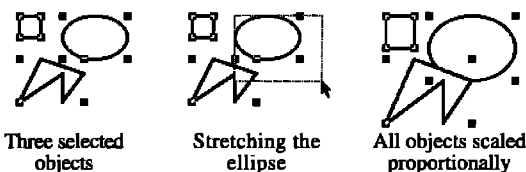


Figure 4. Modifying selections in SmallDraw.

ment, its end points. During scaling, as the handle under the cursor is moved, the opposite handle remains stationary. The SmallDrawController knows which handle is under the cursor and asks the object for the opposite handle point (to be anchored down). Finally, the scaling factor is calculated and the model is asked to scale its set of selected objects. Each object scales itself uniformly about its own handle in the same position as the anchored handle point. In this way, all selected objects are scaled proportionally (Figure 4).

```
scaleObject: anObject usingHandle: aHandlePoint
    "The cursor is over the handle at aHandlePoint. The opposite handle
    point should remain stationary (anchorHandle) as the cursor is moved."
    | scale anchorHandle newPoint |
    scale := self view displayScale.
    anchorHandle := anObject handleOpposite: aHandlePoint.
    newPoint := (anObject animateUsingRectangle
        ifTrue: [self cornerOfRectangleFromScreenWithOrigin:
                anchorHandle * scale]
        ifFalse: [self endOfLineFromScreenWithOrigin: anchorHandle *
                scale]) / scale.

"NOTE: If the object being scaled is Squished (either vertical or
horizontal), it will have a zero in its original vector diagonal so scaling
should be done in absolute terms and the percentage is calculated from
the resulting unitVector multiplied by the ratio of the
newDiagonalDistance over the oldDiagonalDistance. Absolute scaling
should only performed by other 'squished' objects.
    If the object being scaled is not squished there is no danger of
division by zero so the percentage is calculated from the ratio of the
newDiagonalVector divided by the oldDiagonalVector."
    self model
        scaleBy: (anObject isSquished
                ifTrue: [(newPoint - anchorHandle) unitVector *
                        ((newPoint dist: anchorHandle) /
                            (aHandlePoint dist: anchorHandle))]
                ifFalse: [(newPoint - anchorHandle) /
                        (aHandlePoint - anchorHandle)])
        aboutHandleAt: (anObject indexOfHandle: anchorHandle)
        absolute: anObject isSquished
```

RESPONDING TO CHANGE

In response to the **update:with:** message, the SmallDrawView was designed to invalidate itself whenever the changed aspect was **#add**, which would eventually lead to redisplaying its entire contents. However, in general, most changes that occur only affect small portions of the view. Redisplaying the entire picture in response to each change leads to overkill and as the number of objects in the scene increases, so does the time it takes to refresh the view.

From all the different types of changes that can occur in the SmallDraw application, there are only three distinct possibilities: (1) new objects are added to the drawing, in which case only the new objects and their handles need to be displayed, (2) the selection of objects changes, in which case only their handles need to be turned on or off, and (3) changes are made to selected objects such that a specific region of the view needs to be redisplayed, which means invalidating and redisplaying only that region.

One thing that can be done is add a little "smarts" to SmallDrawView that will speed things up considerably. ScheduledWindows have a refresh mechanism built-in such that they automatically redisplay themselves (which includes their components) whenever they are reexposed. This can happen whenever a window has been buried under other windows and is brought to the front, when a menu appearing on top of a window is closed, or when a window is resized exposing more portions of windows behind. In most cases, only subareas of the view need to be refreshed, the worst case being the entire view. In all cases, the areas needing "repair" are rectangular regions.

Views can also specify rectangular regions to be repaired. This ability will be useful in the third case described above. The following SmallDrawView instance method accepts a rectangle (in the model's scale), scales it, and expands it to compensate for handles that extend beyond an object's bounding box:

```
repairRectangle: aRectangle
    self invalidateRectangle: ((aRectangle scaleBy: self displayScale)
        expandBy: (DrawHandle extent / 2) rounded)
        repairNow: true
```

Of the three possible types of changes that can occur, the third situation describes the cases where selected objects have been translated, scaled, or one of their visual attributes has been changed (inside color, border color, line width). For translation and scaling, the region needing repair can be found by merging the object's bounding box before the change with the object's bounding box after the change (see Figure 5).

Of course, modifications to the selection apply to all selected objects so the total repair region would be the combined areas of all of the selected objects' bounding boxes. However, as it turns out, an object displayed in a view is not always guaranteed to be completely contained by its bounding box. An object can stretch beyond its bounding box if it has a thick line width or if its line segments are joined using the default join style (GraphicsContext joinMiter) as shown in Figure 6.

Thick lines can be compensated for by defining a **displayBox** method for SDGraphicObjects that simply answers the boundingBox expanded by at least one or half the receiver's line width:

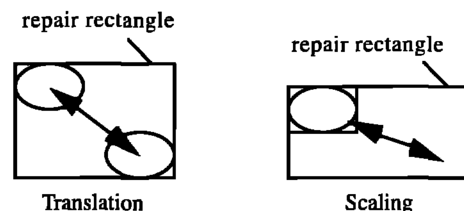


Figure 5. Repair rectangles for translation and scaling.

Figure 6. Thick lines and default join style extend beyond boundingBox.

```
displayBox
    ^self boundingBox expandBy:
        (1 max: (self lineWidth/2) truncated) asPoint
```

This displayBox can then be used for repairing regions in the view. However, it won't always enclose polygons and polylines drawn with the default join style. This can be remedied by specifying a join style that will keep lines within bounds with the following SDGraphicObject method and modification of the SDPolyline display method (see Figure 7).

```
joinStyle
    "Answer the appropriate join style for displaying the receiver."
    ^GraphicsContext joinBevel

displayOn: aGC scale: aScalePoint
    | displayPoints |
    aGC joinStyle: self joinStyle.
...
```

Now the SmallDrawView can be made to respond to each of the three change situations as follows:

```
update: anAspectSymbol with: anObject
    #add = anAspectSymbol
        ifTrue: [self displayObjects: anObject on: self graphicsContext.
            self toggleHandlesFor: anObject on: self graphicsContext].
    #selection = anAspectSymbol
        ifTrue: [self toggleHandlesFor: anObject
                on: self graphicsContext].
    #rectangle = anAspectSymbol
        ifTrue: [self repairRectangle: anObject].
```

Note that in some cases the argument, anObject, will be a collection of objects and in other cases it will be a rectangle. Corresponding **changed:with:** messages must also be inserted in SmallDraw at the appropriate places, for example:

```
addFirst: anObject
    ...
    self changed: #add with: (Array with: anObject)

selectRange: aCollectionOfObjects extend: aBoolean
    ...
    self changed: #selection with: aCollectionOfObjects
```

Finally, it will be necessary for the SmallDraw model to calculate the total display box of all the selected objects for the changes that require repairs to rectangular regions (see also the methods: **changeInsideColor, changeBorderColor, changeLineWidth**):

Figure 7. GraphicsContext join styles: joinMiter and joinBevel.

```
selectedObjectsDisplayBox
    "Answer the display box that contains all currently selected objects."
    | allObjects |
    ^(allObjects := self selectedObjects)
        inject: allObjects first displayBox
        into: [:bb :o | bb merge: o displayBox]

translateBy: aPoint
    "Translate the selected objects by aPoint, notify dependents of clean
    up region."
    | cleanUp |
    cleanUp := self selectedObjectsDisplayBox.
    self selectedObjects do: [:o | o translateBy: aPoint].
    self changed: #rectangle with: (cleanUp merge: (cleanUp
            translateBy: aPoint))

scaleBy: aPercentagePoint aboutHandleAt: anIndex absolute: aBoolean
    "Scale all currently selected objects by aPercentagePoint from each
    respective handle at anIndex, notify dependents of clean up region."
    | cleanUp |
    cleanUp := self selectedObjectsDisplayBox.
    self selectedObjects do: [:o | o scaleBy: aPercentagePoint
        aboutHandleAt: anIndex absolute: aBoolean].
    self changed: #rectangle with: (cleanUp merge: self
            selectedObjectsDisplayBox)
```

SUMMARY

In this article, we added the abilities to select, translate, scale, and modify the visual attributes of objects in a drawing to SmallDraw. This has been accomplished by modifying the definition of SDGraphicObjects to include two more instance variables (**boundingBox** and **handles**), and additional behavior such as the ability to determine whether an object contains a point or is contained in a rectangle. The SmallDraw model has the responsibility of keeping track of selected objects, the SmallDrawController determines which objects to select or modify, and the SmallDrawView can now display handles and has improved update capabilities.

SmallDraw is beginning to look more like an editor now that objects can be manipulated. The next article in this series will present additional functionality to SmallDraw such as object alignment (using a DialogView), grouping of objects, vertical and horizontal scrolling of the view, a cut/copy/paste clipboard, and support for command keys. ■

Dan Benson is a Ph.D. candidate in the Department of Electrical Engineering at the University of Washington where he is developing a 3-D spatial database for human anatomy using Smalltalk and the GemStone OODBMS. Its unique features are the ability to formulate symbolic and spatial queries specifying volumes-of-interest and the incorporation of object-oriented spatial indexing structures in the database. Besides Smalltalk, he enjoys playing jazz bass and classical piano. He may be contacted at: Department of Electrical Engineering, FT-10, University of Washington, Seattle, WA 98195, by phone at 206/ 685.7567, or email: benson@ee.washington.edu.

Errata

The following figures were omitted from Part 1 of this series, featured in last month's issue (Smalltalk Volume 1 Number 7, May 1992). We regret the error.

The source code for both the first and second parts of Dan Benson's SmallDraw application have been posted to both the Illinois and Manchester Smalltalk archives. For more information regarding these archives, see the "Best of comp.lang.smalltalk" column in Volume 1 Number 6—Eds.

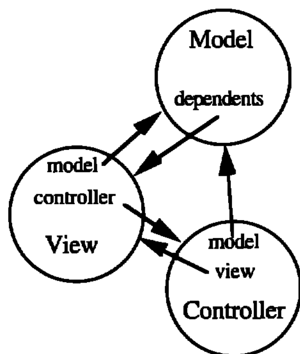


Figure 1. MVC relationships.

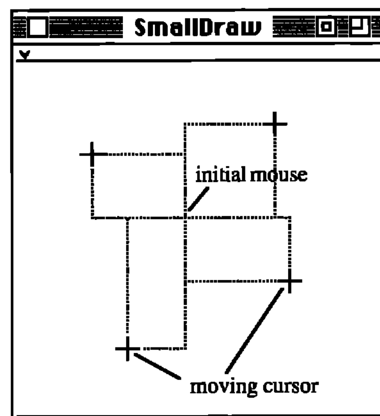


Figure 4. Rubber banding a rectangle in four directions.

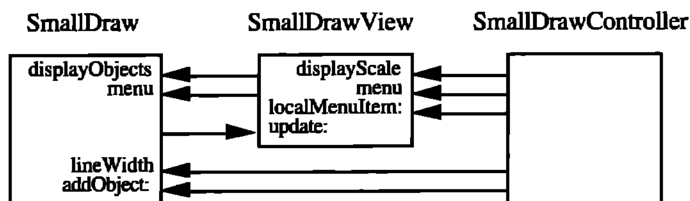


Figure 2. SmallDraw MVC interface selectors.

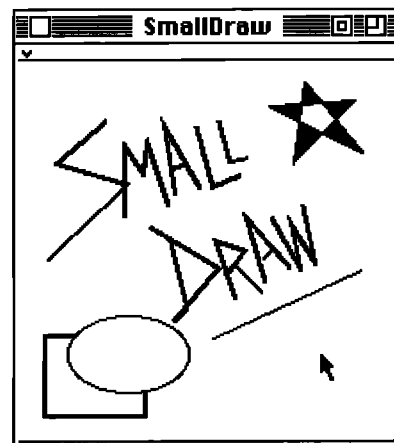
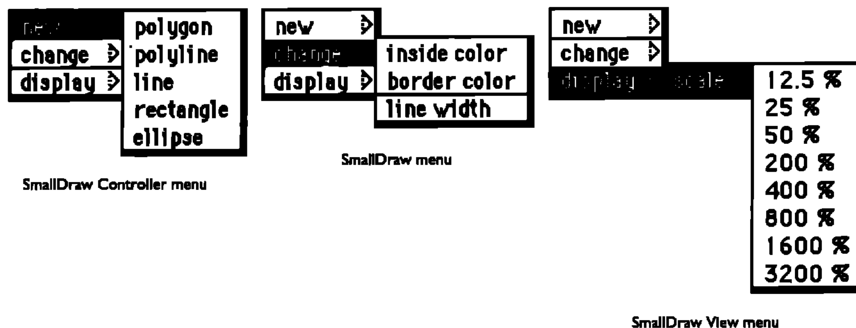


Figure 5. A SmallDraw Drawing.



SmallDraw Controller menu

SmallDraw menu

SmallDraw View menu

Figure 3. SmallDraw menus.

Kent Beck

The dreaded super

What is it with this super thing, anyway?

The pseudo-variable super is a curious wart on the otherwise unblemished Smalltalk language design. Ordinary messages are sufficient to capture composition, conditionals, and looping. The three components of structured programming thus become unified in Smalltalk. Yes, but....

The exception is super. It means the same object as self, but when you send it a message the difference is revealed. When you send an ordinary message the class of the receiver of the message is examined for an implementation of that message. If one is not found the superclass is searched, then its superclass, and so on until a method is found. Super circumvents normal message lookup. When you send a message to super the search begins in the superclass of the class in which the executing method was found.

Super is pretty benign as warts go. It makes no reference to explicit class names, so methods still contain no assumptions about what other classes are called. You cannot affect method lookup in other objects by using super, only the one currently executing. The only assumption introduced by using super is that some superclass of the current class contains an implementation of a method.

A variety of idioms, some useful and necessary, some gratuitous, have grown up around super. For example, I found that about 7% (28 out of 381) of the methods which use super in the Objectworks\Smalltalk Release 4 image could use sends to self without changing their meaning. In this article, we'll examine the uses and abuses of super. Where is super appropriate? What does it cost? What should you avoid when using it?

The code quoted in this article was taken from ParcPlace Systems Objectworks\Smalltalk Release 4. ParcPlace owns all the copyrights. In places, I have simplified the code for purposes of presentation.

ORTHODOX USE

The usual use of super is to invoke a superclass' implementation of the currently executing method. One example from Objectworks\Smalltalk is the initialization of subclasses of ValueModel. ValueModel>>initialize sets its instance variable accepted to false.

```
initialize
    accepted := false
```

The subclass ValueHolder first invokes super initialize, then sets its instance variable active to true.

```
initialize
    super initialize.
    active := true
```

This use of super segments behavior across several classes. The subclass' method depends on nothing that the superclass does. The subclass only assumes that the superclass has something useful to say about the message and makes sure it gets invoked. The "meaning" of the message in the context of the receiver is spread disjointedly across several classes.

Initialization is a common situation in which several methods which have nothing to do with one another except that they are all executing on behalf of the same object may be invoked by means of super. PostCopy, the message which cleans up a copy of an object, is another case where instance variables are manipulated without many messages, and which fits the description of segmented behavior. Using super to segment behavior across classes does not create much risk of cross-class dependencies. Even if the superclass changes, the subclass should not have to change.

A use of super which involves slightly more risk is the modification of a superclass' behavior. ChangeListView>>displayOn: is a good example.

```
displayOn: aGraphicsContext
    super displayOn: aGraphicsContext.
    self displayRemovedOn: aGraphicsContext
```

It operates by first invoking super displayOn: (which is implemented in ListView) to draw the list elements, and then drawing a line through all the list elements which have been removed. The dependency is that should ListView dramatically change its presentation of items (such as by displaying them as file folder tabs), ChangeListView displayOn: would also have to change.

Another kind of modification of superclass behavior is invoking general-purpose algorithms only when special purpose one fail. For example, ByteEncodedString>>at: is implemented by grabbing the raw byte at an index and then encoding it.

```
at: index
    ^self stringEncoding decode: (self byteAt: index)
```

ByteString, the optimized subclass, implements at: primitively.

ANALYZING USES OF SUPER

The standard image does not contain much support for searching the image for uses of super. During the preparation of this column I had to extend the system to let me flexibly search for uses of super. Here is the code I created, working from the bottom up.

InstructionStream, the object which simulates execution of Smalltalk bytecodes, has support for decoding message sends, but not sends to super in particular. I added peekForSuper, which returns nil if the method being interpreted is not about to send to super and the selector which is to be sent if it is. It is modeled after InstructionStream>>peekForSelector.

```
peekForSuper
    "If this instruction is a send to super, answer its selector,
otherwise answer nil."
    | byte x1 x2 |
    byte := method byteAt: pc.
    byte = OpXSuper
        ifTrue:
            [x1 := method byteAt: pc + 1.
            ^method literalAt: (x1 bitAnd: 31) + 1]
    byte = OpXXSend
        ifTrue:
            [x2 := method byteAt: pc + 2.
            ^method literalAt: x2 + 1].
    ^nil
```

Next, I implemented a CompiledCode method to test whether the code sends a message to super by using InstructionStream>>peekForSuper. It is similar to CompiledCode>>sendsSpecialSelector:.

```
sendsSuper
    "Answer whether the receiver sends to super."
    | scanner |
    self withAllBlockMethodsDo:
        [:meth |
        "Make a quick check"
        ((meth bytesIncludes: OpXSuper)
            or: [meth bytesIncludes: OpXXSuper])
          ifTrue:
            [scanner := InstructionStream on: meth.
            (scanner scanFor: [:byte |
            scanner peekForSuper notNil])
                ifTrue: [^true]]].
    ^false
```

SystemDictionary>>allSelect: iterates through all the CompiledMethods in the system. Now I could find out how many methods in the system send super by executing:

```
(Smalltalk allSelect: [:each | each sendsSuper]) size
```

or open a browser on all of the methods by executing:

```
Smalltalk browseAllSelect: [:each | each sendsSuper]
```

To do more sophisticated analysis of sends to super I needed more objects than just the CompiledMethod. I invented a protocol similar to SystemDictionary>>allSelect: which I called allMethods:. The Block argument to allMethods: takes three parameters instead of one: the class of the method, the selector of the method, and the method itself.

```
allMethods: aBlock
    "Answer a SortedCollection of each method that, when used
with its class and selector as the arguments to aBlock, gives a true
result." |
aCollection |
    aCollection := OrderedCollection new.
    Cursor execute showWhile:
        [self allBehaviorsDo:
            [:eachClass |
            eachClass selectors do:
                [:eachSelector |
                (aBlock
                    value: eachClass
                      value: eachSelector
                        value: (eachClass compiledMethodAt:
                      eachSelector))
                ifTrue: [aCollection add:
                  eachClass name , ' ' , eachSelector]]]].
    ^aCollection asSortedCollection
```

To complete the analysis I also needed the selector sent to super, not just a Boolean telling me whether a send took place or not. I implemented CompiledCode>>selectorSentToSuper to provide the selector.

```
selectorSentToSuper
    "Answer the selector the receiver sends to super or nil."
    | scanner |
    self withAllBlockMethodsDo:
        [:meth |
        "Make a quick check"
        ((meth bytesIncludes: OpXSuper) or:
        [meth bytesIncludes: OpXXSuper])
          ifTrue:
            [scanner := InstructionStream on: meth.
            scanner scanFor:
                [:byte || selector |
                selector := scanner peekForSuper.
                selector notNil ifTrue: [^selector].
                false]]].
    ^nil
```

Now I could find all of the methods which sent super with a different selector by executing:

```
Smalltalk allMethods:
    [:class :selector :method || superSelector |
    superSelector := method selectorSentToSuper.
    superSelector notNil & (superSelector ~= selector)]
```

Or the methods in which sends to super could be changed to sends to self by executing:

```
Smalltalk allMethods:
    [:class :selector :method || superSelector |
    superSelector := method selectorSentToSuper.
    superSelector notNil and: [class includesSelector:
    superSelector)
not]]
```

I find the creation of this kind of quick analysis tools one of the most fun things about Objectworks\Smalltalk. When I program in Smalltalk/V, with its limitations on access to system internals, I always miss the ability to quickly answer complex questions about the system.

```
at: index
    <primitive: 63>
    ^super at: index
```

Should anything occur which the primitive is not prepared to handle (for example an unknown character or an index out of range) it will fail. The Smalltalk implementation of ByteString>>at: returns the result of super at:. Thus, the special case gains a speedup most of the time, but under normal circumstances it is prepared to invoke the full available generality.

Array>>storeOn: is another example of using super to implement a specialized algorithm that is prepared to devolve into a more general one.

```
storeOn: aStream
    self isLiteral
        ifTrue: [self storeLiteralOn: aStream]
        ifFalse: [super storeOn: aStream]
```

Arrays of literals are treated specially by the compiler and can thus be printed more compactly than arrays of general objects. If the receiver is not literal (meaning some of its elements are not literals), Array>>storeOn: invokes super storeOn: so as to use the general collection storing method.

Otherwise, the receiver is printed so that the compiler can recreate the array while parsing the printed string.

DISINHERITANCE IN THE FACE OF SUPER

Using super involves a more direct reference to a superclass than a regular message send. You can get into situations where you will feel trapped by the use of super. One common pitfall is the need for disinheritance. For example, suppose we create an abstract communication object, Communicator, which sets up important state in its initialize method.

```
initialize
    "Set up important state..."
```

We then create a concrete subclass, SocketCommunicator, which uses a socket for communication. Its initialize method will send super initialize and then create the socket.

```
initialize
    super initialize.
    socket := Socket new
```

Finally we create a subclass of SocketCommunicator, TestCommunicator, which is used for testing. It reads its input from a file. Its initialize method needs the behavior in Communicator>>initialize, but it can't just send super initialize without accidentally creating a socket.

The solution is to factor the initialization of the communication channel of SocketCommunicator out of the initialize method into its own method, initializeChannel.

```
initialize
    super initialize.
    self initializeChannel
```


VOSS

Virtual Object Storage System for

Smalltalk/V

Seamless persistent object management with update transaction control directly in the Smalltalk language

- Transparent access to Smalltalk objects on disk
- Transaction commit/rollback
- Access to individual elements of virtual collections and dictionaries
- Multi-key and multi-value virtual dictionaries with query by key range and set intersection
- Class restructure editor for renaming classes and adding or removing instance variables allows incremental application development
- Shared access to named virtual object spaces
- Source code supplied

Some comments we have received about VOSS:

"...clean ...elegant. Works like a charm."
–Hal Hildebrand, Anamet Laboratories

"Works absolutely beautifully; excellent performance and applicability."
–Raul Duran, Microgenics Instruments

Logic ARTS

VOSS/286 $595 ($375 to end of February 1992) + $15 shipping.
VOSS/Windows $750 ($475 to end of February 1992) +$15 shipping.
Quantity discounts available. Visa, MasterCard and EuroCard accepted.
Logic Arts Ltd. 75 Hemingford Road, Cambridge, England, CB1 3BY
TEL: +44 223 212392 FAX: +44 223 245171


```
initializeChannel
    socket := Socket new
```

Then TestCommunicator can override initializeChannel directly.

```
initializeChannel
    socket := Filename fromUser
```

TestCommunicator may not even need to override initialize at all.

In general, the need for disinheritance points out opportunities for you to factor your code better. Remember that as you break a class down into smaller methods you make it easier to subclass later. If each method does one and only one thing, when you want your subclass to do one thing differently you will find exactly the right method to override.

EXCEPTIONS

So far the uses of super have been limited to invoking a method in a superclass which has the same name as the one currently executing. Because the names are the same, any violation of encapsulation between the subclass and superclass is limited. After all, if a message makes sense to the subclass it probably should make sense to the superclass as well. However, there are legitimate reasons to use super with a different selector than the currently executing method.

One reason to invoke super with other than the current method's selector is to avoid looping in mutually recursive methods. One example is ControllerWithMenu. In controlActivity it checks to see if the red mouse button is pressed, and if so sends itself redButtonActivity.

s·i·l·e·n·c·e
a collection of tools for project management and code delivery
• full multi-user project management
• source code version control
• automatic change documenting
• release packaging
• ship compiled code without source
• reconfigurable installation tool
• change log browser and restorer
• code performance profiling
$149.95
NEW! UFO persistent object toolkit
File any object in or out, flatten any object to a null terminated string
Available for the introductory price of $49.95
Included free with every silence purchase!
digamma solutions
Unit 6, 387 Spadina Avenue, Toronto, Ontario, Canada, M5T 2G6
Phone: (416) 351-8833 Fax: (416) 408-2850

```
controlActivity
    self sensor redButtonPressed & self viewHasCursor
        ifTrue: [^self redButtonActivity].
    super controlActivity
```

ControllerWithMenu>>redButtonActivity wants to invoke the superclass' controlActivity as its default behavior.

```
redButtonActivity
    super controlActivity
```

If the send were to self an infinite loop would result. Instead, redButtonActivity sends controlActivity to super, thus avoiding the loop.

Another reason for this kind of send to super is to avoid duplicating effort. For instance, ComposedText class>>new initializes the new instance with a new Text.

```
new
    ^self withText: '' asText
```

ComposedText also has an instance creation method which takes a Text as an argument, withText:style:. If it sent new to self the new instance would be initialized twice, once with a new Text and once with the Text passed in as an argument. To avoid this duplication ComposedText>>withText:style: sends new to super.

```
withText: aText style: aTextStyle
    ^super new
        text: aText
        style: aTextStyle
```

A final reason for invoking super with a different selector is if the subclass has different external protocol than the superclass but is implemented using the superclass' behavior. An example of this is Dictionary, which is a subclass of Set although its external protocol is much more like SequenceableCollection. Dictionary>>includesAssociation: wants to use includes: except that Dictionary overrides includes: to look only at the values, not the whole Association. IncludesAssociation: sends includes: to super to invoke the correct method.

```
includesAssociation: anAssociation
    ^super includes: anAssociation
```

This last exception is probably the least defensible of the three listed here, as subclassing solely for implementation sharing where there is little commonality in external protocol is generally a bad idea.

All of these sends to super where the selector is different than that of the current method are suspect, as they introduce the possibility of unnecessarily using super. If a message is sent to super and the class does not implement that method the message could just as well be sent to self. If at a later date you decide to override the message you can spend many frustrating hours trying to find out why the new method is not being invoked. If you find an unnecessary super don't worry. As I noted at the beginning of the article, 28 out of the 381 uses of super in the Objectworks\Smalltalk Release 4 image are unnecessary (see the sidebar for the code I used to find these numbers).

Joel Spiegel came up with the only plausible reason for using super where self would do. You might use super if you were absolutely certain that a superclass' implementation of a method had to be invoked and you didn't want to give future subclasses any opportunity to interfere. I would be interested in any legitimate examples of this kind of "prophylactic" use of super.

CONCLUSION

Super must be used carefully. It introduces a dependency between the subclass and superclass. Also, as an exception to the otherwise remarkably consistent control flow of Smalltalk, it will make your code harder to read. Correct use can significantly enhance your ability to partition behavior between classes. It can also make it easier to incrementally modify the behavior of subclasses. ■

Kent Beck has been discovering Smalltalk idioms for eight years at Tektronix, Apple Computer, and MasPars Computer. He is also the founder of First Class Software, which develops and distributes reengineering products for Smalltalk. He can be reached at P.O. Box 226, Boulder Creek, CA 95006 or via email kentb@maspar.com.

Greg Hendley and Eric Smith

Drag/Drop in Smalltalk/V PM

If you are writing applications that run under Presentation Manager chances are your clients would like the latest in "normal" user interfaces. They may not be ready for something as interesting as full direct manipulation as in the Alternate Reality Kit (now a six-year-old technology). Still, they would like something snappier than a copy/paste buffer between applications. What they want is Drag/Drop!

Drag/Drop is a protocol supported in software by a dynamic link library (DLL). As you may already know, one of the best ways to use a PM feature is to wrapper it in Smalltalk then use the wrapper. Fortunately Smalltalk/V PM has done this for drag/drop. Two example applications are even provided. One application, FileBin, uses drag/drop with list panes. The other, ContainerTester, uses drag/drop with containers.

Even with the examples, it is not necessarily clear how to use it in your own applications. Rather than give you a third example, we will discuss how to learn drag/drop and some of the issues you will need to address in using it.

LEARNING FROM THE EXAMPLES

If you just want to learn drag/drop, look at FileBin.

If you want to learn to use drag/drop with containers, look at FileBin first. Then learn to use the icon views of containers. Next, learn to use the details view of containers. Finally, learn to use drag/drop with containers. Trying everything at once can be a bit frustrating.

The mechanics of drag/drop that we need to know can be found in the classes FileBin and DragFile. In FileBin, look at startDrag:, dragComplete:, and dropComplete:. Also look at the last part of the method addFilePane. In DragFile the interesting methods, in sending sequence, are transfer:, targetTransfer:, and transfer:to:. The example does not use the method DragFile>>sourceTransfer:. For the example, it does not seem to matter what string is returned by DragFile>>renderingMechanism.

STEPPING THROUGH A DRAG/DROP

Now that you know which classes and methods we care about, let's look at how they are used. We will take a chronological tour through the dragging of an item between two instances of FileBin.

OPENING A FILEBIN

When an instance of FileBin is opened, its list pane is registered with an instance of DragDrop. The instance of DragDrop is told of its mechanisms. In this case, the dragDrop has one mechanism, an instance of DragFile.

STARTING A DRAG

The drag begins when the user holds down mouse button 2 and moves the mouse. The list pane gets a startDrag event. In the startDrag: method, drag items are created, one for each selected item in the list pane. The dragDrop is then given the collection of dragItems.

STARTING THE DROP

The drop starts when the user releases the mouse button 2 over the listPane of another instance of FileBin. At this moment, the instance of DragFile does its work. The dragFile does some checking in its method transfer:. It also checks to see if it should do a source or destination transfer. In the example, it always sends itself targetTransfer:. In targetTransfer: the dragFile sends itself the method transfer:to:, and based on the result either declares success or failure. The method transfer:to: does the actual transfer work assumed in the drag/drop operation. Here two types of transfer may be done. The file may be copied or just moved.

FINISHING THE DRAG/DROP

Given the drop was successful, the destination pane gets a dropComplete event and the source pane gets a dragComplete event. The methods dragComplete: and dropComplete: are essentially the same. They both tell the panes to update themselves so they reflect the change in their contents resulting from the drag/drop operation.

ISSUES

Several issues need to be addressed before using drag/drop in an application. What are you dragging? What do the panes represent? What does it mean drag something from one pane to another? These issues must be addressed sooner or later. Believe me, the sooner you address them the less aggravation you will experience.

HIDING INSIDE SMALLTALK

If you are like many Smalltalk programmers, you will see these as trivial issues. You are dragging an object. The panes represent collections of objects. When you drag an object from one pane to another you want to add the object to the collection represented by the destination pane. If you were doing a move instead of just a copy, then you also want to remove the object from the collection the source pane is representing. Simple.

This actually is easy to implement if the source and destination panes will always be in the same Smalltalk image. The

other restriction is you have to decide that a drag will always mean a move and not a copy.

The implementation involves trying to ignore DragFile, creating a global cut/paste buffer, and misusing the startDrag:, dragComplete:, and dropComplete: methods. Let DragCutPasteBuffer be the cut/paste buffer. Let each application have a method collection that answers the collection the list pane represents. With these givens the methods in the applications will be similar to:

```
startDrag: aPane
    | aDragItem aDragList |
    "Put the items being dragged from aPane into the buffer."
    DragCutPasteBuffer := aPane selections collect: [:selectionIndex |
        self collection at: selectionIndex ].
    "Give aPane's dragDrop a dummy drag list."
    aDragItem := (DragItem new)
        name: 'noName';
        type: #('Unknown');
        format: #( 'DRF_UNKNOWN' );
        container: 'noContainer').
    aDragList := OrderedCollection with: aDragItem.
    aPane dragDrop drag: aDragList.
dropComplete: aPane
    "Add the items dragged to aPane. Update aPane."
    self collection addAll: DragCutPasteBuffer.
    aPane event: #getContents.
dragComplete: aPane
    "Remove the items dragged from aPane. Update aPane."
    self collection removeAll: DragCutPasteBuffer.
    aPane event: #getContents.
```

We will ignore DragFile, a subclass of DragTransfer, by making our own dummy subclass of DragTransfer. This and the dummy list in startDrag: are needed so the drag and drop complete events get sent to the source and destination panes.

Create a class DummyDragTransfer as a subclass of DragTransfer. Give it a transfer: method. Use code from targetTransfer:. The method should look something like this:

```
transfer: item
    "Do not do a transfer. But do claim success."
    item pmItem
        sendTransferMsg: DmEndconverstaion
        response: DmTargetsuccessful.
    owner freeTransferItem: item.
```

Finally, copy the method renderingMechanism from DragFile. With this you should have a quick and dirty use of drag/drop that does moves between panes in the same Smalltalk image.

ADDRESSING ISSUES HEAD-ON

The more difficult use of drag/drop is its intended use, that is, to transfer items between applications. This means between panes in applications written in completely different languages. Here the object that gets transferred is a string that names the thing to be transferred. Objects do not get transferred because other languages do not have Smalltalk objects. (I admit you should be able to transfer icons, bitmaps, and other bit streams, but all the applications I have heard of stick to strings.)

SMALLTALK/OBJECTWORKS 4.0
IMMEDIATE OPPORTUNITIES
ROTHWELL INTERNATIONAL seeks Smalltalk Professionals, Min. 3 Yrs Exp. for High-Profile Positions in Houston,TX.
EXPERTISE needed in the following areas:
♦ Project Mentor ♦ Training Expertise ♦ Interface to DB2
♦ Objectworks Tool Development/Support
Rothwell International, RWI
(713) 541-0100 (800) 256-0541 F (713) 541-1167

The panes will still represent some sort of collection. However, they will be identified by strings that name the collections. Look at FileBin, ContainerTester, and DragFile. You will see the containers, sources, destinations, and item names are all file and path names. They are not the files and paths themselves.

The implication in all of this is the collections and objects to be dragged need names by which they can be uniquely identified. They also need to be accessible by their names. This could be done with a Smalltalk dictionary of all collections and objects that could be involved in drag/drop.

You may have noticed another problem now. What happens if you drag an object (actually, its name) from a pane in Smalltalk to a pane outside Smalltalk? Given a name, how does the destination pane ask Smalltalk for the object? Even if the outside pane got the object, what would it do with it if it does not know objects?

One solution would be to create a cut/paste file. The objects being dragged would be written to the file in a form both applications understand. Then the name of the file would be passed between the panes. This is similar to the solution for when both panes are inside Smalltalk. Another solution would be to pass string representations of the objects being dragged instead of their names.

A third solution is for both the Smalltalk pane and the outside pane to be views on information that exists outside both applications. The FileBin and ContainerTester examples do just this. They both access the operating system's file system. Files and directories are accessed by name through the file system. They are accessed by unique path names. In this third case it makes sense to have a subclass of DragTransfer to do the work of moving objects from one collection to another. The FileBin and ContainerTester handle this very well. ■

Greg Hendley is a member of the technical staff at Knowledge Systems Corporation. His OOP experience is in Smalltalk/V(DOS), Smalltalk-80 2.5, Objectworks Smalltalk Release 4, and Smalltalk/VPM.
Eric Smith is a member of the technical staff at Knowledge Systems Corporation. His specialty is custom graphical user interfaces using Smalltalk (various dialects) and C. They can be contacted at Knowledge Systems Corporation, 114 MacKenan Drive, Cary, NC 27511, or by phone 919/481.4000.

Alan Knight

Encapsulation and information hiding

One of the more interesting discussions this month on USENET involved the relationship between information hiding and encapsulation. The two concepts are often used interchangeably, but some claim they are separate, though related, ideas. The discussion began with a question from Vincent K. S. Oh (vincent@lancs.ac.uk), who wrote:

> **I am posting this in the hope of gathering some opinions on the exact distinction (or is there one?) between encapsulation & information hiding.**
>
> **As I understand it, encapsulation is the bundling of data and its related operations into one data structure enclosed by an imaginary boundary...Now my problem is that some authorities, like Grady Booch, have equated the two terms together. The external world communicates with the object via a public interface. Through encapsulation we get information hiding for free.... If encapsulation is OFTEN used to implement information hiding, what are the other ways to implement information hiding?**

Grady Booch does seem to equate the two concepts. In his book, Object-Oriented Design with Applications, he refers to "encapsulation—also known as information hiding." This is a strong voice in favor of unifying the concepts, but people on USENET are seldom reluctant to disagree with an authority. If the concepts are to be separated, then the definitions need to be clarified. Opposing definitions were offered. On the one hand, Larry Marshall (lmarshal@pil9.pnfi.forestry.ca) writes:

> **Possibly this oversimplifies the issue, but it seems to me that those wanting to make the distinction between these two concepts emphasize that encapsulation is an abstraction process whereas information hiding is an access constraining process.**

Earl Waldin (waldin@lcs.mit.edu) offers an apparently contradictory definition and goes back to the origins of the term information hiding:

> **The idea of "information hiding" as a principle of software engineering goes quite a ways back...**
>
> **A practical example of its use is given in: "The Modular Structure of Complex Systems" D.L. Parnas, et al. in IEEE Trans. on Software Engineering, March 1985. This article elucidates the principle thus:**
>
> **"According to this principle, system details that are likely to change independently should be the secrets of separate modules; the only assumptions that should appear in the interfaces between modules are those that are considered unlikely to change."**
>
> **In the late 70s the term encapsulation was used to refer to language constructs that supported (via enforcement) good software engineering practices, among them being information hiding.**
>
> **So, historically speaking, information hiding is a software engineering principle for organizing software, whereas encapsulation refers to language constructs. Information hiding is a more general concept that is supported...by encapsulation.**

So on the one hand, we have a definition in which information hiding refers to language mechanisms that enforce access constraints, and on the other a definition where these meanings are reversed. Read separately, either one sounds fairly plausible.

Adding to the confusion, Eric Smith (eric@tfs.com) writes:

> **The difference between encapsulation and information hiding is simple. Encapsulation means using an abstract concept to deal with complicated ideas. You stuff all the complicated ideas in to a capsule, give it a name, and from then on just think about the capsule instead of the complicated ideas it contains. The purpose of encapsulation is to reduce the external complexity of the ideas. However, all the complexity contained in the capsule is available whenever it is needed.**
>
> **Information hiding means keeping some of that information secret to make sure no one comes to rely on it. That allows you to make major changes to your software without worrying about the impact on users of previous versions, because the fact that the information is hidden proves that no one could possibly be relying on it.**

This definition seems to equate encapsulation with abstraction, but agrees with the definitions attributed to Parnas for information hiding. The discussion then goes through a number of metaphors and examples. Piles of objects in the corner of a room are encapsulated by throwing sheets over them, and it is argued whether they can be considered encapsulated if the sheets are of clear plastic. Storing files in subdirectories is also claimed as an example of encapsulation without information hiding.

Finally, William F. Ogden (ogden@seal.cis.ohio-state.edu) adds the important clarification:

Parnas' point was that you can't successfully achieve information hiding without using specifications. If you don't want a client to look at how you've implemented something, then you have to give him a satisfactory alternative explanation. In two words then, information hiding is about having a good cover story.

Encapsulation mechanisms are an important technical improvement, but judging form his examples, Parnas appears to have successfully used information hiding in a FORTRAN system. Conversely, many programmers use encapsulation mechanisms without any apparent attempt at information hiding.

This makes a very good point. I currently work in an engineering department, and I've seen some good examples of information hiding in FORTRAN code. This is achieved through clear specification and intimidation. Many of the programs perform a well-understood mathematical function using extremely ugly code. When the spec is easy to read and the code is incomprehensible, there's a strong incentive to let the implementation details remain hidden. I doubt this was quite what Parnas had in mind and it doesn't augur well for maintenance, but it does succeed in hiding information.

I've also seen code that appears designed to circumvent encapsulation mechanisms. My favorite example is the Smalltalk goody that automatically writes get and set methods for all instance variables in a class. This not only exposes the representation of the class, it is often combined with bizarre names and absence of class or method comments to produce baffling code like:

```
!Phlogiston methods!
north: aValue
    north:=aValue
```

In summary, I see the general opinion as being that the two concepts are closely related, but not synonymous. Encapsulation refers to the language mechanisms enforcing information hiding, but is also often used to describe the abstraction these mechanisms encourage. Information hiding is the software engineering principle that tries to minimize coupling between modules, and is aided by encapsulation mechanisms such as Smalltalk's restriction access to instance variables to methods within the class.

HOW CAN I GIVE AWAY MY CODE?

As I've previously described, one of the most frequently asked questions on the news groups is Where can I get free Smalltalk code? Regrettably, the question How can I contribute my code to the archives? is almost never heard.

I think most Smalltalk programmers have some favorite goodies that they would gladly share with the community. The stumbling blocks lie in not knowing how to contribute code and in not ever having gotten the time to tidy it up for release. After reading this column, you'll have lost the first excuse. As for the second, that's your responsibility, and all I can do is gently remind you that there are lots of us out there who would appreciate your efforts.

The question of contributing to the archives came up recently. Naturally, it didn't start with a would-be contributor, but with someone wondering why there is so little code for Smalltalk/V available in the archives. Mario Wolczko (mario@cs.man.ac.uk), the maintainer of the Manchester archive, responded:

We *are* interested in holding Smalltalk/V code. Although this site primarily uses Smalltalk-80, that's no reason to restrict the archive to only ST-80 code. The problem is that nobody sends us any ST/V code. In fact, to be honest, hardly anybody *sends* us anything. The code we have has come from a small number of sources:

1. **Internally produced, as part of research projects**
2. **From another archive, such as that at UIUC or ParcBench**
3. **Saved from postings to comp.lang.smalltalk**
4. **Individual contributions**

I have received very, very few individual contributions while the archive has been in operation.

This is curious, given that we are all supposed to be writing reusable software. So, if you have written some code you think should be more widely used, send it along! It doesn't have to big and fancy. In fact, some of the most popular goodies in the archive are small, simple but original ideas, expressed in just a few Kb of code.

If you want to have some code added to the archive, and you can use Internet ftp, drop it in the "incoming" directory and send me some email. If you use the autoreply server, that has its own provisions for accepting submissions.

The ftp server referred to is the one at mushroom.cs.man.ac.uk. More detailed instructions on how to submit code to the archives are available by sending email to goodies-lib@cs.man.ac.uk with the subject line "help;submit."

Ralph Johnson (johnson@cs.uiuc.edu), one of the people involved with the University of Illinois archives, adds:

I would be happy for someone to try to organize the ST/V part of the UIUC archives better. Most of the ST/V code is in the ISA (International Smalltalk Association) archives, and unfortunately ISA disbanded, and so nobody is watching out after that code anymore.

If anyone with internet access would like to volunteer to organize and upgrade the Smalltalk/V archives, contact Professor Johnson.

AUSTRALIAN ARCHIVE SITE

In other business, this month also saw the announcement of an archive site in Australia. This will also contain code from

continued on page 26...

Reviewed by Wayne Beaton

VOSS—VIRTUAL OBJECT STORAGE SYSTEM

VOSS/Windows from Logic Arts is a persistent object storage system for Smalltalk V/Windows that provides a much-needed abstract representation for secondary storage in Smalltalk.

The package I received from Logic Arts contained a beta 0.41 version of VOSS/Windows and the manual for VOSS/286. VOSS is shipped on a single diskette, and includes an 82-page manual. Installation is a simple matter of copying files to the hard drive and executing some Smalltalk code. I installed VOSS into a new image; judging from the number of classes and methods that VOSS adds to the system, installation into a clean image is highly recommended. The tutorial chapter provided in the manual touches on every feature offered without getting too technical.

The VOSS manual boasts, "The operation of VOSS is transparent to the programmer, and virtual objects require only a few special considerations." If the operation of VOSS was in fact transparent it would be an outstanding product. In many cases, the "few special considerations" require considerable code modification and a lot of thought.

WHAT IS VOSS?

VOSS builds a series of object spaces on a disk that can be used for persistent object storage. Any object in Smalltalk can be placed in a space (with some exceptions, e.g., behavior, contexts, etc.) and later retrieved. More important, the object can be retrieved by a different image, provided the object's class is known. When an object is stored, it is made into a virtual object. A virtual object is essentially a reference to an encoded and stored version. When a virtual object is retrieved, the entire "real" object must be decoded.

The VOSS Control Panel (Figure 1) provides a direct way of creating and manipulating virtual spaces. All currently active spaces are listed in the control panel. At most, one space is selected as current, indicated by an asterisk beside its name, and is the recipient of any objects made virtual. The control panel displays statistics about the space, including such information as the creation date, the number of objects stored and the amount of disk that the space occupies. The control panel updates whenever it is activated, so the information that it displays is always current.

Even with the control panel, I had some trouble opening a virtual space for the tutorial. When a new space is created, the control panel prompts for a space name and a directory path to store the space in. No error detection is performed—at one point VOSS thought that I had successfully created a space even after a walk back indicating that I had entered an illegal path name appeared. I simply "closed" the bogus space and tried again.

VOSS implements a transaction protocol similar to one a database might provide. Transaction information, such as the number of changes made since the transaction started, are displayed in the control panel. Any modifications to a space become parts of a transaction that must be explicitly committed before changes are made permanent. As might be expected, a transaction can also be rolled back to disregard changes made since it started. New transactions, started while another is open, exist within the scope of the parent transaction.

When an object is stored into a space, its reference must be remembered, or it will be unreachable. An object may be made virtual, and as such, occupies space. The reference to that object is virtual, but still an object in Smalltalk. Without explicitly remembering the object, it will be forgotten.

All spaces have a root dictionary that is directly accessible from the representation of the space. Every object must be linked, at least indirectly, to the root dictionary. The root dictionary sorts the keys that you insert into it. Since every space contains an entry with key stateDictionary, only keys that can compare themselves to strings are accepted.

As mentioned previously, when an object is stored to the space it is completely encoded. This may be a little excessive when storing large collections of objects whose contents are accessed sparingly. VOSS implements a large number of specialized collection subclasses which handle the virtual world considerably better than their "real" counterparts. When a virtual collection is accessed, it reads as much information from the space as it needs to; as its elements are addressed, they are read from the space. Virtual collections have a broad range of applications. Since a virtual collection's contents are never completely read into the memory, virtual collections can potentially contain far more objects than can be held in memory (limited of course by disk space). However, a very large collection may still run into memory problems if transactions are not handled carefully.

VOSS also implements a new type of collection that uses multiple dictionaries to reference the contents, a sort of multi-key, multi-index collection. Although the database is a clever idea, its implementation is frustrating. Look-ups into the database can access only a single key. Further, the interfaces are very nonstandard. In Smalltalk, a block is generally used to

SIXGRAPH™ Smalltalk/V users: the tool for maximum productivity

- Put related classes and methods into a single **task-oriented object** called application.
- **Browse** what the application sees, yet easily move code between it and external environment.
- Automatically **document** code via modifiable templates.
- Keep a **history** of previous versions; restore them with a few keystrokes.
- **View** class hierarchy as graph or list.
- **Print** applications, classes, and methods in a formatted report, paginated and commented.
- **File** code into applications and **merge** them together.
- Applications are unaffected by **compress log change** and many other features..

Imager — Browsers.. — Class / Application / Yarn / History; Deleted classes; Deleted methods; Code recovery; Utilities.. — Application printing and more..

CodeIMAGER™ V286, VMac $129.95
VWindow & VPM $249.95
Shipping & handling: $13 mail, $20 UPS, per copy
Diskette: ☐ 3 1/2 ☐ 5 3/4

SixGraph™ Computing Ltd.
formerly **ZUNIQ DATA Corp.**
2035 Côte de Liesse, suite 201
Montreal, Que. Canada H4N 2M5
Tel: (514) 332-1331, Fax: (514) 956-1032
CodeIMAGER is a reg. trademark of SixGraph Computing Ltd.
Smalltalk/V is a reg. trademark of Digitalk, Inc.

select a subset of a collection. The database lookup in VOSS implements a few primitive methods like for:between:and: where the sender specifies the key to select on, and the range of values to accept. A more general manner of selection might be a method named for:select: which expects the name of the key and a block specifying the selection criteria (unfortunately, the use of a block would require inspection of every object in the collection which may not be a reasonable expectation).

I tried to resist examining the code for VOSS—in any case, a full code review is beyond the scope of this review. However, VOSS provides a large number of virtual collection classes, and at some point in the future, there may well be some need to create a subclass. With this in mind, I browsed the code to see what would be involved.

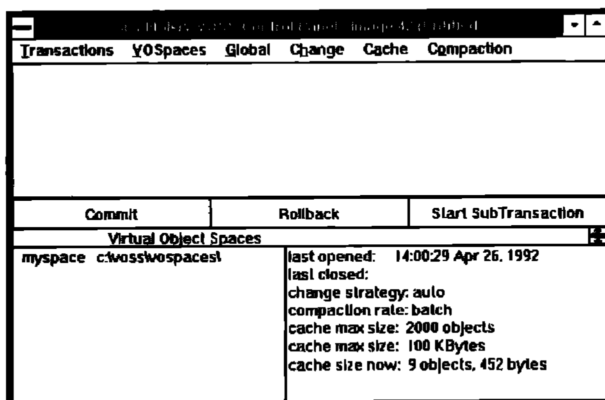


Figure 1. The VOSS control panel.

VOSS has approximately twice as many classes as it should. Each VOSS class actually has a subclass with no behavior and a short name. The class VirtualDictionary, for example, has a subclass named VD. The comments indicate that this is for efficient storage. Essentially, when objects are stored in the space, they take their class names with them. Furthermore, these shortened names are used in place of the long names in all the VOSS code. This should be enough to make any programmer used to having non-cryptic class names explode with frustration. I have to believe there is a better way to do this.

The code itself is somewhat reminiscent of C, but functional, and, for the most part, easy to follow. Creating a virtual collection subclass should be a simple matter.

The "few special considerations" the manual suggests are required to use VOSS "transparently" are a bit more of an issue than the manual suggests. VOSS spaces do not act like any other objects in Smalltalk, but do bear a close resemblance to collections or streams. A space should be a subclass of one of these with similar behavior. Perhaps the root dictionary could play a more significant role.

When objects are stored into a space, the programmer must take special care that the meaning is maintained. In one test case that I ran, I wrote an instance of Point into the space under two different names in the root dictionary. When I restarted the image, and read the values associated with the names, they were not identical. Clearly, they were identical when they were inserted. To solve this identity problem, I passed the message madeVirtual to the point before attempting to add it. Both methods worked, but with different results. The former did not preserve identity, the latter did.

Logic Arts requires a license be purchased for each image using VOSS, but offers substantial discounts for large volumes. With a purchase price of $750 US for a single copy, using VOSS can become very expensive for systems with large distributions. Free telephone support is offered, but I found no need to take advantage of it.

For small scale projects requiring limited, single-user database functionality, VOSS is a low-cost, no-frills alternative to commercial database packages such as Sybase or Oracle. However, as the size of the user base grows, so does the cost of using VOSS.

VOSS is a good product—it certainly delivers what it advertises. I recommend it for projects with small resource pools, coded entirely in Smalltalk. For larger projects, it may be more economical to custom code persistent storage for the requirements of the particular application.

For more information, contact Logic Arts Limited, 75 Hemingford Road, Cambridge CB1 3BY, England; tel. +44.233.212392; fax +44.233.245171. ■

Wayne Beaton is a senior member of the technical staff of The Object People. He has specialized in the development of graphical user interface systems using Smalltalk/V Windows and PM. Wayne can be reached at 613/230.6897.

WHAT THEY'RE SAYING ABOUT SMALLTALK

Excerpts from industry publications

ABOUT SMALLTALK

... [David] Taylor and other proponents of object-oriented programming stressed one point as highly significant: Using object-oriented languages such as C++ and Smalltalk does not guarantee that software is object-oriented. These languages are simply programming languages set up to make the mechanisms of object-oriented programming easy to execute. Procedural programs can be written in C++ and Smalltalk, while object-oriented programming can be done in procedural languages. Indeed, even simple languages like BASIC and Pascal have been given object-oriented extensions... .

Taking the "lego" approach to software design, Mitch Wagner, ***OPEN SYSTEMS TODAY**, 3/16/92*

..."The simple fact is that if reusability is your goal—if you can't find something, you won't reuse it, and if you don't know what is does in the system, you won't reuse it," said [ParcPlace's Linda] Seiner... .

The staying power of Smalltalk, ***OPEN SYSTEMS TODAY**, 3/16/92*

ABOUT OBJECT TECHNOLOGY

... Taking the object perspective during analysis, design, and programming mandates special talents, skills, and genius. Data flow techniques—the current rage—only require that observers act as bloodhounds who trace the flow of paper. Where objects are the goal, "Many are called, but few are chosen." ...For too long, we've believed that endless process, protocols and software packages could do our analysis, design and programming—and we probably always will. The OOP approach forces the industry to recognize that we have too many people, inappropriately selected and trained, with the wrong set of tools. Will we ever learn?

Letters: Where many are called and few chosen, Howard D. Weiner, ***COMPUTERWORLD**, 3/16/92*

...In addition, while high-power RISC technology is being built into PCs, video hardware such as computer-based multimedia machines are expected to become more sophisticated. Giant consumer-electronics companies like Sony Corp. are threatening to dominate the home market at the expense of multimedia PCs. Some observers believe that the motivation behind recent technology agreements between erstwhile rivals IBM and Apple has less to do with the onslaught of cheap computer clones than with trying to establish a platform to compete with the likes of Sony later in the decade... .

High Power: Next generation PCs, Michael Antonoff, ***POPULAR SCIENCE**, 4/92*

... Typically, in the computer industry, many different vendors introduce different and incompatible versions of a new technology. Proprietary market wars persist for years and sometimes decades, creating a heterogeneous mess in which users fund it impossible to integrate systems. In the emerging object-oriented Enabler environments, this shakeout has substantially occurred before there were many products... .

Advances and Research: The future of manufacturing Enablers, James E. Heaton, ***ESD TECHNOLOGY**, 3/92*

... Thus the war over whose objects to use is likely to replace the war over operating systems. And the lack of a common foundation on which to build the software that enables computers to do increasingly sophisticated tasks is likely to continue to inhibit the American computer industry. "It seems almost certain now that the battle lines are drawn," said Andrew Singer, a computer researcher at Enginuity, a high technology company based in Palo Alto, CA.

Ideas & Trends: New weapons prolong the computer wars, John Markoff, ***THE NEW YORK TIMES**, 4/5/92*

... The current wave of interest in virtual reality as well as in object-oriented programming and systems opens the door to a renewed interest in simulation. My guess is that simulation will be where some of the most exciting new PC tools and applications will emerge over the next few years... Simulation is also "object oriented," at least in the sense that it involves re-creating objects existing in the real world within the representational world of the simulated system... .

Simulation: The ultimate virtual reality, William F. Zachmann, ***PC MAGAZINE**, 3/31/92*

... The way many people try to reuse software is not reuse at all, in the view of Adele Goldberg, president and CEO of ParcPlace Systems... "Most people don't employ reuse; they do cloning. They take a copy of something and modify it in such a way that if the original is changed, the benefit of that change is not realized in the copy. This happens a lot, even in supposedly object-oriented shops," she said. The "not-invented-here" syndrome is fairly common in the development world. People are often unwilling to trust someone else's code as a component in their own application, especially when that person is not a close associate... .

... "Less than 5% of commercial information is automated," said Tom Atwood, founder and chairman of Object Design, Inc. Burlington, Mass. "This is in large part [due to the fact] that most of this information—including conversations, pieces of mail, memos, brochures or other image-based information—doesn't fit in record-oriented databases. Object databases can allow you to capture and automate a lot more of this information without having to force it into one or another format," he said... .

Object technology means object-oriented thinking, Eric Aranow, ***SOFTWARE MAGAZINE**, 3/92*

... David Taylor, principal of Taylor Consulting of San Mateo, California, and author of Object-Technology: A Manager's Guide, sees object technology leading to completely new types of enterprises. "The technology has the capability of creating a real-time organization with fully integrated workflow, distributed knowledge, and decision support systems," says Taylor. "Entire enterprises will be modeled in fully functional object-oriented software structures, greatly speeding internal and external communications. This will make the organization more adaptive, more responsive to its environment, and able to outcompete organizations still struggling with procedure-based software," he says... .

Obscure objects of industry desire, Lee Mantelman, ***INFOWORLD**, 3/16/92*

... The marketing war between object-oriented and relational databases is just beginning, with vendors of each type of system espousing different and completely opposite claims. About the only thing that relational vendors will concede is that object-oriented databases are more appropriate for some, very complicated CAD functions... .

Dueling databases divide vendors, ***OPEN SYSTEMS TODAY****, 3/16/92*

... But more important, with some modifications to the ORB specification, an OODBMS can be a more efficient object adapter for large numbers of small objects. The object adapter defined in the ORB model uses RPCs for object-client communication and thus is more suited to handling large objects such as whole spreadsheets of documents. Invoking an RPC for each object in a CAD drawing can totally obliterate the OODBMS' native performance. Thus, the [Object Database Management Group] proposes two additional object adapters to work with the ORB: the Library Object Manager (LOA), which uses an RPC the first time an object is invoked and sets up a direct link thereafter, and the Object Manager Object Adaptor (OOA), which provides a direct link from the OODBMS database to the client. These new object adapters would allow an application to work with small objects without incurring the overhead of RPCs and to access objects in multiple databases using a standard interface. This is only the first step, but I think it is critical in terms of the future viability of OODBMSes.

Data Management, Julie Anderson, ***OPEN SYSTEMS TODAY****, 3/16/92*

A five-year, $100 million program designed to increase US competitiveness in the semiconductor industry is entering the homestretch. Joint efforts among chip maker Texas Instruments, Inc. and two US defense agencies are expected to spawn revved-up chip plants based on single-wafer processes, distributed computing and object orientation by late next year. The team effort to hasten production turnaround time and gain real-time control of factory information in order to lower costs is expected to first be adopted by silicon makers and then spill over to other US businesses that are struggling to compete globally... The project seeks to accelerate chip-making cycle times using single-wafer processes—as opposed to traditional batch equipment—to quickly product low volumes of specialized chips... .

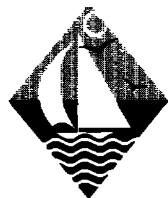
Factory automation plan nears completion, Joanie M. Wexler, ***COMPUTERWORLD****, 4/6/92*

... The interactive engine allows programmers to execute software one small section at a time, said [CenterLine's president and CEO Sesha] Pratap. "It's vital for effective object-oriented programming, because you want to be able to write several objects and execute them, and build your program by constructing objects and linking them together one at a time," he said. "This is, in fact, how most advanced programming environments are designed. It's one of the reasons why LISP was able to yield such huge productivity gains"... .

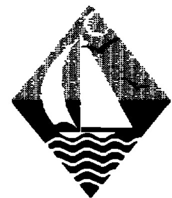
Coping with the OOP challenge, ***OPEN SYSTEMS TODAY****, 3/16/92*

... The [Distributed Management Environment] breaks new ground in software development because it utilizes advanced object-oriented software technology. OSF argues strongly that this powerful new approach to structuring software will solve the complex distributed systems management problems that users say are their primary challenge today... .

Bringing the DME into sharper focus, James Herman, ***NETWORK WORLD****, 3/30/92*



Calendar



THE SMALLTALK CALENDAR presents conferences and meetings that focus exclusively on object-oriented technology. To have a meeting or conference listed, please send the dates, conference name and location, sponsor(s) and contact name and telephone number to SIGS Publications, 588 Broadway, Ste. 604, New York, NY 10012 ph–212/274.0640, fax–212/274.0646.

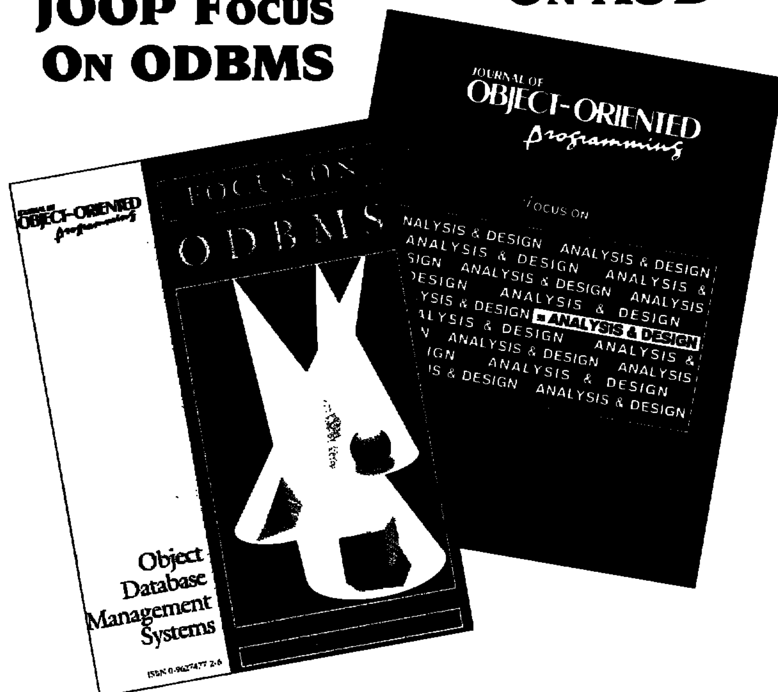
June 8–12 USENIX	June 15–19 Xhibition '92	June 16–18 SD '92	July 14–17 Object Expo Europe	July 21–23 Object World
San Antonio, TX Contact: 614.588.8649	San Jose, CA Contact: 617.621.0060	London, UK Contact: 081.742.2828	London, UK Contact: 212.274.0640	San Francisco, CA Contact: 508.879.6700

3 books to help you stay ahead of the competition.

Now Available!

JOOP Focus On A&D

JOOP Focus On ODBMS



Part of the continuing "Focus On..." series from JOOP, these two idea-packed publications are each a compilation of the most stimulating articles which have appeared in **JOOP, Object Magazine** and the **Hotline on Object-Oriented Technology** on these topics. A "must read" for anyone using this technology, the **JOOP** "Focus On..." series is the most complete source available on current ODBMS and A&D methods and techniques.

The 1992 International OOP Directory

The only comprehensive Directory available for complete listings of O-O companies, products, services, bibliographies, and market and technical research. With over 640 information-packed pages, the 1992 edition is 50% larger than the previous 1990 edition. Cross-referenced product and vendor guides by languages and category makes this guide easily accessible and easy-to-use.

JOOP Focus On ODBMS/JOOP Focus On A&D/1992 International OOP Directory

- ❑ JOOP Focus On ODBMS @ $39.00 each ________
- ❑ JOOP Focus On A&D @ $29.00 each ________
- ❑ 1992 International OOP Directory @ $69.00 each ________
  - Add $4 each for U.S. shipping/handling ________
  - Add $12 each foreign airmail/handling ________

TOTAL: $________

PAYMENT METHOD:

❑ Check enclosed (payable to SIGS *Publications*). Foreign orders must be prepaid in US dollars and drawn on a US bank.

❑ Charge my ❑ Visa ❑ MC

Card # ____________________

Exp. Date: ____________________

Signature: ____________________

Name: ____________________

Company: ____________________

Address: ____________________

City: ____________________

State: ____________________

Country: ____________________

Zip: ____________________

Phone: ____________________

RETURN TO: SIGS Publications, 588 Broadway, Suite 604, New York, NY 10012 phone (212) 274-0640 or **FAX (212) 274-0646**

...continued from page 20

CompuServe, which should help relieve the shortage of Smalltalk/V code. Hopefully these files will also become available on the other archives. The announcement was made by Robert Hinterding (rmh@matilda.vut.edu.au), who wrote:

We are in the process of setting up a Smalltalk archive here. We will mirror the archives in the USA and UK. We are interested in Smalltalk/V classes, and will make available the classes we have from CompuServe. Any submissions will be greatly appreciated. Anonymous ftp to matilda.vut.edu.au is already enabled. The mirroring of the USA and UK sites is not complete yet....

HOTDRAW FOR RELEASE 4.0

Finally, an announcement of a new piece of free software. Ralph Johnson and Pat McClaughry have recently written a version of HotDraw for Objectworks/Smalltalk 4.0. HotDraw is a framework for structured graphics editors, invented by Kent Beck and Ward Cunningham when they were at Tektronix. I'm embarrassed to admit that HotDraw is another of the many packages I've known about from some time, but have never quite gotten around to looking into. Since I'm not qualified to comment on it, I reprint from the announcement:

You want to use HotDraw if you are interested in building a program under 4.0 that manipulated a drawing like a flow chart, a circuit diagram, etc. HotDraw works well with animation and as part of a larger application. It is also a very good example of 4.0 graphics.

HotDraw, including PostScript documentation, is available from the archives at st.cs.uiuc.edu. The code is almost without restrictions, as it can be used "for any purpose, without royalties or fees of any kind, as long as credit is given to us." ■

Alan Knight is a researcher in the Department of Mechanical and Aerospace Engineering at Carleton University, Ottawa, Canada, K1S 5B6. He currently works on problems related to finite element analysis in ParcPlace Smalltalk, and has worked in most Smalltalk dialects at one time or another. He can be reached at 613/788.2600 or via email as knight@mrco.carleton.ca.

Universal Database
OBJECT BRIDGE ™
This developer's tool allows Smalltalk to read and write to: ORACLE, INGRES, SYBASE, SQL/DS, DB2, RDB, RDBCDD, dBASEIII, Lotus, and Excel.

Intelligent Systems, Inc.
506 N. State Street, Ann Arbor, MI 48104 (313) 996-4238 (313) 996-4241 fax

PRODUCT ANNOUNCEMENTS

Product Announcements are not reviews. They are abstracted from press releases provided by vendors, and no endorsement is implied. Vendors interested in being included in this feature should send press releases to our editorial offices, Product Announcements Dept., 91 Second Ave., Ottawa, Ontario K1S 2H4, Canada.

Smalltalk Nexpert Object Bridge for MS Windows

Arbor Intelligent Systems now offers a Microsoft Windows version of its bridge from Objectworks Smalltalk to Nexpert Object, an expert system shell from Neuron Data. The **Smalltalk Nexpert Object Bridge** enables developers to take a knowledge base created with Nexpert and embed it along with Nexpert's inference engine in a Smalltalk application. All objects created in Nexpert thus become available as Smalltalk objects. Expert systems created with this bridge are instantly portable across Macintosh, MS-Windows, and The X Window System without a single change to the code. Existing versions have been shipping for the Macintosh and Sun SPARCstation using Objectworks Smalltalk from ParcPlace Systems. Future versions will support Digitalk's Smalltalk/V.

For more information, contact Arbor Intelligent Systems, 506 N. State St., Ann Arbor, MI 48104; (313)996-4238, fax (313)996-4241.

WindowBuilder for Smalltalk/V PM

Cooper & Peters Inc. released the OS/2 Presentation Manager version of **WindowBuilder**, their popular Smalltalk/V user interface construction kit.

Smalltalk/V PM provides the ideal object-oriented language for corporate development, but lacks tools for rapidly designing graphical user interfaces. WindowBuilder fills this gap with a powerful point-and-click interface editor.

Using WindowBuilder, an interface is built by moving, sizing, and editing interface components directly on the screen. To promote rapid prototyping, WindowBuilder allows developers to instantly test examples of their interfaces as they are created and revise them any time in the design process. Other WindowBuilder highlights include a full-featured menu editor, a tab order editor, sophisticated component alignment and distribution tools, resource file importing/exporting, and a sizing specification editor.

WindowBuilder generates standard Smalltalk/V PM code and supports all built-in Presentation Manager graphical components. Also, WindowBuilder/V PM can import interfaces produced using WindowBuilder for Microsoft Windows, so developers who work with both platforms will be able to move back and forth easily.

For more information, contact Cooper & Peters Inc., Stanford Financial Square, 2600 El Camino Real, Suite 609, Palo Alto, CA 94306; (415)855-9036, fax (415)855-9856.

... a publication ... malltalk users.

- Smalltalk in the MIS World
- Smalltalk as a Vehicle for Real-Time and Embedded Systems
- Teaching Smalltalk to COBOL Programmers
- Interfacing Smalltalk to a SQL Database
- Realizing Reusability

Getting Real: Juanita Ewing, *Instantiations, Inc.*
Design: Rebecca Wirfs-Brock, *Tektronix*
Smalltalk with Style: Ed Klimas, *Allen-Bradley*, and Suzanne Skublics from *Object Technology International*

Plus:

- Hard-hitting product reviews
- Book and conference reviews
- Lab reviews
- Best of Smalltalk Bulletin Board
- Personality Profile

The Smalltalk Report
The International Newsletter for Smalltalk Programmers
THE COMMERCIAL EVOLUTION OF SMALLTALK

SIGS PUBLICATIONS

If you're programming in Smalltalk, you should be reading The Smalltalk Report. ***Become a Charter Subscriber!***

Charter Subscription Form

☐ **Yes,** enter my Charter Subscription at the terms indicated.

☐ 1 year (9 issues) ☐ 2 years (18 issues)

	1 year	2 years
Domestic	$65	$120
Foreign	$90	$170

Method of Payment

☐ Check enclosed (payable to *The Smalltalk Report*)
foreign orders must be prepaid in US dollars drawn on a US bank

☐ Bill me

☐ Charge my ☐ Visa ☐ MasterCard

Card # ________________ Exp. Date ______

Signature ________________

Name ________________
Title ________________
Company ________________
Address ________________
City ____________ State ______ Zip ______
Country ________ Phone ________

Mail to: The Smalltalk Report
Subscriber Services, Dept. SML
PO Box 3000, Denville, NJ 07834
or Fax: 212-274-0646

D2GG

WINDOWS AND OS/2: PROTOTYPE TO DELIVERY. NO WAITING.

In Windows and OS/2, you need prototypes. You have to get a sense for what an application is going to look like, and feel like, before you can write it. And you can't afford to throw the prototype away when you're done.

With Smalltalk/V, you don't.

Start with the prototype. There's no development system you can buy that lets you get a working model working faster than Smalltalk/V.

Then, incrementally, grow the prototype into a finished application. Try out new ideas. Get input from your users. Make more changes. Be creative.

Smalltalk/V gives you the freedom to experiment without risk. It's made for trial. And error. You make changes, and test them, one at a time. Safely. You get immediate feedback when you make a change. And you can't make changes that break the system. It's that safe.

And when you're done, whether you're writing applications for Windows or OS/2, you'll have a standalone application that runs on both. Smalltalk/V code is portable between the Windows and the OS/2 versions. And the resulting application carries no runtime charges. All for just $499.95.

So take a look at Smalltalk/V today. It's time to make that prototyping time productive.

Smalltalk/V

Smalltalk/V is a registered trademark of Digitalk, Inc. Other product names are trademarks or registered trademarks of their respective holders.
Digitalk, Inc., 9841 Airport Blvd., Los Angeles, CA 90045
(800) 922-8255; (213) 645-1082; Fax (213) 645-1306

LOOK WHO'S TALKING

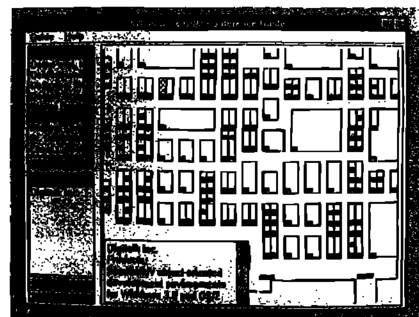
HEWLETT-PACKARD
HP has developed a network trouble-shooting tool called the Network Advisor. The Network Advisor offers a comprehensive set of tools including an expert system, statistics, and protocol decodes to speed problem isolation. The NA user interface is built on a windowing system which allows multiple applications to be executed simultaneously.

NCR
NCR has an integrated test program development environment for digital, analog and mixed mode printed circuit board testing.

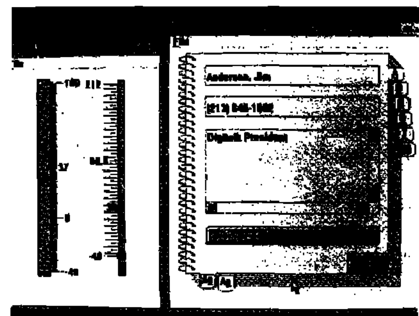
MIDLAND BANK
Midland Bank built a Windowed Technical Trading Environment for currency, futures and stock traders using Smalltalk V.

KEY FEATURES

- World's leading, award-winning object-oriented programming system
- Complete prototype-to-delivery system
- Zero-cost runtime
- Simplified application delivery for creating standalone executable (.EXE) applications
- Code portability between Smalltalk/V Windows and Smalltalk/V PM
- Wrappers for all Windows and OS/2 controls
- Support for new CUA '91 controls for OS/2, including drag and drop, booktab, container, value set, slider and more
- Transparent support for Dynamic Data Exchange (DDE) and Dynamic Link Library (DLL) calls
- Fully integrated programming environment, including interactive debugger, source code browsers (all source code included), world's most extensive Windows and OS/2 class libraries, tutorial (printed and on disk), extensive samples
- Extensive developer support, including technical support, training, electronic developer forums, free user newsletter
- Broad base of third-party support, including add-on Smalltalk/V products, consulting services, books, user groups



This Smalltalk/V Windows application captured the PC Week Shootout award—and it was completed in 6 hours.



Smalltalk/V PM applications are used to develop state-of-the-art CUA-compliant applications—and they're portable to Smalltalk/V Windows.