

The Smalltalk Report

The International Newsletter for Smalltalk Programmers

February 1994

Volume 3 Number 5

CREATING IPF HELP PANELS FOR SMALLTALK/V OS/2 APPLICATIONS: PART 1

by Marcos Lam
and Susan Mazzara

Contents:

Features/Articles

1 Creating IPF help panels for
Smalltalk/V OS/2 applications:
part 1
by Marcos Lam & Susan Mazzara

8 Cross-process exception
handling: part 2
by Ken Auer & Barry Oglesby

Columns

12 *Getting Real:*
The art of designing meaningful
conversations
by Rebecca Wirfs-Brock

15 *The best of comp.lang.smalltalk:*
Booleans
by Alan Knight

19 *Product Review:*
Tensegrity release 1.0 for
Windows and OS/2
by David Bush

W

HEN DO YOU seek help for using an application? If you are like many people, you consult documentation only when you feel confused and frustrated with the application you are trying to use. On-line help systems, at their best, can bring instant relief to the frustrations of using a new or complex application. When well written and organized and when supplied with tools for retrieving information easily, they give instant instruction on using an application. At worst, they raise your level of frustration, which caused you to seek help to begin with. On-line help that is poorly written and organized or that does not provide the right help in the right context is a hindrance rather than a help.

Many factors, including writing style, the tools used to create the help panels, and the tools used to link them to an application, make the difference between a help and a hindrance. Digitalk's Smalltalk/V for OS/2 provides classes for linking a Smalltalk application to help panels created using IBM's Information Presentation Facility (IPF). Creating truly helpful help panels for applications in Smalltalk/V for OS/2 can be a challenge, but with a clear understanding of IPF and some enhancements and extensions to Digitalk's classes, you can greatly improve the convenience and readability of your help panels. Part 1 of this article explains some of the features of IPF, how it processes help requests, and some of its essential requirements for tagging help panels. Part 2 explains the Digitalk classes that support IPF and suggests ways to enhance and extend them to create helpful help.

WHAT IS IPF?

IPF is part of the OS/2 software developer's toolkit and consists of a tag language for marking text, a compiler for formatting the marked text, and a viewing program for opening on-line documents and hooking context-sensitive help into an application. On-line documents are books, with tables of contents and indexes, presented on line instead of on paper. You display an on-line document by executing the IPF viewing program. On-line documents are not connected to an application; they are simply on-line versions of books. Context-sensitive help is displayed by selecting an item from an application's window, such as a menu or menu item, and pressing F1 to get help for that item. Once you have started the help system in this way, you can read through it like a book. IPF combines features of on-line documentation and context-sensitive help in a compiled help library. A help library is a single file that contains the help panels for an application. For on-line documents, these files have the extension .INF, and for context-sensitive help the extension .HLP. If you tag your source files properly, the same source files can be compiled into an on-line document or a context-sensitive help library.

Figure 1 is an example of an IPF help panel. The cover-page is the main help window and contains controls for the help system. It has menus and buttons for searching and browsing the help library. The help panel itself contains the text of the help system. Hypertext and hypergraphic links connect to other help panels in the library.

continued on page 4...



John Pugh



Paul White

EDITORS' CORNER

Over the past year or so, there has been a shift from discussion of object-oriented language issues to methodology issues. The "language wars" have abated in favor of the "methodology wars." Thinking positively, this has resulted in lots of discussion about important issues that otherwise might have remained ignored. More recently, there has been voluminous discussion about the merits (or otherwise) of the growing number of CASE tools that have emerged to support the various methodologies. On the front line however, the managers of large Smalltalk projects are looking for guidance on sound project management practices—a topic on which the object-oriented community is far less voluble. We have talked to enough managers charged with the task of managing medium to large-scale Smalltalk developments to recognize how strongly they feel they are entering uncharted territory and how frightening that feeling can be.

The "science" of project management has improved dramatically over the last decade. Unfortunately, most of the tools and techniques developed for traditional systems simply don't apply to Smalltalk projects. Issues like scheduling and planning, staffing and budgeting, project tracking and metrics need to be revisited when placed in an object-oriented context. Managers are being sold on a new software lifecycle but when they ask the obvious question of how to manage it, there seem to be few answers. This is, of course, great for the consulting business but not so great for the people with their necks out on the line. Advice such as "keep a good project log so you can do better the next time" is not acceptable. Hopefully, there will be lots more discussion on this topic soon. If you have thoughts in this area we would like to hear from you.

Now to this issue. In the first of two articles, Marcos Lam and Susan Mazzara take a look at the problem of creating "helpful" help systems and in particular discuss how the classes provided in Digitalk's Smalltalk/V for OS/2 can be extended and enhanced to improve the convenience of linking Smalltalk to help panels created using IBM's Information Presentation Facility (IPF).

In the second part of their article, Ken Auer and Barry Oglesby continue their discussion of problems inherent in handling exceptions across processes in a generic non-intrusive fashion.

In the "Art of Meaningful Conversations," Rebecca Wirfs-Brock describes her experiences in using the notions of use cases and system/actor conversations to guide the object modeling process. Finally, Alan Knight's watch on the Smalltalk bulletin board focuses on the recent controversy over Boolean variables.

This issue's product review represents our first look at the growing number of object-oriented database systems that may be used with Smalltalk. This month, David Bush takes a look at the Tensegrity product from Polymorphic Software.

John Pugh *Paul White*

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 Inc., 588 Broadway, New York, NY 10012 212.274.0640. © Copyright 1994 by SIGS Publications. 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 publisher.

Mailed First Class. Canada Post International Publications Mail Product Sales Agreement No. 290386. Subscription rates 1 year (9 issues): domestic, \$79; Foreign and Canada, \$94; Single copy price, \$8.

POSTMASTER: Send address changes and subscription orders to: THE SMALLTALK REPORT, P.Box 2027, Langhorne, PA 19047. For service on current subscriptions call 215.785.5996.

To submit articles, please send electronic files on disk to the Editors at 509-885 Meadowlands Drive, Ottawa, Ontario K2C 3N2, Canada, or via Internet to pugh@scs.carleton.ca Preferred formats for figures are Mac or DOS EPS, TIF, or GIF formats. Always send a paper copy of your manuscript, including camera-ready copies of your figures (laser output is fine).

PRINTED IN THE UNITED STATES.

The Smalltalk Report

Editors

John Pugh and Paul White
Carleton University & The Object People

SIGS PUBLICATIONS

Advisory Board

Tom Atwood, Object Design
Grady Booch, Rational
George Bosworth, Digitaltalk
Brad Cox, Information Age Consulting
Adele Goldberg, ParcPlace Systems
Tom Love, IBM
Bertrand Meyer, ISE
Meilir Page-Jones, Wayland Systems
Sesha Pratap, CenterLine Software
Cliff Reeves, IBM
Bjarne Stroustrup, AT&T Bell Labs
Dave Thomas, Object Technology International

THE SMALLTALK REPORT

Editorial Board

Jim Anderson, Digitalk
Adele Goldberg, ParcPlace Systems
Reed Phillips, Knowledge Systems Corp.
Mike Taylor, Digitalk
Dave Thomas, Object Technology International

Columnists

Kent Beck, First Class Software
Juanita Ewing, Digitalk
Greg Hendley, Knowledge Systems Corp.
Ed Klimas, Linea Engineering Inc.
Alan Knight, The Object People
Eric Smith, Knowledge Systems Corp.
Rebecca Wirfs-Brock, Digitalk

SIGS Publications Group, Inc.

Richard P. Friedman
Founder & Group Publisher

Art/Production

Kristina Joukhadar, Managing Editor
Susan Culligan, Pilgrim Road, Ltd., Creative Direction
Seth J. Bookey, Production Editor
Andrea Cammarata, Electronic Publishing Coord.
Margaret Conti, Production Assistant

Circulation

Bruce Shriver, Circulation Director
K.S. Hawkins, Fulfillment Manager

Marketing/Advertising

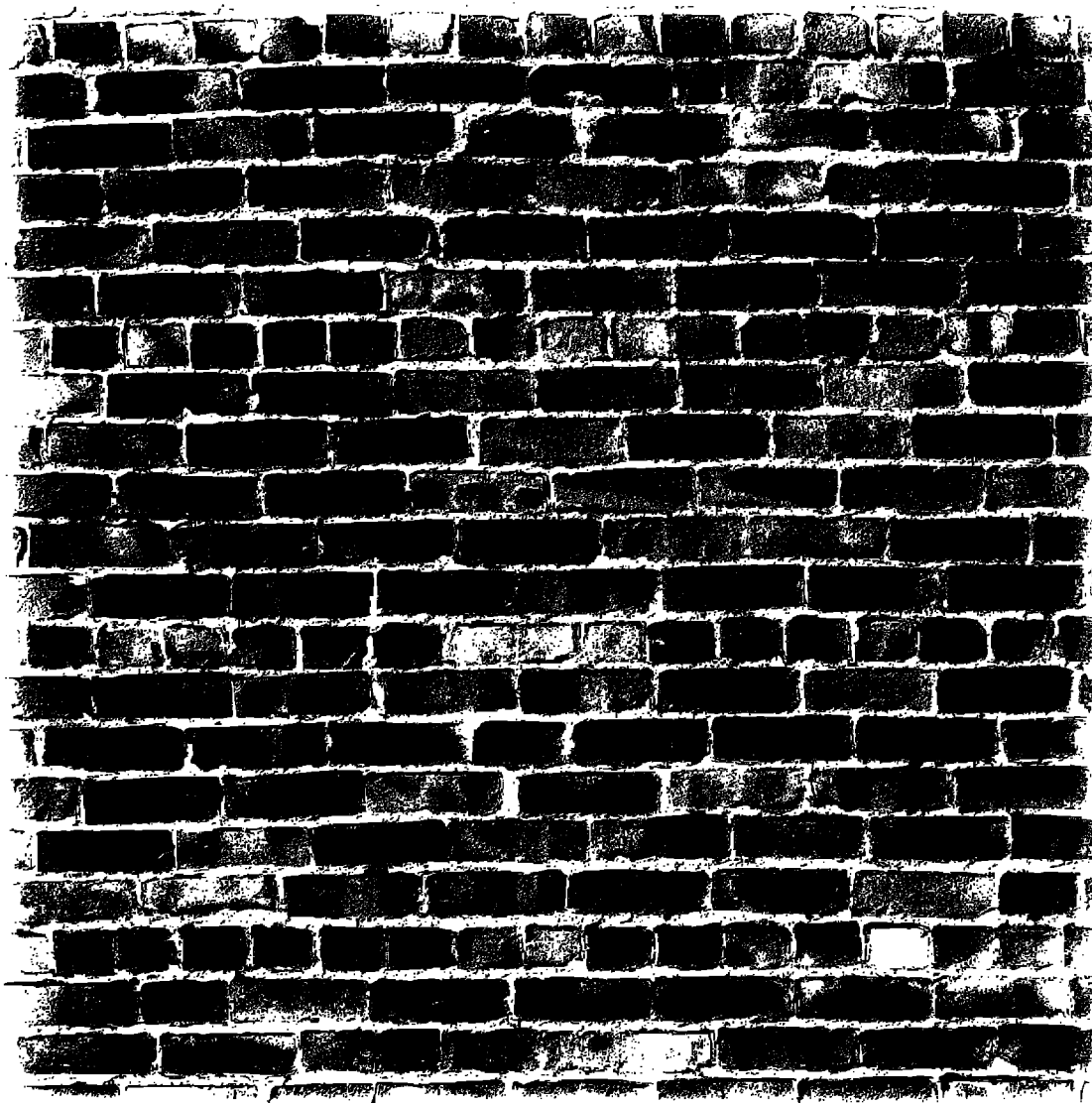
Shirley Sax, Director of Sales
Gary Portie, Advertising Mgr—East Coast/Canada
Gabrielle James, Advertising Mgr—West Coast/Europe
Helen Newling, Advertising & Exhibit Sales
Wendy Plumb, Recruitment Advertising
Sarah Hamilton, Manager of Promotions and Research
Caren Polner, Promotions Graphic Artist

Administration

William J. Ryan, Chief Operations Officer
Margherita R. Monck, General Manager
David Chatterpaul, Accounting Manager
James Amenuvor, Bookkeeper
Amy Melsten, Assistant to the Publisher



Publishers of JOURNAL OF OBJECT-ORIENTED PROGRAMMING, OBJECT MAGAZINE, C++ REPORT, THE SMALLTALK REPORT, and THE X JOURNAL.



FORCE-FIT RELATIONAL TECHNOLOGY AND YOU COULD REALLY HIT IT BIG.

Maybe you're beating your head against the relational database wall – trying to integrate your Smalltalk applications with an RDBMS. Maybe you're spending all your time debugging SQL calls instead of building great applications. Or maybe you've hit the relational performance wall because you're wasting too much processing time on object decomposition and recomposition.

Servio™ has a better way. With our high-performance GemStone® object database management system,

you can store Smalltalk objects directly in the database. We make your development time more productive and your object applications more efficient.

Learn for yourself by calling us today for a copy of "Object or Relational? A Guide for Selecting Database Technology." After all, the best way to deal with an obstacle is to avoid it in the first place.

SERVIO
OBJECT TECHNOLOGY
FOR THE REAL WORLD

Call 1 800-243-9369 for a free copy of "Object or Relational? A Guide for Selecting Database Technology."

Servio is a trademark and GemStone is a registered trademark of Servio Corporation.

Now! Automatic Documentation

For Smalltalk/V Development Teams — With Synopsis

Synopsis produces high quality class documentation automatically. With the combination of Synopsis and Smalltalk/V, you can *eliminate the lag between the production of code and the availability of documentation.*

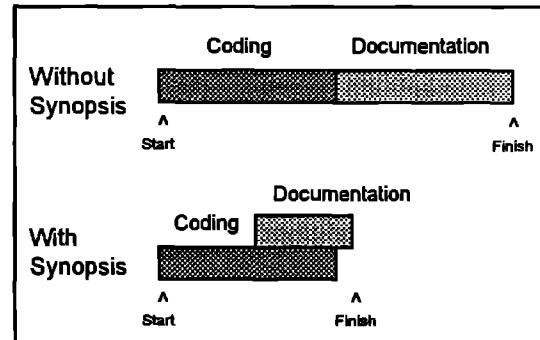
Synopsis for Smalltalk/V

- Documents Classes Automatically
- Provides Class Summaries and Source Code Listings
- Builds Class or Subsystem Encyclopedias
- Publishes Documentation on Word Processors
- Packages Encyclopedia Files for Distribution
- Supports Personalized Documentation and Coding Conventions

Dan Shafer, Graphic User Interfaces, Inc.:

“Every serious Smalltalk developer should take a close look at using Synopsis to make documentation more accessible and usable.”

Development Time Savings



Products Supported:

Digitalk Smalltalk/V

OTI ENVY/Developer for Smalltalk/V

Windows: \$295 OS/2: \$395



Synopsis Software

8609 Wellsley Way, Raleigh NC 27613

Phone 919-847-2221 Fax 919-847-0650

...continued from page 1

Aside from writing style, two factors contribute to effective help panels: how easily you can retrieve them in context and how easily you can move from one help panel to another. The first factor is largely determined by the interface between the help system and the application. The second factor is determined by the tool used to create the panels.

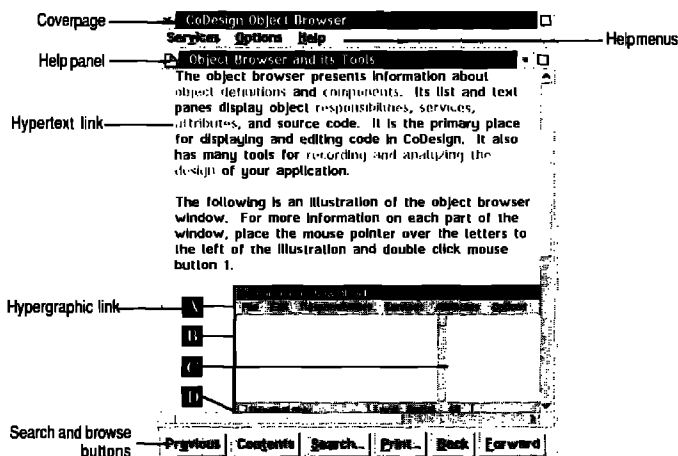


Figure 1. Sample help panel.

RETRIEVING HELP PANELS

Context-sensitive help provides help for the part of the window that has focus:

- Extended help for a window or dialog. Extended help is the main help panel for a window or dialog. It contains introductory material about an application or the current window or dialog of an application for one that has multiple views. You invoke it by pressing F1 when the window opens.
- Help for menus and menu items. Help for menus and menu items explains how to use the application's commands. You invoke it by clicking on a menu item and pressing F1.
- Help for subpanes and buttons. Help for subpanes and buttons explains the contents of the subpanes on a window or dialog and the function of its buttons. You invoke it by shifting focus to the subpane or button and pressing F1.

Processing a Help Request

To communicate with IPF to display information from a help library, an application does the following:

1. Sets up a help table and help subtable. The help table associates an application's windows with their help subtables and extended help panel ID. The help subtable associates the window's controls, such as menus, menu items, buttons, and entry fields, with a particular help panel in the help library.

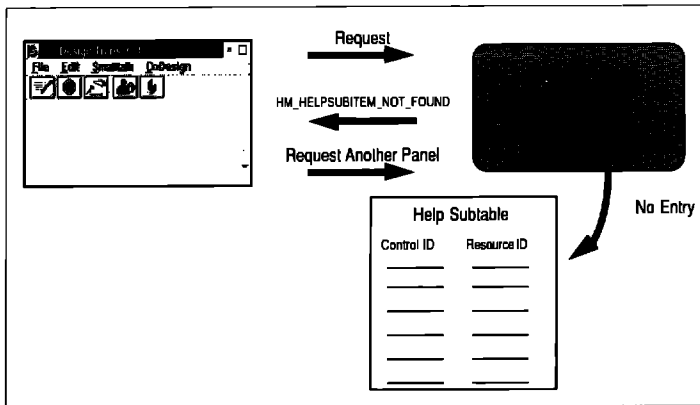


Figure 2. How IPF processes help requests.

2. Initializes the `HELPINIT` structure. The structure contains values, such as the name of the help library, that IPF requires to create a help instance.
3. Creates a help instance through the API `WinCreateHelpInstance` with the `HELPINIT` structure as one of the parameters.
4. Associates the help instance with the application window by issuing the API `WinAssociateHelpInstance` with the help instance and handle of the window as parameters.

When IPF receives a help request, it looks up the help subtable to determine the ID of the help panel that corresponds to the control for which you have requested help information. IPF then displays the help panel. This process is illustrated in Figure 2.

When IPF receives a help request for which it cannot find the corresponding help pane, it sends the `HM_HELPSUBITEM_NOT_FOUND` message to the application. (The receiver must either answer true to instruct IPF to display no panel or false to display the extended help panel.) The application can respond in one of the following ways:

- Do nothing—ignore the message and answer true
- Display its own window and answer true
- Send the `HM_DISPLAY_HELP` message to IPF to instruct it to display a particular help panel and answer true.
- Answer false to display the panel for extended help.

This process is illustrated in Figure 3.

Terminating a help instance

An application terminates a help instance by calling the API `WinDestroyHelpInstance` with the handle of the help instance. It is important that an application destroy help instances to free memory and release the link to the help libraries. Destroying help instances is particularly important during the development of a help system. Information development is typically iterative: information developers write help panels, compile them, test them, and then rewrite them. If the link between the application and the help library is not released, the help library cannot be replaced with a new version.

NAVIGATING THROUGH HELP LIBRARIES WITH IPF

To see an overview of the help library and read through a context-sensitive help library like a book, you can start with the table of contents, or page through the panels using the previous, back, and forward menu commands and buttons. Other ways of navigating through help panels are hypertext and hypergraphic links. Hypertext links connect words and phrases to other panels. Hypergraphic links connect areas of a bitmap to other panels.

Contents

You can open the contents window for a help library with a menu command or a button. The contents window lists all level-one headings and can be expanded to show any other heading levels defined in the help library. To compile a good, comprehensive table of contents for a complex Smalltalk application, all help panels for an application or a particular tool in an application must be included in a single help library. This requirement can be problematic for Digitaltalk because its help classes show an incomplete understanding of how IPF retrieves help panels and some rather simplistic assumptions about applications. These problems and how to solve them are the subject of part 2 of this article.

Hypertext and Hypergraphic Links

Text or graphics in a help panel can link to a heading or a footnote. A heading link displays another help panel. A footnote link displays a small window inside the current help panel. Links can be internal or external. An internal link displays help from the same help library as the current panel. An external link displays help from a different help library. Each type of link is described below with suggestions about how to use them in an on-line help system.

Heading links Heading links give quick access to more information about a topic or step-by-step instructions for performing an operation related to the current panel.

Footnote links Footnote links are an effective way to define terms. If your application introduces many new terms or ideas, a footnote link can make these new terms more accessible to your users.

Internal links An internal link is a link to another panel within the same help library. If the current help panel is number 2000 in the help library called `ONLINE.HLP`, for example, then an internal link might display panel number 2005 in `ONLINE.HLP`.

External Links An external link is a link to another panel in a separately compiled help library. If the current help panel is number 2000 in the help library called `ONLINE.HLP`, for example, then an external link might display panel number 3010 in `ONLINE2.HLP`.

TAGGING HELP PANELS

To create a help library, you tag the help text with IBM's stan-

standard generalized markup language (SGML) and then use the IPF compiler to format the text into .HLP files. SGML tags have the following format: `:tagname attributes`.

Some tags require an ending tag: `:etagname`.

SGML defines tags for nearly every conceivable purpose in publishing, whether in book or on-line format. IPF interprets a small subset of these tags. The tags that most concern a Smalltalk application developer are heading and linking tags because these are the ones that determine which help panels are displayed when you request help for an application.

Heading tags

IPF heading tags have a tag name and several attributes that control how the help panel is identified, formatted, and accessed. The following example shows the heading tag and attributes that play an important role in retrieving help panels. The text in italics uniquely identifies each help panel.

```
:h# res=# id=identifier global.Help Panel Title
```

This tag contains the following parts:

- `:h#` is the heading level. You can create heading levels one through six. Heading level two is subordinate to level one, level three subordinate to level two, and so on. This hierarchy pertains only to how headings appear in the library's table of contents. You can access any heading level through an F1 request.
- `res=` is the resource ID. IPF requires this numeric ID in help subtables to associate a control with a help panel and to process internal hypertext and hypergraphic links.
- `id=` is the ID. IPF requires this alphanumeric ID to process external hypertext and hypergraphic links.
- `global`. IPF requires this attribute to mark the help panel as accessible through an external hypertext or hypergraphic link.

Hypertext link tags

IPF's tag for creating hypertext links is coded differently according to the type of link you need to make in your help

panel: heading or footnote, and internal or external. The following are examples of each:

Internal heading link Internal heading link tags have the following format:

- ```
:link reftype=hd res=#.selection text:elink.
```
- `:link` is the link tag.
  - `reftype=hd` is the type of link; `reftype=hd` creates a link to another help panel.
  - `res=` is the resource ID of the panel to which we are linking. IPF requires resource IDs for internal hypertext links.
  - `selection text` is the text that is sensitive to the link request. This text is highlighted (usually with color) in the help panel.
  - `:elink.` is the link ending tag.

**External heading link** External heading link tags have the following format:

- ```
:link reftype=hd refid=identifier database='filename'.selection text:elink.
```
- `:link` is the link tag.
 - `reftype=hd` is the type of link.
 - `refid=` is the ID of the panel to which we are linking. IPF requires IDs for external hypertext links.
 - `database=` is the name of the help library containing the help panel to be linked to.
 - `selection text` is the text that is sensitive to the link request.
 - `:elink.` is the link ending tag.

Footnote link All footnote links are internal and tagged as follows:

- ```
:link reftype=fn refid=identifier.selection text:elink.
```
- `:link` is the link tag.
  - `reftype=fn` is the type of link; `reftype=fn` creates a link to a footnote.
  - `refid=` is the ID of the footnote tag to be linked to. A footnote tag has the format `:fn id=identifier.footnote text`.
  - `selection text` is the text that is sensitive to the link request.
  - `:elink.` is the link ending tag.

**Hypergraphic link tags**

IPF's tags for creating hypergraphic links serve two purposes:

1. The artwork tag imbeds a bitmap into a help panel and names a link file that defines the sensitive areas of the bitmap.
2. The link tags associate each sensitive area with an internal or external help panel or a footnote.

The following are examples of each.

**Artwork tag** Artwork tags have the following format:

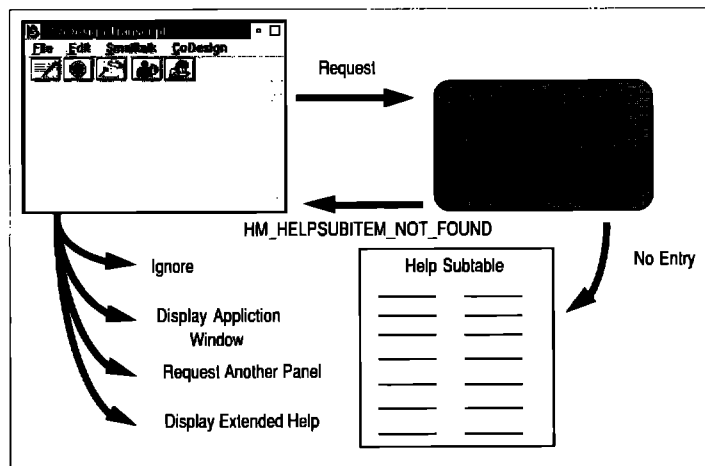


Figure 3. How IPF processes help panels it cannot find.

:artwork name='filename' linkfile='filename'.

- :link is the artwork tag.
- name= is the name of the file containing the bitmap.
- linkfile= is the name of the file containing the link tags.

**Link files** A link file contains the following tags. The link tags in this example link to an internal help panel. You can also link to external help panels or footnotes.

```
:artlink.
:link reftype=hd res=32114 x=10 y=337 cx=25 cy=25.
:link reftype=hd res=32101 x=10 y=300 cx=25 cy=25.
:eartlink.
```

Link files define areas of a bitmap that are sensitive to hyper-graphic links. The link tags have some additional attributes for defining areas of sensitivity:

- x= and y= coordinate where the sensitive area begins.
- cx= and cy= define the extent of the sensitive area along the x and y axis.

#### LIMITS OF IPF

From the perspective of application development and linking applications to IPF help libraries, IPF has some limits that, if overcome, would make IPF easier to develop with and use:

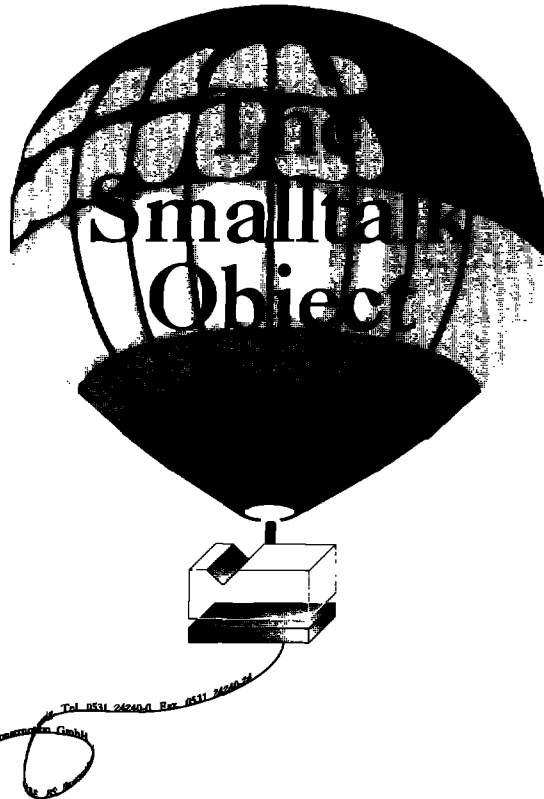
- An application cannot ask a help library if it has a certain help panel by its resource ID or alphanumeric ID.
- Whenever a help library is asked to display a help panel that it does not have, IPF always responds by displaying a message box saying "Linking not found". Since users are not generally aware that IPF is the source of the help panels, this message is cryptic, and it is not possible to prevent the message box from appearing or replace the puzzling message with a more informative one.

#### IPF AND DIGITALK

When you need to link IPF help panels to a Smalltalk/V OS/2 application, IPF's requirement to use resource IDs for context-sensitive help and internal links and IDs for external links is an important issue. Attempting to use these tags differently results in compiler errors in IPF. Digitalk's implementation of its help classes, however, requires the use of IDs for context-sensitive help. If you attempt to make an external link to a panel that also provides context-sensitive help according to Digitalk's guidelines, HelpManager returns an error. Digitalk's help classes render context-sensitive help and external links incompatible and greatly reduces your readers' options for retrieving information. Part 2 of this article explains the specific requirements of Digitalk's help classes and their limitations. ■

*Marcos Lam is a member of the technical staff, and Susan Maz-zara is a technical writer at Knowledge Systems Corporation. Both have worked on the development team for CoDesign, a Smalltalk development environment that integrates design and code. They can be contacted at Knowledge Systems Corporation, 114 MacKenan Drive, Cary, NC 27511, 919.481.4000.*

# ODBMS



## ODBMS 2.0 Smalltalk Object Management

**Client-Server Architecture  
Object Management supporting  
Versions, Transactions, Distribution  
Multimedia-Objects  
Objects to RDBMS**

**Available as  
Single User, Network and Server Version**

**Supports Smalltalk under  
Windows, Windows NT, OS/2, Unix**

**Successful applications:  
Smalltalk Team Development  
Personal Data Manager  
Configuration of Complex Systems**

### Objectoriented Technology by VC Software

**USA: VC Software, Houston TX  
v: (713) 333-8936, f: (713) 333-3743**

**Germany: VC Software Construction  
Petrivorwall 28, 38118 Braunschweig,  
v: +49 531 24240-0, f: +49 531 24240-24**

**USA: Object Power, Harvard MA  
v: (508) 456-8854, f: (508) 263-0696  
UK: Cocking & Drury, London  
v: +44 71 436 9481, f: +44 71 436 0524**



# CROSS-PROCESS EXCEPTION HANDLING: PART 2

Ken Auer and Barry Oglesby

**T**HE FIRST PART of this article (SMALLTALK REPORT 3[4]) provided some general background on the process and exception handling mechanisms as provided by the vendor. In addition, it described a problem that arises when dealing with parallel, concurrent processes, as well as our extensions to the current framework that provide a solution to this problem. In this issue we describe a problem that arises when dealing with master/subordinate processes and present our solution with some examples. Further, we provide an extensive example that incorporates both types of extensions.

## MASTER/SUBORDINATE PROCESSES

A single controlling master process often creates one or more subordinate processes to complete a single task. The master often is required to know about problems that occur in its subordinates but, once forked, a process is independent, so even if the master keeps a handle on its subordinates, the subordinates cannot communicate directly with their master. A typical scenario in machine-control software, for example, is for a machine supervisor process to create one or more subordinate processes to control the machine's resources. In the current framework, if one subordinate fails the master does not find out unless the subordinate has a direct handle on its master. If a problem occurs in a subordinate process, the resulting signal raised will only be handled by that subordinate. Its master does not automatically respond to that signal. Figure 1 demonstrates this scenario.

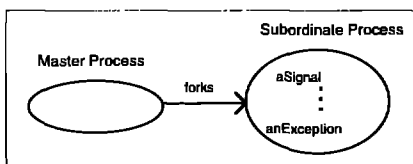


Figure 1. Master-subordinate processes (current).

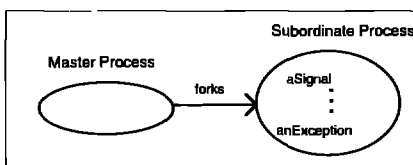


Figure 2. Master-subordinate processes (desired).

Further, because the master does not know that a subordinate failed it certainly doesn't know why it failed or what should be done about it. For example, depending upon the reason for the failure, the master might create a new subordinate to com-

plete or restart the failed one's task or, in the worst case, simply terminate its other subordinates.

This problem can be solved on a case-by-case basis by setting up semaphores or some other explicit communication between processes to communicate their progress or lack thereof. However, the master usually doesn't care about the subordinates unless something goes wrong. Additionally, the master may often need to do multiple tasks that would preclude it from either waiting on a semaphore or polling some variable every so often to find out if there's a problem. Instead, if the master process could be notified in case of exception, then no explicit communication would be required.

A subordinate process could signal an exception in the master process by interrupting it with the appropriate block. However, this would mean that the subordinate must have an explicit handle on its master accessible. Additionally, it would have to explicitly set up its own exception handlers to act appropriately.

What is desired is to have the capability to allow both the subordinate raising a signal and its master to automatically respond in the appropriate manner. In Figure 2, a signal raised in the subordinate process is handled not only by that subordinate but also by its master with no explicit communication.

## PROBLEM SOLUTION

Usually, the only time that both master and subordinate processes are known is at creation time. Once a process is executing, its knowledge of related processes is gone unless common variables hold the information. This need to maintain such variables has been replaced by a new method implemented in BlockClosure called `forkAt:forwardExceptionsTo:`. In this method, an exception handler that handles `Signal noHandlerSignal` is wrapped around the block before creating the process. At the time of invocation, the `handleBlock` encapsulates within it the master process. If an exception occurs in the subordinate process, a signal will be raised. Because no handler exists for this signal, an exception on the `NoHandlerSignal` will be created with the original exception as its parameter (see `Exception>propagatePrivateFrom:`). The exception handler defined in `forkAt:forwardExceptionsTo:` can and will handle this exception. It will do so by passing the exception as an argument to its `handleBlock`. The master process (which was encapsulated within the `handleBlock`) will be interrupted with a block raising the exception's parameter (the original exception). This will cause the normal processing of the master process to be interrupted and exception handling to begin.

Additionally, other methods have been added to BlockClosure to use this root method to allow the flexibility to create parallel and subordinate detector processes. The implementation of these methods is as follows:

`forkAt: aPriority forwardExceptionsTo: aprocess`

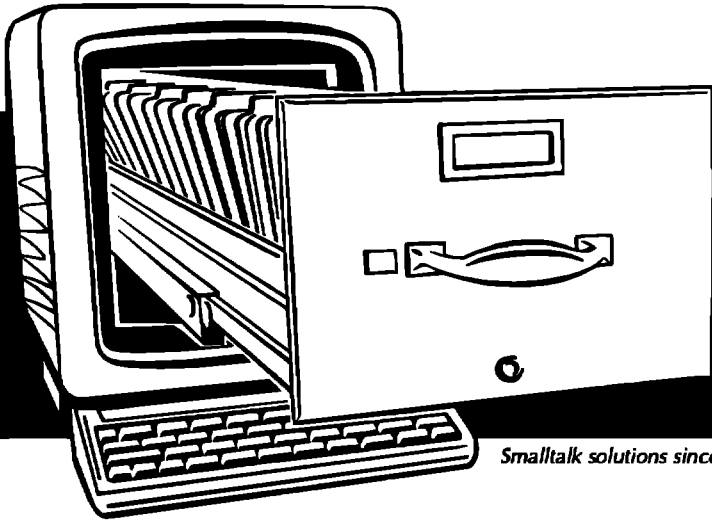
"Create and schedule a process running the code in the receiver at aPriority Any unhandled exceptions raised by this new process will be forwarded to aprocess. Answer the new process."

`^[Signal noHandlerSignal`



# REPORTOIRE

A REPERTOIRE OF QUERY AND REPORTING TOOLS FOR VISUALWORKS



Ad-hoc query and reporting  
Integrated with Visual Works  
Database or object model access  
Shared reporting repository  
Portable font and printer support

## SMALLTALK/SQL

Portable database interface  
Sybase, Oracle, Gupta, ODBC, and DAL  
Compatible with both Smalltalk V  
and VisualWorks  
Complete browsers and tools

*Smalltalk solutions since 1989*

## Synergistic Solutions, Inc.

15 Stanford Drive • Kendall Park, NJ 08824  
908.422.0450 Voice • 908.422.0901 FAX  
70233, 2017@Compuserve.com

All product names are registered trademarks of their respective owners.

```
handle: [:ex |
 aprocess interruptWith:
 [ex parameter
 searchFrom: currentprocess suspendedContext;
 raise]]
do: [self value]] forkAt: aPriority
```

**forkForwardExceptionsTo: aprocess**

"Create and schedule a process running the code in the receiver. Any unhandled exceptions raised by this new process will be forwarded to aprocess. Answer the new process."

**^self**

```
forkAt: processor activePriority
forwardExceptionsTo: aprocess
```

**forkSubordinate**

"Create and schedule a subordinate process running the code in the receiver. A handle to the master process to this subordinate is captured so that exceptions raised by the subordinate can be handled by the master. Answer the new process."

**^self forkForwardExceptionsTo: processor activeprocess**

Two simple examples are defined below.

### Example 1

This example illustrates the master process handling an exception occurring in its subordinate process:

```
Object indexNotFoundSignal
handle:
 [:ex | Transcript cr; show: 'Master handling exception']
do: [| delay |
 "Create and fork subordinate process."
```

```
[| collection |
collection := OrderedCollection with: 1 with: 2.
collection at: 3] forkSubordinate.
"Repeat master process forever."
delay := Delay forSeconds: 5.
[Transcript cr; show: 'Master process alive'.
delay wait] repeat]
```

Execution of this block of code will cause a subordinate process to be forked from a master process. The subordinate's task is to create an OrderedCollection of two elements and access the third element of this same OrderedCollection. This, of course, results in the raising of object `indexNotFoundSignal`. Because this process was forked as a subordinate, the normal execution of the master process (which is executing an endless loop) will be interrupted, and its exception handling block will be executed. This will result in "Master process handling exception" being written to the transcript window.

### Example 2

This example illustrates both the subordinate and master processes handling an exception occurring in a detector process.

```
Object errorSignal
handle:
 [:ex | Transcript cr; show: 'Master handling exception']
do:
 [| delay subordinate |
 "Create and fork subordinate process."
 subordinate :=
 [Object errorSignal
```

```

handle: [:ex |
 Transcript
 show: 'Subordinate handling exception'.
 ex reject]
do:
 [| delay |
 delay := Delay forSeconds: 3.
 [Transcript cr; show: 'Subordinate process alive'.
 delay wait] repeat]] forkSubordinate.
"Create and fork simulated detector process."
[| simulatedErrorSignal |
simulatedErrorSignal := Object errorSignal newSignal
 notifierString: 'simulated external error'.
(Delay forSeconds: 5) wait.
simulatedErrorSignal raise]
 forkForwardExceptionsTo: subordinate.
"Repeat master process forever."
delay := Delay forSeconds: 3.
[Transcript cr; show: 'Master process alive'.
delay wait] repeat]

```

Execution of this code will cause a subordinate process to be forked from a master process and a detector process to be forked with the first subordinate process as its dependent process. Additionally, an exception handler will be wrapped around the subordinate as well as the master process. Now when the simulated error signal is raised, the resulting exception will be handled first by the subordinate's exception handler. After the subordinate process is done, it is rejected, and handled by the master's exception handler. The transcript window will first show "Subordinate handling exception" followed by "Master handling exception."

**EXTENSIVE EXAMPLE**

For the exception handling enhancements presented in this article to become clearer, a more robust, concrete example will be presented. This example will embody both enhancements described above.

The example is very loosely based around a simulation of machine-control software. In broad terms, a manufacturing machine normally accepts some kind of raw material, processes it, and produces a finished product. The machine could be anything from a shop lathe, which accepts a piece of wood as the raw material and produces a baseball bat, to a semiconductor manufacturing machine, which accepts silicon wafers as the raw material and helps produce finished computer chips. To produce a finished product, in addition to the raw material, many machines require the use of another kind of material, namely a tool, to facilitate the processing of the raw material. For example, a chisel or gouge tool is necessary for a lathe to correctly shape the wood to produce the bat. Further, for the lathe to produce the bat, both the wood and the chisel must be present at the same time in the machine.

For the simulation, class `MachineResource` was added to the hierarchy as a subclass of `Object`. The responsibilities of `MachineResource` are to accept material, process it, and pass it along (either to the next `MachineResource` or out of the machine). A `MachineResource` forks several processes concurrently in order to accomplish its tasks. Among them are processes to facilitate the arrival, processing and departure of material. For

this example, the material arrival behavior will be the primary focus. The arrival behavior itself is divided into three main processes, namely the overseer or master process and two subordinate processes. The overseer coordinates the activities of its subordinates, allowing them to proceed when required and to stop when necessary. The main task of the two subordinates is to wait for material to arrive. The first subordinate waits for the raw material (e.g., the piece of wood); the second subordinate waits for the tool (e.g., the chisel). Neither subordinate knows about the other. If one fails, it performs its own exception handling to clean itself up then rejects the exception. This provides a path to its master, which performs its own exception handling.

For the purposes of brevity, implementations of many methods are left out and only trimmed-down versions of several pertinent `MachineResource` methods are included, namely the material arrival overseer process (the master process) and the raw material block (converted to the material arrival subordinate process). A partial implementation of the overseer process is described below:

```

materialArrivalOverseerprocess
"Answer the material arrival overseer process.
This process coordinates the activities of its
subordinates."

^[self class failedToArriveSignal
 handle:
 [:ex |

 Transcript cr; show: 'Overseer handling exception'.
 "real exception handling code goes here"]

do:
[Transcript cr; show: 'Overseer starting'.
"Fork subordinate material and tool processes."
self forkMaterialArrivalprocess.
self forkToolArrivalprocess.
["Wait for notification of material and tool arrival."
self waitForMaterialAndToolToArrive.
"As soon as both the material and tool arrive,
signal subordinates to begin looking for the
next material and tool."
self getNextMaterialAndTool] repeat]] newprocess

```

This process starts by sending messages, each of which results in the fork of a subordinate process. The method `forkMaterialArrivalprocess` forks the subordinate process (via `forkSubordinate`), which facilitates the arrival of the raw material. The method `forkToolArrivalprocess` forks the subordinate process (via `forkSubordinate`), which facilitates the arrival of the tool material. Once these subordinates are forked, the overseer goes into a repeat loop. In this loop, the overseer waits for notification that both the tool and material have arrived (via `waitForMaterialAndToolToArrive`). Upon receiving this notification, it allows the subordinate processes to continue (via `getNextMaterialAndTool`). This entire block of code is wrapped by a normal exception handler (via `handle:do:`), which is sent to a new signal called `FailedToArriveSignal`. If an exception occurs and `FailedToArriveSignal` is raised in the overseer process or either of its subordinates, the `handleBlock` will handle it. `FailedToArriveSignal` has two more specific signals: `Tool-`

FailedToArriveSignal and MaterialFailedToArriveSignal.

The implementation of the BlockClosure that is converted to the subordinate raw material process is as follows:

```
forkMaterialArrivalprocess
"fork the material arrival process as a subordinate process.
This simulates the arrival of the raw material."

^[self class materialFailedToArriveSignal
onException:
[:ex |
Transcript
show: 'Material process handling exception'.
"real exception handling code goes here"
ex reject]
do:
[Transcript cr; show: 'Material process starting'.
[self waitForSignalToContinue.
Transcript cr; show: 'Waiting for raw material'.
self simulateMaterialArrival.
Transcript cr; show: 'The material has arrived'.
self notifyThatMaterialHasArrived] repeat]]
forkSubordinate
```


This process repeatedly waits for a signal to continue (via wait-ForSignalToContinue), simulates the arrival of a material (via simulateMaterialArrival), and notifies all interested parties that the material has arrived (via notifyThatMaterialHasArrived). This entire block is wrapped by a new exception handler (via onException:do:) which is sent to the MaterialFailedToArriveSignal. If an exception occurs and MaterialFailedToArriveSignal is raised by any process anywhere in the system, this handler will be called upon to handle the exception. After performing its own exception handling, the handler will reject the exception to allow its master to do any additional handling.

A test interface was built so that a simulated hardware error could be easily introduced to the MachineResource. Upon notification that an error has occurred in the material handling hardware (accomplished simply via executing MachineResource materialFailedToArriveSignal raise), the exception handler for the material process (as well as any other dependent of this signal) will be called upon to handle the exception. First, the handler will print the message "Material process handling exception" to the transcript window, perform any real exception handling, and, finally, reject the exception. The material arrival overseer process will then provide any additional handling. The handler for the overseer process will first print the message "Overseer handling exception" to the transcript, then perform any real exception handling.

### CONCLUSIONS

Through some fairly simple extensions to the already powerful and flexible exception handling framework in Smalltalk, we have sufficiently addressed several problems inherent in handling exceptions across processes in a generic, non-intrusive fashion. Certainly, additional scenarios not yet raised exist in which these extensions would prove very useful. On the other hand, scenarios yet to be uncovered most likely exist in which our solutions will not apply.

Due to the openness of Smalltalk's implementation, we



**The  
Smalltalk  
Store**

405 El Camino Real, #106  
Menlo Park, CA 94025  
voice: 415-854-5535  
fax: 415-854-2557  
email: info@smalltalk.com  
compuserve: 75046,3160

| Digital                                                                                                                                                                                                                           | List     | TSS Price        |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|------------------|
| Smalltalk/V Macintosh or Windows, 16 bit API                                                                                                                                                                                      | \$495    | \$269            |
| Smalltalk/V Windows, 32 bit API                                                                                                                                                                                                   | \$995    | \$899            |
| Smalltalk/V OS/2                                                                                                                                                                                                                  | \$995    | \$859            |
| Parts Workbench, OS/2                                                                                                                                                                                                             | \$1995   | \$1729           |
| Parts Workbench, Win32                                                                                                                                                                                                            | \$1995   | \$1899           |
| <b>Objectshare</b>                                                                                                                                                                                                                |          |                  |
| WindowBuilder Pro, ST/V Win16                                                                                                                                                                                                     | \$295    | \$269            |
| WindowBuilder Pro, ST/V OS/2                                                                                                                                                                                                      | \$495    | \$459            |
| WindowBuilder, ST/V Win16                                                                                                                                                                                                         | \$149.95 | \$139            |
| WindowBuilder, ST/V OS/2                                                                                                                                                                                                          | \$295    | \$269            |
| WidgetKit/CUA '91, ST/V OS/2                                                                                                                                                                                                      | \$295    | \$269            |
| <b>LogicArts</b>                                                                                                                                                                                                                  |          |                  |
| VOSS collection, any ST/V                                                                                                                                                                                                         | \$150    | \$139            |
| VOSS DLL, ST/V OS/2 or Win32                                                                                                                                                                                                      | \$595    | \$549            |
| VOSS source, ST/V OS/2 or Win16                                                                                                                                                                                                   | \$1950   | \$1779           |
| VOSS source, ST/V 286                                                                                                                                                                                                             | \$950    | \$869            |
| <b>GSoft</b>                                                                                                                                                                                                                      |          |                  |
| MathPack 3.0, PPST                                                                                                                                                                                                                | \$395    | \$369            |
| MathPack 2.1, ST/V Win, OS/2, Mac or 286                                                                                                                                                                                          | \$125    | \$119            |
| BusinessGraph 1.2, ST/V Win, OS/2 or Mac                                                                                                                                                                                          | \$95     | \$89             |
| <b>LPC Consulting - Special through February 28, 1994.</b>                                                                                                                                                                        |          |                  |
| ODBTalk ST/V Win16 or Win32                                                                                                                                                                                                       | \$199    | \$189            |
| <b>Smalltalk Store exclusives</b>                                                                                                                                                                                                 |          |                  |
| XoterX Package Manager                                                                                                                                                                                                            |          | \$125            |
| <b>Smalltalk Starter Kit</b> includes ST/V for Win16, WindowBuilder, XoterX Package Manager, and two books - <i>Discovering Smalltalk</i> and <i>Designing Object-Oriented Software</i> . Everything you need to learn Smalltalk! |          |                  |
| Total list price for the Starter Kit is over \$895                                                                                                                                                                                |          | TSS Price: \$495 |

were able to extend exception handling entirely in the Smalltalk language itself. Additionally, the fact that everything in Smalltalk is an object, including processes, Signals, and Exceptions, allowed us to solve the problem in an object-oriented fashion, employing encapsulation, inheritance, and polymorphism in our solution.

Conceptually, these solutions could be applied to other object-oriented languages that have an open process creation and exception handling framework. ■

### References

1. ParcPlace Systems Inc. OBJECTWORKS\SMALLTALK USER'S GUIDE, 1992, Chap. 8.
2. Hinkle, B., and R.E. Johnson. Taking exception to Smalltalk, part 1, THE SMALLTALK REPORT, 2[3], 1992.
3. Hinkle, B., and R.E. Johnson. Taking exception to Smalltalk, part 2, THE SMALLTALK REPORT, 2[4], 1993.
4. Pyle, I. C. THE ADA PROGRAMMING LANGUAGE. Prentice Hall International, Englewood Cliffs, NJ, 1981.

Ken Auer is the Director of Development Services at Knowledge Systems Corporation, Cary, NC. He can be reached at 919.481.4000 or kauer@ksccary.com. Barry Oglesby is a member of the technical staff of Knowledge Systems Corporation, Cary, NC. He can be reached at 919.481.4000 or boglesby@ksccary.com.

## The art of designing meaningful conversations

**I**N MY LAST column, I introduced a framework for developing and describing use cases (see Fig. 1). *Use cases, scenarios, or scripts* are roughly synonymous terms for important ways to focus our design activities. In this column, we'll see how use cases, system/actor conversations, and supporting object designs fit together. We'll explore some issues surrounding the development of these models. We're still learning about where certain techniques work well and where they fall short. There are several helpful hints and some pitfalls. First, let's briefly review this framework.

### USE CASE FRAMEWORK

A high-level scenario is a textual description of some process or business transaction that our system must support. Large system development efforts may generate a wealth of detailed process descriptions. Less formal designs still benefit from writing high-level descriptions of desirable actor activities.

We prefer to transform these high-level descriptions into conversations before we model with objects. This forces us to separate out what the actor does and expects from our system's behavior. We purposely leave out details from conversations that should be recorded elsewhere. We omit complex conditional logic from conversations (e.g., if this happens and such and so is true and x is not, then . . .). Supporting information belongs in detailed process descriptions or other design documents. We focus on designing the flow of the conversation between the actor and our system.

There are two central parts to a conversation: a description of the actor's inputs to our system, and a corresponding description of our system's responses. Together, these "side-by-side" narratives capture a dialog between an actor and our system. We also list alternatives to the main course. These alternatives represent a reasonably complete list of conditions that object designers must be able to detect and to design appropriate responses for. We also list constraints, timing considerations, reasonable actor or system default behaviors, and other outstanding issues and desired outcomes (as we think of it) when recording each conversation. Figure 2 shows a sample conversation for beginning a session at a video kiosk.

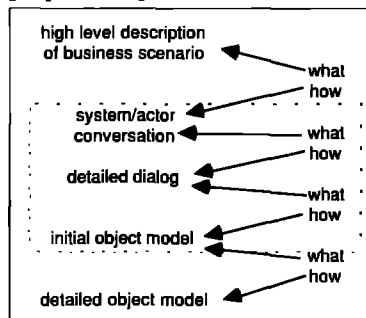


Figure 1. A use case design framework.

Conversations guide our initial object modeling activities. We develop a high-level view of key objects and their interactions for each system response side of a conversation. We start by selecting candidate objects according to their roles and stereotypes. If we already have a rough idea of our objects from previous modeling sessions, it is easier. We still review these partially completed objects and our assumptions about them before plunging into modeling. If this is the first time, we obviously spend some time picking out and discussing candidates and their possible roles.

We use whiteboards a lot during modeling sessions. We record responsibilities and collaborations on design cards. We also draw and redraw potential collaboration sequences, wave our hands, and run through consequences listing pluses and minuses and looking for better ways to distribute object responsibilities. We construct an object interaction diagram depicting what each object does and roughly how it collaborates with others. The results of all this activity are a better understanding of our design and an updated issues list!

Figure 3 illustrates object collaborations for beginning a video kiosk session. This diagram shows a collaboration sequence that supports either a regular or preferred customer. Beginning a session involves signing on and picking a transac-

|                                                                                                                                                                                                                                                                                           |                                                                        |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------|
| <b>Conversation:</b> Begin Session                                                                                                                                                                                                                                                        |                                                                        |
| <b>Actors:</b> Regular Customer or Preferred Customer                                                                                                                                                                                                                                     |                                                                        |
| <b>Overview:</b> A customer identifies herself to our Video Kiosk application. After she has signed on, she can rent, reserve, cancel reservations, or preview movies.                                                                                                                    |                                                                        |
| <b>Context:</b> The video kiosk application screen is currently displayed.                                                                                                                                                                                                                |                                                                        |
| <b>Actor Action</b>                                                                                                                                                                                                                                                                       | <b>Model Response</b>                                                  |
| Initiate Session                                                                                                                                                                                                                                                                          | Present greeting<br>Prompt customer for ID                             |
| Enter unique ID                                                                                                                                                                                                                                                                           | Receive customer info<br>Validate it<br>Prompt for permissible actions |
| Select action                                                                                                                                                                                                                                                                             | Transfer to selected action                                            |
| <b>Alternatives:</b>                                                                                                                                                                                                                                                                      |                                                                        |
| 1. User mis-enters identification<br>Sign customer on as a guest, after informing her                                                                                                                                                                                                     |                                                                        |
| 2. User's account is overdue by 30 days or more than \$50<br>Inform user and ask if she wishes to make a payment.                                                                                                                                                                         |                                                                        |
| <b>Timing Considerations:</b>                                                                                                                                                                                                                                                             |                                                                        |
| Time out and terminate session if user doesn't respond in n minutes.                                                                                                                                                                                                                      |                                                                        |
| <b>Reasonable defaults/other considerations:</b>                                                                                                                                                                                                                                          |                                                                        |
| Make it very easy for guests to sign on. They shouldn't have to identify themselves in the same way as other customers. (Note that the conversation for a guest's beginning a session may be shown in another conversation or it may not, depending on how different or important it is.) |                                                                        |

Figure 2. Beginning a session.

tion from among renting a movie, previewing, and canceling reservations, and reserving movies. Responsibilities were parceled out among Session Manager, Transaction Manager, Customer and Customer Database objects in this modeling session. Because the diagram was entered into a computer, it looks more polished than it actually should. The collaborations aren't finished. What may look like message names are rough, and more akin to responsibilities than precise message signatures. We've shown sequences of important collaborations, and also shown responsibilities that the Session Manager performs itself (collaborations 1, 2, and 5 in the diagram).

### FINDING THE RIGHT SIZE USE CASE

Finding the right size for a use case is largely a matter of personal choice. I have carefully laid out a long string of system/actor events and asked colleagues and design students to split them into meaningful units. We all try to follow these guidelines:

- Look for meaningful sequences that form some namable subtasks.
- Don't look for "reusable chunks" to start.
- Find an all-encompassing name to call this activity.
- Clearly name the subtask using a single action name.
- Look for loops or repetitive event sequences.

How people group tasks together varies widely. Some have a hard time justifying their decisions ("it just feels right" *isn't* a justification). It really is a matter of how much or how little detail you can handle, and what hidden assumptions you are making about the underlying complexity of the task. Working in a team, over time, the group as a whole usually develops a collective sense of the right size for a use case. The above guidelines are useful hints, not rules.

### DULL CONVERSATIONS

Sometimes conversations that you generate from use cases are too simplistic. This is often the case with event-driven systems. Picking an item off a menu doesn't often generate an interesting conversation, either. If you have this problem, look at grouping more tasks together. Perhaps your application isn't very conversational by nature. An actor might initiate a lot of complex tasks for your system to work on, but the interactions to get them

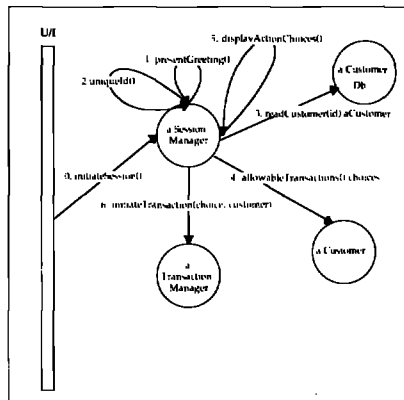


Figure 3. Object collaborations for beginning a video kiosk session.

started may be pretty straightforward. A command-driven monologue isn't that interesting. Don't force conversations.

### DEALING WITH SLIGHT VARIATIONS ON A THEME

Actors can be users or other systems. Further distinctions can be made between actors.

# EC-Charts

Just touch a button to put a chart view in your window!

## Add charts to your VisualWorks palette

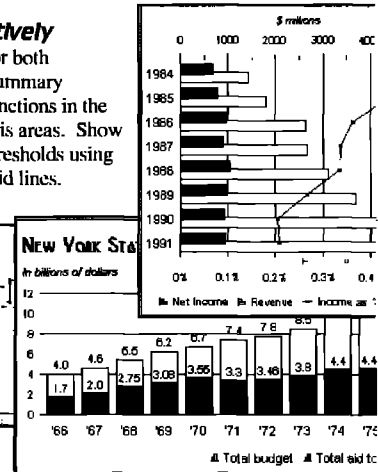
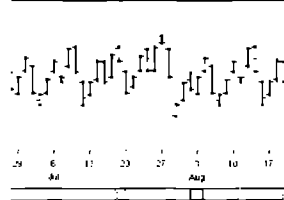
**Dynamic** Add or change data points, with minimal screen repainting. Add or remove data series to/from the chart.

**Interactive** Select data points with the mouse—EC-Charts informs your application.

### Uses screen space effectively

Scroll the chart view in one or both directions. Mark values of summary

functions in the axis areas. Show thresholds using grid lines.



Now only \$350

No runtime license fee

Call for a technical paper on EC-Charts

East Cliff Software

(408) 462-0641

VisualWorks is a trademark of ParcPlace Systems, Inc.

21137 East Cliff Dr · Santa Cruz · CA 95062

I've seen applications that have dozens of different types of users and dozens of external systems to which they have to connect. Different actors may need or are privileged to conduct slightly different dialogs. Even when performing even the same task, different actors often need to have slightly different conversations. Wading through all this takes a lot of time. The bottom line is that unless a conversation or alternative radically differs from a previously documented one, I find it perfectly acceptable just to record the differences. Unless it is that different, I don't bother building a collaboration for each slight variation, either. My goal is to produce and build designs, not generate lots of paper to wade through. Consequently, I make sure to record new responsibilities and collaborations on design cards, but I don't necessarily show a lot of other supporting diagrams if it isn't warranted.

### WHAT REALLY GOES ON DURING MODELING?

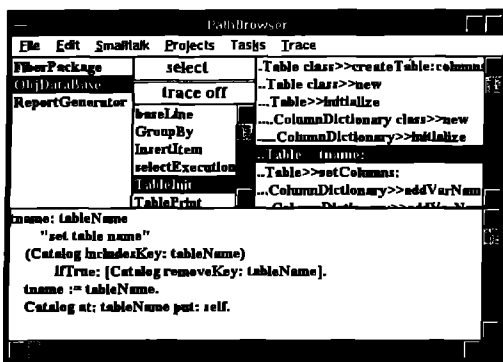
I can't ignore the big picture while designing the objects for a particular conversation. I don't know anyone who can. Too many side excursions into the weeds can be frustrating. However, remembering consequences of prior modeling decisions and checking for possible inconsistencies is extremely important. Consolidating and unifying design is a constant background mental process that should be allowed to bubble to the surface at times.

For example, if I know what Session Manager should do, I don't park my understanding of its other duties elsewhere while designing how to begin a kiosk session. If I did that, I might add to its pre-existing responsibilities in an inconsistent way.

### PathBrowser

— a trace and documentation tool for Smalltalk

Eliminate the need to bring up multiple windows to follow an execution path. By using the *PathBrowser* you can quickly browse the message flow an event creates across multiple classes, from a single window. Select the classes to be traced, or have the *PathBrowser* automatically generate the class list based on a primary class. Then you need only enable the trace, create the event, & voila!



#### BrowseIt Software

PathBrowser for Digitaltalk (Win OS2 Win32) \$99 Tel: (303) 730 - 0806  
 PathBrowser for ParcPlace VisualWorks \$149 Fax: (303) 730 - 0812  
 Site licenses & educational discounts. Money back 30 day guarantee

### SHOWING MODEL RESPONSES

Typical object collaboration diagrams look either deceptively simple or overly complex. They don't reflect the thought or effort that went into producing them. To understand their significance, it is necessary to know what are key objects and what level of detail the designer intended to model. To find this information, we need to look at supporting documents such as emerging class designs, prior conversations, and other supporting evidence. Designers can also help others through their design. Diagrams alone can't show the significance of key design decisions. This information is hard to find in class specifications, too. That's why designers need to tell us what they considered to be important.

For example, our Customer Database object isn't a simple-minded database interfacier. It doesn't just humbly reconstitute customer objects from stored information. Sure, it encapsulates details of some relational database interface objects. But it is really smart. It detects whether a customer is typical, preferred, or a guest and dynamically builds the right kind of Customer object based on a number of decisions it must make (e.g., is the customer late paying her bill, how many rentals has she made recently, etc.) You can't see those interior details unless we show the Customer Database object in much further detail on this diagram. However, when we were focusing on the high-level objects and their interactions, we didn't want to clutter up our diagrams with too many lower-level details.

### HOW DETAILED SHOULD AN OBJECT DIAGRAM GET?

I used to have a problem showing message sends to self. It seemed like too much detail too early. Now it doesn't bother

me if it is done sparingly. Designing an object involves deciding what it does itself as well as all of its collaborations. If you haven't internalized that you need to send messages to self to create well-designed objects, you need to show this detail. If you know this, but your peers still find it confusing, you probably need to show them that detail. Diagram details are a matter of personal discretion, design team standards, and need to comprehend. Either too much clutter or too few details can cause problems.

One important thing to note about our begin session collaboration diagram is that we showed interactions with the user interface as dashed lines. We knew which objects collaborated with the user interface, but we didn't how they collaborated. We didn't show this detail on purpose. We did this for several reasons. We wanted to first concentrate on the interactions between business objects, controllers, service providers, information holders, and the like.

If we ignore the interface details for just a bit, we can plug our design into a variety of different user interface solutions after understanding our object model. We wanted to give ourselves the freedom to explore potential user interface designs as a separate activity.

We also didn't want to overly constrain our design with user interface requirements. Employing this tactic prevents us from embedding a detailed, quirky understanding of particular user interface objects. I personally don't have a big problem with setting aside the user interface details until I know what the other objects generally have to do. I advise you to worry about the details of that after the interplay between other objects settles down.

Following this strategy causes some feedback and tuning of both our conversation and the objects that interact with the user interface when we actually do attend to those details. If we limit the number of objects that actually interact with user interface objects (which is always a good design principle to follow), the impact of this fine tuning can be minimized.

### CONCLUSION

There are open issues about use cases, conversations, and object modeling. However, there is a big payoff in designing this way. Use cases and conversations guide design activities. Tying our model back to conversations, use cases, and supporting documentation is one big step in the right direction. My goal in applying these techniques is always to enhance my abilities to communicate with non-object experts and improve my ability to produce the right design solution to the right problem. ■

*Rebecca Wirfs-Brock is the Director of Object Technology Services at Digitaltalk and co-author of DESIGNING OBJECT-ORIENTED SOFTWARE. She has 18 years experience designing, implementing, and managing software products. For the last nine years, she has focused on object-oriented software. She managed the development of Tektronix Color Smalltalk and has been immersed in developing, teaching, and lecturing on object-oriented software. Comments, further insights, or wild speculations are greatly appreciated by the author. She can be reached via email at rebecca@digitaltalk.com. Her U.S. mail address is Digitaltalk, 7585 S.W. Mohawk Drive, Tualatin, OR 97062.*

## Booleans

This month's column focuses on the recent controversy over Boolean variables. That's right, Boolean variables. If you don't think true and false are the stuff of impassioned debate, read on.

### IFNIL

Jack Shirazi (js@biu.icnet.uk) started the debate by asking:

How many times have you written

```
something isNil ifTrue: block1 ifFalse: block2.
```

How come there isn't a ifNil:ifNotNil: method as standard?

Several people explained how to write ifNil:ifNotNil: and what a wonderful language Smalltalk is because it allows you to write your own control structures. He then explained further:

I am obviously aware that I can add the method! ;-) It was specifically the loss in performance that concerned me ...

Why isn't it there and optimized like the other ones.... It is the only choice method that I can think of, which would be used as much as ifTrue:ifFalse:

It's certainly true that a lot of Smalltalk's conditional statements are used to test for nil values. Mario Wolczko (mario@cs.man.ac.uk) provided some quick statistics.

I counted how many methods in a standard 4.1 image include code like

```
a isNil ifTrue:
x == nil ifFalse:
nil ~ z ifTrue:
```

and so on. That is, how many methods could be written more concisely if ifNil:, notNil:, ifNil:notNil:, etc., were available.

Out of 8,494 methods in the image, 704—i.e., 8% or 1 in 12 of all methods—had such a test. Seems like things would be a lot cleaner if it were to be added and used.

He also did a small experiment on the relative performance of user-added control structures.

Numbers taken from a Sun IPC, repeating 1,000,000 times:

```
|b| b := nil. nil isNil. b yourself "to establish a baseline" 2,339
|b| b := nil. nil isNil ifTrue: [b yourself] 2,792
```

```
|b| b := nil. nil ifNil: [b yourself] 13,912
```

(with the obvious definition of ifNil: in UndefinedObject)

You pay a high price for really building the block and sending the message.... Of course, you could always modify the compiler to treat your construct specially, too—this is left as an exercise for the interested reader :-)

Jan Steinman (jan.bytesmiths@acm.org) proposes a slightly different syntax. He writes:

One of the more common usages of nil tests is to guard against passing that nil along the way. Many times I've wanted the equivalent of the C ?: operator, which, given bool ? trueVal : falseVal returns one or the other. So I implemented ?. I chose a binary selector so that it could be easily placed "in-line" without parentheses, and I added the block test so that you can do things of arbitrary complexity in the "not nil" case.

He provides code for a ? operator, which allows expressions like the following:

```
variable ? defaultValue
self keyword: SomeClass defaultInstance ? 'hi there'.
foo ? [foo := 'initial value']
```

So far, this hasn't been too controversial. Few people would object to a few extra control structures to make dealing with nil easier. Kent Beck even discussed the ifNil: construct back in the February 1993 issue of THE SMALLTALK REPORT. A ? operator looks a little too much like C for some people's taste, but they're free to call it something more Smalltalk-ish or not to use it at all. The controversy started when SmalltalkAgents actually implemented many of these ideas in a way that left many people unhappy.

### SmalltalkAgents

In case you hadn't heard, there's a new Smalltalk implementation for the Mac, called SmalltalkAgents, and produced by a company called Quasar Knowledge Systems (QKS). This new implementation received a lot of attention on USENET for two reasons. First, it departs significantly from traditional Smalltalk implementations in many areas, and the design choices have

generated a lot of discussion. Second, David Simmons (whose signature describes him as the Chief Technical Officer of QKS), has posted several lengthy articles describing and justifying those choices.

Note that this article deals with one minor design choice and the discussion it generated on USENET. This is not representative of the general discussion on SmalltalkAgents, and should not be taken as any kind of a review. If you want a review, see Jan Steinman's article in the November/December 1993 SMALLTALK REPORT.

Since I don't have access to a Mac at the moment, I haven't been able to try out SmalltalkAgents yet. I am, however, absolutely delighted to see a new Smalltalk implementation. There hasn't been enough meaningful competition in the Smalltalk market so far, and the existing vendors have been able to get away with far too much for far too long.

### ifTrueOrAlmostAnythingElse:

What was the design choice that generated such controversy? Alun ap Rhisiart (vollrath@vax.oxford.ac.uk) describes it as follows:

STA has adopted C's concept of Booleans. `ifTrue:[]` will execute the true block for any receiver that is not false, nil, or zero.

```
35 ifTrue:[Speakers beep]
```

will beep. Now suppose you want to protect yourself from errors where an object (almost any object) has been passed, but you are expecting a Boolean. You might have something like this

```
param
 ifTrue: [true block]
 ifFalse: [false block]
```

now you have to write

```
param == true
 ifTrue: [true block]
 ifFalse: [false block]
```

Since when you write `ifTrue:` you normally mean just that, it is tedious and error prone to have to keep in mind that this is not really a truth test, in spite of what it says. Tests of truth value, and tests of existence (instantiation) are semantically different, and C simply has this wrong. Occasionally, you can drop a message send by writing object `ifTrue:[]` instead of object `isNil ifTrue:[]`, but you have to pay heavily for that by having a difficult time tracking down a class of bugs, which should have raised an exception right away.

I have to say that my first reaction to this is to agree completely with Alun. Booleans aren't numbers. Numbers aren't Booleans. UndefinedObjects aren't anything. Mixing them all up can only lead to trouble. This is particularly true in a dynamically typed language, where you rely on strict message protocols to catch

stupid typing errors. One of my most common stupid mistakes is to write a method that should return a value and then forget to return anything, returning the receiver by default. This is usually caught the first time you call the method, but wouldn't be with an implementation like this.

### isNilOrZeroOrFalse

This is not the only change. At least `ifTrue:` and its relatives are defined as errors for non-Booleans. Extending them will only allow code that would otherwise crash. SmalltalkAgents also redefines `isNil` in the same spirit.

```
0 isNil ==> true
nil isNil ==> true
false isNil ==> true
anythingElse isNil ==> false
```

This is much worse, in that it can easily break existing code. For example, user-input mechanisms (prompters, input fields, etc.) that expect the user to enter a number, might return nil for an illegal value. If the value is checked using `isNil`, illegal values can't be distinguished from the number 0.

Another opinion. Rik Fischer Smoody (riks@cse.ogi.edu) writes:

I remember systems that can't tell the difference between 0, nil, and false. (One such system is still in widespread use.)

Let's not go back.

I am in favor of adding `ifNil:` to the lexicon of Smalltalk. It is not instead of `ifTrue:`, but in addition. Semantics is related, not equal.

### OTHER LANGUAGES DO IT THAT WAY

C isn't the only language with ambiguous Boolean semantics. For one thing, there are probably lots of other weakly typed languages close to the hardware whose conditionals are based on bit patterns rather than logic. On the other hand, there is LISP. This is a language that is in many ways similar in philosophy to Smalltalk and had a lot of influence on Smalltalk's early design.

LISP has conditional statements that take any object, and use nil (which is also the empty list) for false. Any other object counts as true. There may or may not be distinct true and false objects. LISP doesn't normally count 0 as false, but it does count the empty list, which might be even worse when generalized to Smalltalk. I shudder to think of `isNil` implemented as `^self isEmpty`.

In fact, when I checked which objects counted as false with a LISP user (Andrew Rau-Chaplin, arc@dimacs.rutgers.edu) he described it as:

One of the most implementation-specific issues in LISP, and in my experience one that causes some of the most devilish compatibility bugs.

Apple's new object-oriented language Dylan, which is very strongly influenced by LISP, has a special distinguished object for false, and treats all other objects as true. Dylan doesn't



seem to have a nil, though, and it uses false in many places where Smalltalk would use nil. It does support the slot-initialized function, which tests for uninitialized variables (which would be nil in Smalltalk). They also warn that it may be an expensive operation in many implementations, and that there is no portable way of setting a slot to the uninitialized state once it has been given a value.

---

“  
You could always modify the compiler to treat your construct specially, too—  
this is left as an exercise for the interested reader.”

---

Marks Stumptner (mst@vexpert.dbai.tuwien.ac.at) points out that:

Booleans have a clear specification and mixing them in with everything else should not be done without a very good reason. Note that the LISP/Scheme way of dealing with #true/#false/nil predates their object-oriented extensions. The very fact that they do have a “real” false that can be tested for while being “mostly” the same as nil shows that there’s something fishy here.

#### IN DEFENSE OF SmalltalkAgents

I’ve given a lot of space to those criticizing the way SmalltalkAgents treats Booleans. David Simmons (quasar@qks.com) has also written eloquently in defence of the QKS implementation, and some of that is reproduced below, with occasional commentary.

Why have #(isNil ifNil: notNil ifNotNil:) be true/false for both zero and nil?

1. We factored the class library so that nil has a numeric “value” of zero. Thus, nil can be used in any mathematical operation. `foo := nil. x := 10 + foo.` is OK. So `isNil` is consistent with the numeric behavior. It would be a mistake to have defined `isNil` the way we did if it was the only way to test for identity of `UndefinedObject` instances. However, it is not, and in fact it is not the uniform Smalltalk identity `#==` test.
2. Nil is the default value for all variables until initialized. The STA definition of `isNil/notNil` can therefore speed up code significantly. We use it in tests for bit flag operations as well, such as:

```
userFlags ifNotNil: "or notNil ifTrue: [
 (7 asBitValue & userFlags) ifTrue: [eventList add: #Closed].
 ...].
```

3. In many other languages, the nil concept is expressed as zero and is used interchangeably. This enables the language to put the equivalence to good effect in expressing algorithms concisely, although it is sometimes a problem for those languages in that they cannot differentiate nil from zero. In Smalltalk nil is a first class object and thus it can be differentiated explicitly via the `#==` operation. So in Smalltalk having the (STA) equivalence gives us the ability to be concise, but not suffer from the inability to differentiate nil and zero.

For example, in STA it is more efficient to scan collections (especially sets and their subclasses) using the `isNil` test.

We define hash tables with two forms of entry:

- Nil. Free slot that has never been used before
- Zero. Free slot that has been used before.

This allows us to delete entries from the hash table without restructuring until the empty % tag falls to some threshold.... Without this technique, a hash table must be restructured whenever an entry is removed (or some other form of used-free slot tagging must be employed). We scan for empty slots via `#isNil`. We scan for valid entries via `== nil`.

Judge for yourself, but to me the only one of these that works at all is the efficiency argument. I see no great advantage in being able to use nil in arithmetic. In fact that opens up a whole range of questions. What is the result of `nil isZero`? How about `nil respondsToArithmetic`? Is `0.0 isNil` true?

Being able to write concise, expressive code is nice, but it’s an argument for adding useful messages, not changing the clearly defined semantics of existing, widely used operations. Even for the sake of efficient code, new structures ought to be sufficient.

I’m not sure what the “many other languages” that work this way are. C is the only language I know of that works that way. It wouldn’t shock me if FORTH did. I doubt there are many languages at as high a level as Smalltalk that do.

Efficiency is the only argument I find reasonable, and there are limits to what I’ll do for efficiency.

Why have Boolean equivalence for `ifTrue:...`?

1. We believe that all objects should have a Boolean equivalent.
2. (<expression>) `asBoolean ifTrue:...` in loops and other tests should be as fast as possible. That’s why almost all implementations of Smalltalk inline them in the first place. It is very common to need the `asBoolean` test. Without it, code must be “structured” with extra statements to convert, flag, and track true/false object cases for all forms of flow control statements. The most common cases looking like `foo isNil if...`, `foo == false or: [foo == nil]` if..., etc.

This can lead to unnecessary and sometimes confusing code....

3. Having the equivalence enables efficient constructs where a method returns nil, false, zero as a simple three-way subswitch. The compiler uses this to very good effect since it allows us to keep the code simple and make it faster, but carry additional information in the return value, too.

```
(result := foo <operation>) iffFalse: [^result]. "Sender at some level
above will look at the result and have extra information about the
operation."
```

vs.

```
(#(case1 case2 case3) includes: (result := foo <operation>))
iffTrue:
 [^result].
```

Overall, we find that having this freedom has led most of our users (and ourselves) to be able to write algorithms that are clear and expressive of their real intent. It is also the case that not having to coerce objects to a Boolean prior to their use in flow control methods improves execution speed. The potential drawback is that a "sloppy" design would result in erroneous flow control logic not being caught via a mustBeBoolean error.

This reflects a poor design and the ability to generate an exception for this single example of coding (logic) mistake is not significant when compared against the myriad other coding construct errors that are possible in Smalltalk.

Well, I'm happy that the QKS staff believe objects should have a Boolean equivalent, but I don't share their faith. I also don't see that passing nil/false/zero as a specially encoded three-way switch, as described in point 3, is a particularly good thing.

That leaves us with efficiency again and the argument that the loss in error checking isn't very significant. I might believe the loss of error checking isn't a big deal, but the efficiency improvements had better be very impressive.

**A COMPROMISE**

Finally, here is something that makes me happy and perhaps provides evidence that USENET can have influence on the real world. David Simmons writes:

As of release 1.1 of SmalltalkAgents, ... we provide a compiler directive called ANSI Compliant: that enforces Boolean testing for: #(iffTrue: iffFalse: iffTrue:iffFalse: whileTrue whileFalse whileTrue: whileFalse: and: or: xor:). We may enhance it in the future to also ensure strict nil testing for #(isNil notNil iNil: ifNotNil: ifNil:ifNotNil:).

We always use strict nil testing for #?.

This is particularly nice, because this flag can be set at the level of a method, a class, or a library (a module construct added in SmalltalkAgents). This means it is possible to have system code using the non-ANSI semantics (without which it would probably break) while writing all user code "properly."

**ERRATA**

As usual, I have something to apologize for. This time it's my passing reference to First Class Software's Object Explorer, a Smalltalk add-on for visually working with object structures. I assumed this was for Smalltalk/V, but it is in fact for VisualWorks version 1.0. First Class Software can be reached at 408.338.4649 (voice) or 408.338.1115 (fax). I haven't even seen a demo of Object Explorer, so please don't take this as a recommendation. ■

Alan Knight works for The Object People. He can be reached at 613.225.8812, or by email as knight@acm.org.

## The Smalltalk Report

provides objective & authoritative coverage on language advances, usage tips, project management advice, A&D techniques, and insightful applications.

Don't miss out!

Yes, I would like to subscribe to THE SMALLTALK REPORT.

1 year (9 issues):

|          |                                          |                                              |
|----------|------------------------------------------|----------------------------------------------|
| Domestic | <input type="checkbox"/> Individual \$79 | <input type="checkbox"/> Institutional \$119 |
| Overseas | <input type="checkbox"/> Individual \$94 | <input type="checkbox"/> Institutional \$134 |

2 years (18 issues):

|          |                                           |                                              |
|----------|-------------------------------------------|----------------------------------------------|
| Domestic | <input type="checkbox"/> Individual \$148 | <input type="checkbox"/> Institutional \$228 |
| Overseas | <input type="checkbox"/> Individual \$178 | <input type="checkbox"/> Institutional \$258 |

**Method of Payment**

Check enclosed (payable to THE SMALLTALK REPORT)  
(Checks for US dollars must be drawn on US bank)

Bill me

Charge my:  Visa  MasterCard  American Express

Card No. \_\_\_\_\_

Exp. Date \_\_\_\_\_

Signature \_\_\_\_\_

Name \_\_\_\_\_

Address \_\_\_\_\_

Title \_\_\_\_\_ Company \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_

Country \_\_\_\_\_ Zip \_\_\_\_\_

Phone \_\_\_\_\_

*To order, return this form with payment to*

The Smalltalk Report

P.O. Box 2027, Langhorne, PA 19047

For faster service, fax: 215.785.6073

# Tensegrity Release 1.0 for Windows and OS/2

**P**OLYMORPHIC SOFTWARE'S TENSEGRITY is an object-oriented database system for use with Digitalk's Smalltalk/V for Windows 2.0 (16-bit version) and Smalltalk/V for OS/2 2.0. Any object can be made persistent and stored in a Tensegrity database. Persistent objects can reference any other object in the image or in the database. Persistent objects are defined just like nonpersistent ones and respond to messages in the usual way.

The package comes with a comprehensive tutorial that includes examples of converting an existing application for use with Tensegrity.

## VERSIONS

Single- and multiple-user versions of Tensegrity are available. The single-user version is for developing and testing applications that will be used by a single user on one machine. The multiple-user version is for developing and testing applications that will be executed on more than one machine on a network. Tensegrity works with any network protocol supported by Windows or OS/2. There are no application coding differences between the single- and multiple-user versions of Tensegrity. So a multi-user application could be developed, but not tested, with the single-user version.

## FEATURES

### Persistent object storage

Tensegrity provides a way to store and retrieve an object's data and store a copy of its class definition on some medium outside the image. That medium could be any device that can be addressed using the file system—a hard drive on the user's machine, a hard drive in another machine on the network, or a CD-ROM, among other things. So, you can store objects using Tensegrity, exit the Smalltalk application without saving the image, restart the application, connect to the Tensegrity database, and find your objects just as you left them.

Any object can be made persistent by sending it the #persistent message.

Tensegrity does not store the compiled methods of an object's class, so if you want to distribute a predefined database, the classes of the objects in that database must be defined already in the image that will connect to the database.

When you change the definition of any class that has some instances in the Tensegrity database, you will have to send a new version of the image to the end user as well.

## NEW CONCEPTS

You'll need to understand a few new concepts before using Tensegrity. For example, all database access must be done within a transaction. Atomic blocks provide an easy way to access the database without dealing with the details of transactions. You also need to understand how persistent objects are structured and stored. These new concepts are simple and easily learned.

### Transactions

All database access must occur within a transaction. Transactions provide the locking mechanism necessary to ensure database integrity. Accessing a persistent object from within a transaction is a simple matter. Here's an example:

```
aTransaction := Transaction newRW.
aTransaction makeCurrent.
persistentCustomerSet add: aCustomer.
aTransaction commit; release.
```

This example creates a read/write transaction and adds a customer to a persistent collection. The transaction is then committed and released.

### Atomic blocks

Atomic blocks are macros that make accessing persistent objects even easier than using transactions explicitly. To add an object to a persistent collection just do this:

```
[persistentCustomerSet add: aCustomer] atomic.
```

This has the same effect as creating a transaction, adding the object, committing, and releasing the transaction. Atomic blocks can be a great time saver and will be sufficient for most database access. Most single-user applications will probably never need to use anything but atomic blocks.

Different flavors of atomic blocks provide for retrying a transaction that failed a number of times or performing another block in case of an error.

### Object complexes

Object complexes are used to define the locking boundaries around objects. An object complex can contain one or more objects. When a persistent object is accessed within a transaction, the complex in which that object resides is locked. So you need to be careful which objects you put together in the same complex.

An object complex is created whenever you make an object

persistent. So if you create a collection of all the accounts a bank owns, it would be best to create the collection and make it persistent, then create an account, make it persistent, and add it to the collection. That way, each account will be in its own complex. When one account is accessed within a transaction, the others will not be locked.

An object complex is also the unit of exchange between the persistent store and memory. When an object is accessed within a transaction, its entire complex is brought into memory. It is wise to limit the size of an object complex. It's also a rather good idea to have objects that are accessed at nearly the same time in the same complex.

### Object containers

Tensegrity stores objects complexes in object containers. Each object container consists of two files. Most single-user applications will probably store all objects in a single container. Multiple-user applications may store some objects on a drive that is accessible to other machines on the network and other objects on a local drive. It is up to the developer to decide where the containers reside. Containers can have a user-defined name or a Tensegrity-generated name.

### Object identity

The '=' message should be avoided in applications that use Tensegrity. The results will be unpredictable because of the mechanism Tensegrity uses to reference persistent objects. Tensegrity provides the #areYou: message, which serves the same purpose. It will work regardless of whether the two objects being compared are persistent.

### Garbage collection

We've come to take garbage collection for granted with Smalltalk. It just happens. Not so with Tensegrity. The application designer must be sure to give end users access to Tensegrity's garbage collection facility or provide some means of automatic garbage collection, perhaps based on elapsed time or growth in size.

### Class mutation

When Tensegrity tries to access an object in the database that has a class definition different from the one in the image, it signals the UnknownClass exception. Your application is then responsible for asking Tensegrity to update the class definition in the database and initializing any added variables. The update is done with the #updateSchema message. It can take a bit of time as it sweeps the database of objects that aren't referenced.

When an instance variable is added to a class definition, it will be added to any existing instances of that class and given the value nil. This can be a bit of a problem if that instance variable needs to be initialized. One way around this is to create variable access messages for each instance variable, which might look something like this:

```
accounts
 ^{accounts isNil
```

```
ifTrue: [accounts := BTreeSet new]
ifFalse: [accounts]
```

Then, instead of using an instance variable like this:

```
accounts add: aNewAccount.
```

do it like this:

```
self accounts add: aNewAccount.
```

### Exception handling system

The Exception Handling System (EHS) is a gem of Smalltalk genius that comes along with Tensegrity. It's based on the public domain EHS written by Hal Hildebrand. The EHS provides a way of protecting blocks of code by trapping errors and providing an opportunity for recovery. Proper use of the EHS can prevent the end user from ever seeing a walkback or runtime error window. It can also help the developer produce code that is more efficient and simpler to read.

### Types of exceptions

There are two basic types of exceptions: FatalEvent and ProceedableEvent. A fatal event is an exceptional event that cannot be resumed. Perhaps the error can be corrected and the process restarted. A proceedable event is exceptional event that can be resumed.

### Types of exception handlers

**Context handler** A context handler is used to protect a Context. A context is a block of code enclosed with square brackets. Here is an example of a context handler that allocates more memory when the LowOnMemory exception is raised:

```
[self addNewAccounts]
 when: LowOnMemory
 do: [:event | self allocateMoreMemory. event restart].
```

There is a special type of context handler called the Finally block. It is used when you always want a block of code to execute, even if the code within the executed block fails. Here is an example of unprotected code:

```
line := stream nextLine.
stream close.
```

In the above example, if the #nextLine message fails, it will cause a walkback and the #close message will not be executed.

```
[line := stream nextLine] finally: [stream close].
```

In the example above the stream will be closed even if the #nextLine message fails.

**Class handler** A class handler is like a context handler except it applies to all messages in the class and all messages sent by them.

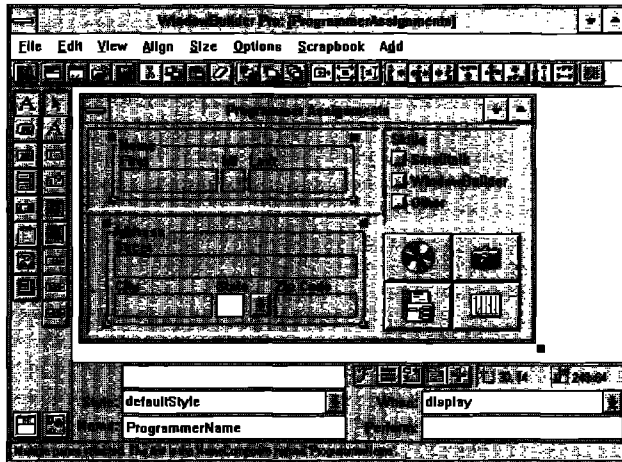
Tensegrity includes a new class hierarchy browser that supports class handlers. There is a radio button added in the top center pane below the instance and class radio buttons. When the

# WINDOWBUILDER PRO!

*The New Power in Smalltalk/V Interface Development*

Smalltalk/V developers have come to rely on WindowBuilder as an essential tool for developing sophisticated user interfaces. Tedious hand coding of interfaces is replaced by interactive visual composition. Since its initial release, WindowBuilder has become the industry standard GUI development tool for the Smalltalk/V environment. Now Objectshare brings you a whole new level of capability with WindowBuilder Pro! New functionality and power abound in this next generation of WindowBuilder.

WindowBuilder Pro/V is available on Windows for \$295 and OS/2 for \$495. Our standard WindowBuilder/V is still available on Windows for \$149.95 and OS/2 for \$295. We offer full value trade-in for our WindowBuilder customers wanting to move up to Pro. These products are also available in ENVY+/Developer and Team/V™ compatible formats. As with all of our products, WindowBuilder Pro comes with a 30 day money back guarantee, full source code and no Run-Time fees.



## Some of the exciting new features...

- CompositePanes: Create custom controls as composites of other controls, treated as a single object, allowing the developer higher leverage of reusable widgets. CompositePanes can be used repeatedly and

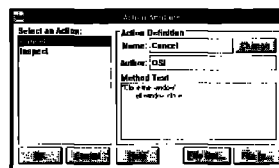
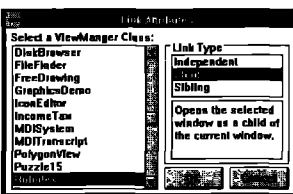
because they are Class based, they can be easily subclassed; changes in a CompositePane are reflected anywhere they are used.

- Morphing: Allows the developer to quickly change from one type of control to another, allowing for powerful "what-if" style visual development. The flexibility allowed by morphing will greatly enhance productivity.

- ScrapBook: Another new feature to leverage visual component reuse, ScrapBooks provide a mechanism for developers to quickly store and retrieve pre-defined sets of components. The ScrapBook is a catalog of one's favorite interface components, organized

into chapters and pages.

- Rapid Prototyping capabilities: With the new linking capabilities, a developer can rapidly prototype a functional interface without writing a single line of code. LinkButtons and LinkMenus provide a pow-



erful mechanism for linking windows together and specifying flow of control. ActionButtons and ActionMenus provide a mechanism for developers to attach, create, and reuse actions without having to write code. These features greatly enhance productivity during prototyping.

- ToolBar: Developers can Create sophisticated toolbars



just like the ones in the WindowBuilder Pro tool itself.

- Other new features include: enhanced duplication and cut/paste functions, size and position indicators, enhanced framing specification, Parent-Child window relationship specification, enhanced EntryField with character and field level validation, and much more...
- Add-in Manager: Allows developers to easily integrate extensions into WindowBuilder Pro's open architecture.

Catch the excitement, go Pro!  
Call Objectshare for more information.

**(408) 727-3742**

Objectshare Systems, Inc  
Fax: (408) 727-6324  
CompuServe 76436,1063

5 Town & Country Village  
Suite 735  
San Jose, CA 95128-2026

handlers radio button is clicked, all the handlers for the selected class appear in the list pane where messages are usually displayed.

Class handlers are added much like methods. When you create a new class handler, the class hierarchy browser fills the method pane with the pattern used for creating a class handler. When you save the handler, the class hierarchy browser asks you to which exception the handler is tied.

When an exception occurs, any context handler that exists for that exception is executed. If there isn't one for that exception, the class handler for that exception is executed if it exists. If no exception handlers exist for that exception, a walkback appears.

#### Collection classes

Tensegrity adds some new B-Tree-based collection classes. They are optimized for use as large persistent objects. These new collections can also be nonpersistent objects.

#### RUNTIME

A runtime license must be purchased before an application that uses Tensegrity can be distributed. A special runtime DLL comes with this license. The DLL that comes with the development version will not allow a runtime application to be tested. All testing must be done within the development environment until the runtime license is purchased.

After a single-user runtime license is purchased, an unlimited number of applications that use Tensegrity can be distributed.

#### SUITABILITY

The word *tensegrity* was coined by R. Buckminster Fuller, famous for the geodesic dome among other things. Tensegrity is a combination of tension and integrity. Fuller believed a structure that had tensegrity could be scaled up or down in size while remaining sound and efficient. Although I haven't used the Tensegrity Database System for any *huge* projects yet, it does seem to possess this quality.

The most important application I developed with Tensegrity has object containers that range in size from 20K to 300M. The contained objects range in variety from strings of text to a trained neural network. It runs in a single-user environment.

Debugging an application that uses Tensegrity can be a bit trying at times. Read and write conflicts can be difficult to find. Polymorphic Software is working on a set of user interfaces that will help with debugging and managing Tensegrity databases. This will be a welcome addition to a very strong package.

Employees of Polymorphic Software maintain a presence in the Digitalk Forum on CompuServe (GO DIGITALK). There is plenty of discussion of Tensegrity as well as other object-oriented databases that work with Digitalk's versions of Smalltalk. ■

*David Bush founded Object Evolution after five glorious years as a systems engineer at EDS. Object Evolution, located in Kailua, Hawaii, sells asynchronous communications classes for Smalltalk/V for Windows and OS/2. He can be reached at dbush@uhunix.uhcc.Hawaii.Edu. Comments and criticism are welcome.*

# RECRUITMENT

To place a recruitment ad, contact Wendy Plumb at 212.274.0640 (v) or 212.274.0646 (f)

## SMALLTALK ARCHITECTS/DEVELOPERS

Minneapolis • Atlanta • Raleigh  
NY • Boulder

As a leader in the delivery of Object-Oriented System Integration Services, SHL Systemhouse invites you to explore challenging and unique opportunities within our organization. **SMALLTALK** opportunities exist in Minneapolis, Atlanta, Boulder, Raleigh and NY, for **Technical Architects, Project Managers, Senior Software Developers and Software Engineers.**

We seek client/server, object-oriented professionals with impressive industry credentials who share our worldwide commitment to excellence. These results-oriented information professionals must thrive on challenges and possess exceptional technical skills as well as business advisory experience.

For Consideration send your resume in confidence to:

Michelle Hayden Dept. SMR294  
SHL Systemhouse  
950 South Winter Park Drive, Suite 200  
Casselberry, Florida 32707  
1.800.769.8704 or Fax 407.767.5309  
(Extra Fine Mode)



## "Smalltalk Architects and Programmers"

Capitalize on your Smalltalk expertise! Haestad Methods is a dynamic software company in Waterbury, CT. We currently develop and market high-end Windows-based applications. We use the most powerful tools and techniques in the industry, such as: ENVY/Developer, Digitalk Smalltalk/V, and WindowBuilder Pro. We are growing and looking for intelligent, honest, hard-working, team-oriented people. Call us today!

**Software Architect** - Seasoned Smalltalk designer with at least two completed software products. Assume responsibility for the overall integrity and design of one of our projects. Analyze user requirements; design key subsystems and contracts.

**Software Engineer** - Experienced programmer with at least one year of Smalltalk and OOP. Be the lead on a major software subsystem. Design the classes to implement key system contracts.

**Haestad Methods, Inc.**  
37 Brookside Road  
Waterbury, CT 06708  
Attn.: John Haestad

**Voice: (800) 727-6555 FAX: (203) 597-1488**

All inquiries will be kept strictly confidential.

## SMALLTALK DESIGNERS AND DEVELOPERS

We Currently Have Numerous Contract and Permanent Opportunities Available for Smalltalk Professionals in Various Regions of the Country.



Salient Corporation...  
Smalltalk Professionals Specializing in the  
Placement of Smalltalk Professionals

For more information, please send or FAX your resumes to:

Salient Corporation  
316 S. Omar Ave., Suite B.  
Los Angeles, California 90013.

Voice: (213) 680-4001 FAX: (213) 680-4030

## WORK IN FLORIDA SMALLTALK DEVELOPERS



Tech Aid has many *long-term* assignments in South Florida for Smalltalk developers in all dialects. DOS, Windows, SQL, GUI, ODBMS, and RDBMS skills a plus.

Call or send resume:

Tech Aid  
P.O. Box 915134  
Longwood, FL 32791  
Fax: (407) 788-1279



Who  
builds  
the tools for  
the toolmaker?

You could, if you join Sybase. Our Enterprise Momentum™ gives developers a new model-based approach to enterprise-wide applications. It allows developers to meet the diverse needs of their users — regardless of hardware, operating system, GUI, data sources, or applications.

Your insight and skill may help us develop more tools for the SYBASE Momentum™ family. Find out by exploring one of these opportunities with Sybase now:

### **Project Manager, Enterprise Program**

Requires extensive project management experience, preferably on large (5000+ task) OO, CASE or client/server software projects.

### **Development Manager**

Your team will deliver forms management services in a GUI database application, Smalltalk environment, requiring extensive expertise with these technologies.

### **Integrity Engineer**

Requires extensive experience in developing tests for a large OO product, including exposure to relational databases.

We offer an excellent compensation package and competitive benefits. Please send, fax or e-mail your resume, indicating position of interest, to: Sybase, Inc., 6425 Christie Avenue, 5th Floor, Corporate Staffing Department, ATTN: AD Code: ED, Emeryville, CA 94608. FAX: (510) 922-5310. E-mail: ellend@sybase.com. EOE/AA employer. Principals only, please.



The Enterprise Client/Server Company™

# FINALLY, CLIENT/SERVER INTEGRATION.

Not long ago, client/server development required massive amounts of time, money and expertise to combine different and complex technologies.

**PARTS**



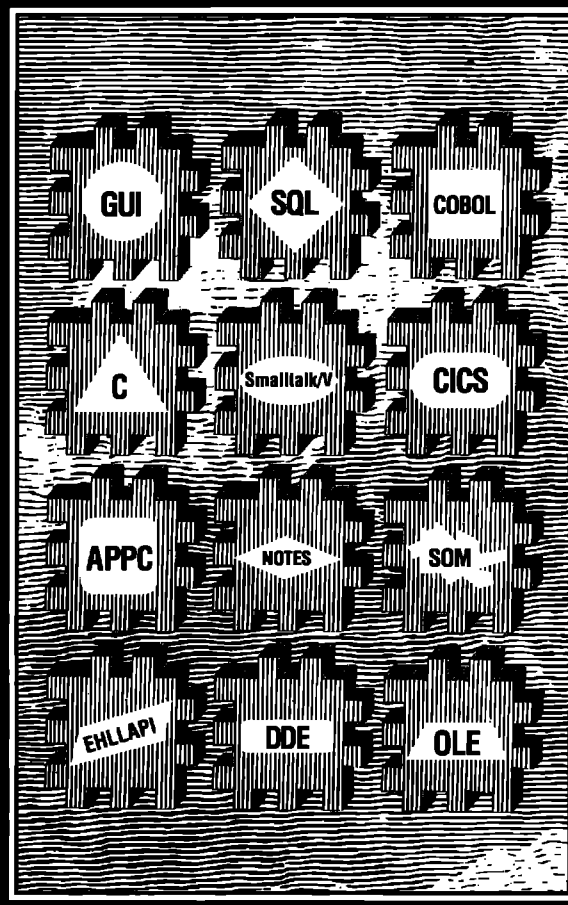
**DIGITAL**

Now **Digitalk PARTS**, a rapid application development tool set, lets you easily integrate your software assets into client/server applications.

**PARTS** is the only object-oriented technology that lets you leverage your legacy code and the knowledge of your current staff.

Only **PARTS** products let you take existing code—written in **Smalltalk/V**, **COBOL**, **C**, **SQL** and other languages—and wrap it into components or “parts.” Which can then be virtually snapped together visually. The result is smooth-running client/server applications in a fraction of the usual time. For a fraction of the usual cost.

**PARTS** supports all popular **SQL** databases like **Sybase**, **Oracle** and **DB2**. Plus legacy or late model



systems like **CICS**, **COBOL**, **APPC** and **SOM**. And **PARTS** lets you develop on both **OS/2** and **Windows**.

**RATED #1—TWICE.**

Only months ago, **PC WEEK** awarded **PARTS Workbench** the highest rating ever in the **OS/2**

category, calling it “the definitive visual development tool!”

And **InfoWorld** ranked **PARTS** the #1 component-based tool for visual development. **InfoWorld's** Stewart Alsop adds: “There’s nothing like it on the PC.”

To make large teams productive, **PARTS** also supports group development and version control. Plus **PARTS** has a host of graphical power tools to give you all the power of objects—without the learning curve.

**10 YEARS EXPERIENCE.**

And **PARTS** is from **Digitalk**. The company that’s been providing object-oriented tools to the **Fortune 500** longer than anyone else in the world—with over 125,000 users.

Call 800-531-2344 X 610 and ask about our

**PARTS Workbench Evaluation Kit.**

With minimum effort, you’ll learn why **PARTS** is the maximum solution for client/server integration.



**PARTS. THE CLIENT/SERVER INTEGRATION TOOL.**

**DIGITAL**