SG24-4828-00

VisualAge for Smalltalk Handbook
Volume 1: Fundamentals

September 1997

International Technical Support Organization
San Jose Center

SG24-4828-00

International Technical Support Organization

VisualAge for Smalltalk Handbook
Volume 1: Fundamentals

September 1997

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix A, “Special Notices.”

First Edition (September 1997)

This edition applies to VisualAge for Smalltalk, Versions 2, 3, and 4, for use with OS/2, AIX, and Microsoft Windows 95/NT.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2
650 Harry Road
San Jose, California 95120-6099

Contents

Preface

This book addresses many common questions in the VisualAge for Smalltalk development arena. It covers various aspects of VisualAge and IBM Smalltalk through answers to frequently asked questions, hints and tips from users and developers, and online bulletin boards inside and outside of IBM.

This redbook will help VisualAge for Smalltalk developers find answers to their everyday questions. The book provides usage guidelines for areas such as change management, performance, database access, and transaction processing, to help developers avoid common programming pitfalls.

The *VisualAge for Smalltalk Handbook* has two volumes:

- Volume 1: Fundamentals

  Volume 1 covers general programming questions on such topics as image maintenance, graphical user interfaces, naming conventions, and the IBM Smalltalk language.

- Volume 2: Features

  Volume 2 focuses on VisualAge for Smalltalk features, such as AS/400 Connection, Communications and Transactions, Distributed, Reports, and Web Connection.

This book is written for VisualAge for Smalltalk programmers, project leaders, and developers. Understanding many of the questions and answers in this book requires substantial knowledge of the VisualAge for Smalltalk product and the Smalltalk language.

This book does not discuss questions related to methodologies, analysis and design. For a discussion of object-oriented analysis and design with VisualAge for Smalltalk, refer to *Visual Modeling Technique*.

How This Redbook Is Organized

This redbook contains 362 pages. It is organized as follows:

- Volume 1: Fundamentals

  This book sets the stage for programming with VisualAge for Smalltalk. It contains questions and answers dealing with the base product and everyday programming pitfalls.

  – Chapter 1, “What Is VisualAge for Smalltalk?”

    This chapter provides some non-technical discussion of programming with VisualAge for Smalltalk.

  – Chapter 2, “General Information”

    This chapter provides general programming tips, from image maintenance and packaging to troubleshooting.

  – Chapter 3, “Graphical User Interface”

    This chapter provides a discussion of graphical user interface parts, such as containers, notebooks, buttons, or tables.

  – Chapter 4, “IBM Smalltalk Programming Language”

    This chapter provides questions and answers regarding the IBM Smalltalk programming language.

  – Chapter 5, “ENVY”

    This chapter provides a discussion of the ENVY team programming environment.

  – Chapter 6, “Microsoft Windows”

    This chapter provides a discussion of programming issues related to the Microsoft Windows platform.

  – Chapter 7, “National Language Support”

    This chapter discusses national language support with VisualAge for Smalltalk.

- Volume 2: Features

  This book details on the various VisualAge for Smalltalk features, some of which are packaged with the base product and some others are separately orderable.

  – Chapter 1, “AS/400 Connection”

    This chapter provides a discussion of the AS/400 Connection feature and related items such as ClientAccess/400 and DB2/400.

- Chapter 2, “Communications/Transactions”

  This chapter provides a discussion of the Communications/ Transactions feature and its protocols, such as TCP/IP, APPC, CICS, or NetBIOS.

- Chapter 3, “Interface to External Routines”

  This chapter discusses the use of external routines, such as C or COBOL programs.

- Chapter 4, “CICS and IMS Connection”

  This chapter provides a discussion of the CICS/IMS Connection feature.

- Chapter 5, “Database”

  This chapter covers questions and answers related to database access from VisualAge for Smalltalk, such as DB2 or Oracle.

- Chapter 6, “Distributed”

  This chapter provides insight into the VisualAge for Smalltalk Distributed feature, for example, remote objects or object spaces.

- Chapter 7, “Reports”

  This chapter covers the VisualAge for Smalltalk Reports feature.

- Chapter 8, “SOM/DSOM”

  This chapter provides a discussion of the System Object Model and the Distributed System Object Model.

- Chapter 9, “Web Connection”

  This chapter discusses the Web Connection feature of VisualAge for Smalltalk.

ITSO on the Internet

Internet users may find additional material about new redbooks on the ITSO World Wide Web home page. Point your Web browser to the following URL:

http://www.redbooks.ibm.com/redbooks

IBM internal users may also download redbooks or scan through redbook abstracts. Point your Web browser to the internal IBM Redbooks home page:

http://w3.itso.ibm.com/redbooks

If you do not have World Wide Web access, you can obtain the list of all current redbooks through the Internet by anonymous FTP to:

```
ftp.almaden.ibm.com
cd /redbooks
get itsopub.txt
```

The FTP server, *ftp.almaden.ibm.com*, also stores the sample software from the diskettes accompanying some of the redbooks. To retrieve the sample files, issue the following commands from the */redbooks* directory:

```
cd SG24xxxx          ← Redbook number
binary
get SampleFile.EXE
ascii
get SampleFile.TXT
```

All users of ITSO publications are encouraged to provide feedback to improve quality over time. Send questions about and feedback on redbooks to:

redbook@vnet.ibm.com *or*
REDBOOK at WTSCPOK *or*
USIB5FWN at IBMMAIL

If you find an error in the software that is supplied with a redbook, please send a bug report with a description of the problem to

bugs@vnet.ibm.com

To receive regular updates on new redbooks and general IBM announcements, you can subscribe to the IBM Announcement Listserver. It automatically supplies an Internet e-mail user with timely new announcement information (titles and optionally the letter or abstract) from selected categories. To get started, send an e-mail to

announce@webster.ibmlink.ibm.com

The keyword SUBSCRIBE must be the only word in the body of the e-mail; leave the subject line blank. You will receive a category form and listserver details. To immediately start your subscription to, for example, AS/400 announcements, put the words SELECT HW120 in the body of the note. On the afternoon of an announcement day, you will receive e-mail with the announcement, along with a list of newly available redbook titles. To obtain a full abstract of a particular redbook, use GET SG242535 in the note.

VisualAge Support on CompuServe

VisualAge users are encouraged to use CompuServe® to ask questions about VisualAge and its features. When logged on to your CompuServe account, type GO VISUALAGE to get into the *IBM Workstation Rapid Application Development* (WRAD) conference. You will find sections to discuss the following and other VisualAge topics:

- Installation/Begin
- Communication/Languages
- Database
- AS/400 Connection
- Web Connection

About the Authors

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose, CA.

Andi Bitterer works as a consultant for the International Technical Support Organization at the Almaden Research Center in San Jose, California. He graduated with a degree in computer science from the Technical University in Darmstadt, Germany. Andi joined IBM in 1987 and worked as an application development specialist in large customer projects for IBM Integration Systems Services and at the German AS/400 Field Support Center. Since joining the ITSO in 1994, he has taught workshops worldwide on object-oriented application development and the Internet. Andi is the author of three other VisualAge for Smalltalk books published by Prentice Hall. You can reach him by e-mail at bit@acm.org.

Vincent Dijkstra works as an Education Consultant at the IBM International Education Center, La Hulpe, Belgium. He holds Bachelor of Arts and Master of Arts degrees in Technology and Policy from the Eindhoven Technical University. Vincent started working for IBM in 1992 on a learning resource to assist managers and teams in transforming their organizations. In 1994 he joined the IBM Object Technology University, where he teaches Smalltalk, VisualAge, and more recently Java and analysis and design techniques. You can reach Vincent by e-mail at vdijkstra@vnet.ibm.com

Boris Shingarov received his Master of Science degree in Quantum Fields at Moscow State University, M.V.Lomonosov, in Russia. He is the head of the Center for VisualAge and IBM Smalltalk, Dialogue/MSU, providing VisualAge consulting and services all over Russia. Currently, Boris teaches VisualAge for Smalltalk courses at MSU. He can be reached by email at boris@visualage.dialogue.msu.su.

Acknowledgments

This book has been a team effort par excellence, even though most contributors were not even aware of being a team. However, we want to acknowledge everyone, inside and outside of IBM, who, knowingly or unknowingly, added information to the collection of invaluable hints and tips in this book.

Thanks go to Dave Allison, Rainer Angstmann, Jean-Pierre Augias, Wayne Beaton, Jan Belik, Paul Berglund, Jonathan Bezuidenho, Debu Bhattacharya, Franco Biaggi, Peter Boonen, Chris Bosman-Clark, Toufic Boubez, David Bourke, Dave Bowman, Mamie Branch, Michel Brassard, Chuck Bridgham, Robert Brown, Jordi Buj, Jerry Callistein, Jon Capezzuto, Tony Carlier, Janelle Carroll, Ron Cherveny, George Chiu, Chris Clark, Eric Clayberg, Stephen Cooper, Ellis Covington, Tim Cowan, Mark Cresswell, Jim Crossgrove, Greg Curfman, James Curtis, Chris Davia, Kirk Davis, Gordon Davis, John DeBinder, Ricardo Devis, Bill Dickenson, Stef van Dijk, Sue Dubbeling, John Earle, Ralf Eberhardt, Kevin Egolf, Linda Fargo, Ahmed Fattah, Gerald Fischer, Hans Forsberg, Mike French, Annick Fron, Sabine Gaissert, Ken Gelsinger, Chris Gerken, Michel Giroux, Erick Godoy, Tom Gordon, Chris Grindstaff, Juergen Guenauer, Sven Guyet, Tim Hanis, Brad Hanks, John Hansen, Nicla Havrup, Chris Hayes, Frank Haynes, Pam Helyar, Andy Heys, Hal Hildebrand, Tim Hilgenberg, Andrew Hobbs, Steve Hobson, Bryan Hogan, Andreas Huber, Greg Hutchinson, Mike Jacobs, Charif Jaouhar, Chuck Jaynes, Christopher Joak, Greg Johnson, Tom Johnson, Jane Jones, Gary Karasiuk, Gijsbert Karens, Dan Kehn, John Kellington, Dave Kennedy, Brian Kent, Kelly Keplinger, Alan Knight, George Kober, Martin Kunz, Bruce Lambert, Micky Lim, Michael Linderman, Daniel Lipp, Grace Liu, Bill Lockard, Tom Logan, Mark Lorenz, Pablo Lorenzo, Sabina Luzzatti, Stewart MacLean, Bill Mathews, Cynthia McCrickard, David McGee, John McIntosh, Ted McKnight, Gerald Meazell, Eric Meredith, Jim Mickelson, Gonzalo Mourino, Pat Mueller, Gen Nagatsuka, Martin Nally, Binh Nguyen, Jimmy Nguyen, Achim Nogli, Ami Noyman, Patrick O′Donnell, John O′Keefe, Dan Ohlhaut, Omar Padilla, Jim Pendergast, Rosa Peral, Ralf Pfiszter, Enrico Piccinin, Oliver Picot, Guenter Pindhofer, Rene Plourde, Greg Plummer, Bob Poulton, Steve Reeves, Brian Remedios, James Rendell, Scott Rich, Cameron Roy, Jouko Ruuskanen, Jari Saari, Craig Setera, Dan Shaver, Ed Shirk, Allen Smith, Christine Smith, Craig Smith, David Smith, Larry Smith, Daniel Stainhauser, Jim Stewart, Jeff Stratford, Jean Talbott, Steve Tang, John Tobin, Mark Tompkin, Mike Toohey, Rick Trotter, David Twyerould, Timo Ullrich, Harold Wadler, Jim Wason, Christopher Webster, Tony Weddle, Ronny Weisz, David Whiteman, Mark Wilkes, Joe Winchester, Rory Woodward, Cindy Wotus, Glorious Wright, Alfred Wu, Barry Young, Ric Zapanta, Sherwood Zern, and Slavik Zorin, for their numerous contributions, questions or answers, hints and tips.

Many thanks go to Alexis Scott, Nancy Lewis, Paul Braun, and Shawn Walsh at the ITSO Raleigh Center, for their great administrative support, and to Pat Donleycott, ITSO Raleigh Center Manager, for hosting the residency.

Thanks also to Shirley Hentzell and Maggie Cutler for their outstanding editing of this book, to Liz Rice for her editorial assistance, and to Geoff Nicholls, without whose advice this book would not have a decent index.

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in "ITSO Redbook Evaluation" to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Home Pages at the following URLs:

For Internet users	`http://www.redbooks.ibm.com`
For IBM Intranet users	`http://w3.itso.ibm.com`

- Send us a note at the following address:

  `redbook@vnet.ibm.com`

Chapter 1. What Is VisualAge for Smalltalk?

Description of VisualAge for Smalltalk

Question What is VisualAge for Smalltalk?

Answer VisualAge for Smalltalk is a client/server rapid application builder for constructing state-of-the-art, flexible, scalable business applications. With extensive libraries of classes, a rich set of preexisting parts from IBM and other vendors, wrappering of existing COBOL, C, IMS transactions or VBX controls, and multimedia authoring support, VisualAge enables rapid assembly of applications with minimal new code. VisualAge for Smalltalk supports development on Windows, OS/2, or AIX, with full portability of applications across these target platforms. It combines the power of a pure object-oriented environment with the ease of visual programming. Users can connect prefabricated, reusable components or extend these components with user-created code. VisualAge for Smalltalk eases the learning and implementation of object-oriented technologies. VisualAge for Smalltalk supports multiple client/server models, GUI construction, popular communications and transaction protocols, access to industry-leading databases, the CORBA standard for object interoperability, and multiple legacy interfaces. VisualAge for Smalltalk Standard is the entry-level product for individual programmers. The Professional product, for individuals or teams, includes version control and configuration management. The Professional Server product includes a LAN repository for team development. Add-on components include support for native ORACLE data access, communications and transactions, reports, AS/400, IMS, and distributed objects.

VisualAge for Smalltalk Technology

Question What type of technology does VisualAge use?

Answer Application developers use a construction-from-components or construction-from-parts paradigm when using VisualAge. VisualAge is based on IBM Smalltalk, which embodies the principles of object-oriented (OO) technology to build applications, providing all of the benefits of reuse and rapid application development. With VisualAge′s GUI-based development platform, proficiency in OO is not a prerequisite, thus easing a developer′s entry into OO.

Description of Smalltalk

Question What is Smalltalk?

Answer Smalltalk is a popular OO language. Often referred to as a 5GL, Smalltalk is a very powerful, high level language that offers exceptional capabilities for rapid prototyping and delivery of applications. It is an emerging language in the programming environment.

Smalltalk has been around since 1971. It was developed in Xerox PARC, and it was the first system in the history of computers featuring a GUI with windows, mice, and other properties of the modern graphical operating systems. IBM Smalltalk is a portable, robust, standards compliant, integrated Smalltalk language and development environment embedded as a component of VisualAge. IBM Smalltalk adheres to industry standards such as OSF Motif and POSIX and is based on the proposed common-base Smalltalk specification submitted to the X3J20 committee for making Smalltalk recommendations to ANSI.

Description of ENVY

Question What is ENVY?

Answer ENVY is an incredibly powerful Smalltalk development environment from OTI, having features that no other Smalltalk development environment has. Originally, ENVY was an environment, to be based on top of other basic Smalltalk implementations. But now, in IBM Smalltalk, you have a full from-scratch integration between the language and the environment.

Stability and Maturity

Question How stable and mature is VisualAge for Smalltalk?

Answer VisualAge for Smalltalk was in development for several years before it became generally available in the first quarter of 1994. According to Eric Ostrom, a consultant for IS International, ″VisualAge [for Smalltalk] is the most solid release 1 development tool I have ever used. Each release is faster and offers more features. Version 3.0 with distributed Smalltalk is outstanding.″ Standards-conforming IBM Smalltalk (and therefore VisualAge for Smalltalk) is based on the proposed common-base Smalltalk specification submitted to the X3J20 committee for making Smalltalk recommendations to ANSI. Both the chairman and secretary of the ANSI Smalltalk committee work for IBM in Smalltalk development. In addition, IBM Smalltalk adheres to industry standards of OSF Motif, X-Windows, POSIX, and IBM′s CORBA implementation (SOM).

Openness

Question In what respect is VisualAge for Smalltalk open?

Answer It is portable among platforms— Windows, OS/2, AIX, Sun, and HP. VisualAge for Smalltalk supports IBM and non-IBM databases and a variety of communication protocols, including TCP/IP, for operation in heterogeneous environments. VisualAge for Smalltalk is built on a nonproprietary standards base (Smalltalk) that can be extended and modified by customers and third party vendors.

Team or Professional Version

Question Does VisualAge have a team or pro edition?

Answer Yes. VisualAge for Smalltalk actually has two base versions—Standard and Professional. VisualAge for Smalltalk Standard is for the individual programmer. VisualAge for Smalltalk Professional is for individuals or teams and can support a LAN-based library of classes and parts, version control, configuration management and other functions that support collaborative team development. Additionally, a Professional Server product is available for VisualAge for Smalltalk and IBM Smalltalk. The Professional Server extends the box product by providing multiple user access and a central LAN-based library of classes and parts that developers can access concurrently.

Advantages of VisualAge

Question What are the key advantages of VisualAge?

Answer VisualAge for Smalltalk is the only tool of its class that offers full integration of visual programming, object-oriented environment, and best-of-breed team support. Programmers never need to leave one tool to go to another to build and test a complete application. Teams of programmers can easily manage complex object-oriented development projects. VisualAge for Smalltalk permits programmers to extend their applications and the

VisualAge for Smalltalk tool itself. Where other tools require programmers to work within the limits of the tool, VisualAge allows programmers to build new extensions to the VisualAge product that become reusable components and part of the complete Smalltalk environment. VisualAge implements ″class-based″ programming, such that new parts become Smalltalk classes, which can be part of the class hierarchy system and can be located with the Smalltalk browsers. VisualAge for Smalltalk includes state-of-the art multimedia authoring support, which allows developers to build exciting kiosk-style applications. VisualAge for Smalltalk also provides a report writer, which provides the ability to lay out reports using the facilities of the VisualAge construction-from-parts paradigm

Advantages of Smalltalk

Question What are the key advantages of IBM Smalltalk?

Answer IBM Smalltalk is a very robust Smalltalk that is based on industry standards such as POSIX for filing systems and Motif for GUI. It is also based on the proposed common Smalltalk standard that was submitted to the X3J20 committee for making Smalltalk recommendations to ANSI in 1993. IBM Smalltalk contains a wide range of functions spanning more than 1100 classes in IBM Smalltalk and more than 2500 classes in VisualAge for Smalltalk. IBM Smalltalk enables robust portability. Unlike other portable implementations that emulate objects on various platforms, IBM Smalltalk applications automatically take on the look and feel of target platforms by using the native platform widgets. Unlike simple check-in and check-out library systems, IBM Smalltalk team support includes an object repository, true version control, and configuration management capabilities. Do you have a vendor program similar to Digitalk′s and Powersoft′s that encourages third party vendors to build parts that can be utilized with VisualAge? We have published a Construction from Parts architecture that teaches vendors and end-users how to create their own parts. IBM′s Object Connection program for VisualAge encourages other companies to build components that enhance the applicability and value of VisualAge. We are currently working with more than 100 corporate members who are marketing more than 50 products.

VisualAge for Smalltalk and VisualAge Generator

Question How would you compar VisualAge for Smalltalk and VisualAge Generator as client-server application development tools?

Answer VisualAge Generator provides a highly productive workstation development environment for traditional host-based COBOL developers that exploit new object-oriented technologies while not requiring an investment in additional skills training. VisualAge Generator uses a 4GL COBOL-like scripting language along with visual programming for developing line-of-business client/server solutions. VisualAge Generator provides facilities for building both client and server components, interactive test at the source level, and generation of both client and server. VisualAge for Smalltalk is a fully object-oriented tool providing a visual programming construction-from-parts paradigm and uses IBM Smalltalk as the scripting language. VisualAge for Smalltalk focuses on building client/server solutions while providing extensive database access, communications support, and support for reusing existing enterprise data and code. The products are complementary. Each has its own strengths. In some customer situations, one of the products may provide the right solution, whereas in others the entire solution may dictate that both products provide the ultimate benefit. For users wanting to explore the world of object-oriented programming while maintaining a 4GL development environment, VisualAge for Smalltalk should be a very attractive addition.

VisualAge for Smalltalk and C++

Question How would you compare VisualAge for Smalltalk and VisualAge for C++?

Answer C++ is the language of choice for the professional C or C++ programmer building high-performance, small-footprint, or real-time applications, for ″close to the metal″ application development. Smalltalk is the language of choice for the professional programmer who wants a rapid application development environment for solving enterprise business problems, for ″business logic″ application development. C++ and Smalltalk are equivalent in capability, for well-trained and skilled developers.

Visualizer

Question How does Visualizer compare with VisualAge for Smalltalk?

Answer Visualizer is a powerful decision support tool that allows end users to visually construct database queries and generate reports. It also features add-on components up to and including a GUI builder. VisualAge is targeted for professional developers rather than end users.

Team Connection

Question Does VisualAge for Smalltalk support Team Connection?

Answer VisualAge provides team support, using an object repository that is tightly integrated in the product and that provides a high-level of support for Smalltalk-based development. Our direction is to expand this excellent object repository with bridges to other technologies such as TeamConnection.

Supported Platforms

Question On which platforms is VisualAge for Smalltalk offered?

Answer VisualAge is available on:

- OS/2 Warp Version 3 or Version 4
- Windows 3.11
- Windows 95
- Windows NT 3.51 or 4.0
- AIX 4.1.4 or 4.2
- Sun Solaris 2.5.1
- HP-UX 10.10

Smalltalk Strategy

Question Is Smalltalk strategic?

Answer Smalltalk is a strategic language for IBM and will be developed as a standards compliant, open, multiplatform language. Smalltalk is an important and emerging object-oriented language in the industry. Its use in business and commercial application development is growing rapidly. IBM is chairing the effort to develop ANSI standards for Smalltalk, and a full range of third party consultants, tool vendors, and service providers are available to complement the offerings of Smalltalk vendors. IBM recognizes the value of Smalltalk and is delivering a full range of tools, education, consulting, and service offerings to address this important market.

Smalltalk and Java

Question Will Java overpower Smalltalk?

Answer While not disregarding the strength of Java, we must note that Smalltalk is superior to Java in many ways. The main strength of Smalltalk is its maturity. Smalltalk has been around for 25 years. By contrast, Java is in its very infancy. Smalltalk′s maturity has some important effects:

- Code quality. Smalltalk systems have been refined by several generations of programmers and have proved to be free of both coding bugs and, more importantly, ideological bugs.
- Legacy code. There is a huge estate of existing Smalltalk applications.
- Performance. Smalltalk is quite fast, thanks to the concerted efforts of many programmers during the last 25 years. Java needs an investment of similar effort to reach a comparable level of performance.

Configuration Management

Question How does VisualAge for Smalltalk support configuration management?

Answer OTI′s ENVY configuration management and version control system is included in the VisualAge for Smalltalk Professional and Professional Server products, precluding the need to order a separate product and managing compatability between VisualAge and ENVY. The October 9, 1995 issue of *InfoWorld,* states that : ″IBM did a superb job weaving Object Technology

Inc.′s Envy Developer into VisualAge, resulting in ummatched team development facilities.″

Packaging Support

Question How does VisualAge for Smalltalk support packaging?

Answer One of the enhancements to Envy found in VisualAge for Smalltalk is the one-button packager, which automatically excludes unnecessary classes from the application executable. The developer can also customize what is stripped out of the image to provide an even smaller executable.

Database Support

Question Which databases does VisualAge for Smalltalk support?

Answer VisualAge for Smalltalk supports a wide variety of database systems including the DB2 family, for example, DB2/2 for OS/2, DB2/6000 for AIX, and on Windows using CAE client support. Through connectivity services of DDCS, VisualAge for Smalltalk can access the DB2 family (including DB2 and SQL/DS and access to the AS/400). Oracle ODBC (with industry standard drivers such as Sybase, SQL Server, Informix) IBM intends to respond to additional customer needs for native support for other databases. In addition to any database driver supported by ODBC, the capability always exists within VisualAge to drop into the Smalltalk development environment to build any type of unique class library and parts you need.

Communication Protocol Support

Question Which communications protocols does VisualAge for Smalltalk support?

Answer Communications protocols are supplied in the VisualAge for Smalltalk Communications/Transactions feature and include TCP/IP, APPC, CPI-C, NetBIOS, CICS ECI, EHLLAPI, MQSeries, and RPC.

Legacy Code

Question How does VisualAge for Smalltalk handle legacy code?

Answer You can wrapper legacy COBOL, C, or C++ applications into parts by using the COBOL, C, and C++ support included with the base product. The IMS Connection feature enables you to wrap existing IMS transactions to be used as parts within the VisualAge for Smalltalk environment. IBM SOM and DSOM support enables you to reuse existing SOM and DSOM objects as VisualAge for Smalltalk parts..

Runtime Distribution Fees

Question Are there any run-time distribution fees for VisualAge or IBM Smalltalk?

Answer No. In contrast to many contemporary application development tools, the VisualAge and IBM Smalltalk license agreements allow you to freely distribute the run-time environment for the applications you write with VisualAge.

Distributing Smalltalk Compiler

Question Can the Smalltalk compiler be distributed with a run-time application?

Answer No. Redistribution of the Smalltalk compiler is not permitted. The compiler is a key component of the VisualAge application development environment itself. You must purchase one license for each compiler distributed. However, you can distribute the run-time without a fee.

Source Code Availability

Question How much of the Smalltalk source is available with the VisualAge for Smalltalk product?

Answer 99% of the source code is available. Some methods remain proprietary and are still hidden; however, those methods are typically code that is not required for application evelopment (for example, method code in the Smalltalk compiler.)

Upward Compatibility

Question Is an application written for one version of VisualAge for Smalltalk compatible with the environment of a higher version of the product?

Answer One of the features of the IBM VisualAge development environment is the ability to easily port existing VisualAge applications to other platforms and to easily upgrade the run-time environment for a new release of VisualAge. All you have to do is repackage the application with the new run-times and DLLs.

Common User Access Compliance

Question Does VisualAge for Smalltalk force Common User Access (CUA) compliance?

Answer CUA compliance is up to the programmer. However, with VisualAge it is a simple task to produce fully CUA-compliant applications.

Multiple Inheritance

Question Does Smalltalk support multiple inheritance?

Answer No. Multiple inheritance is a property that has been confused as a characteristic of the OO paradigm. It is not. It is only mildy useful in certain situations; invariably it causes objects to carry more attributes than necessary.

Porting Smalltalk Applications

Question How do I port a Smalltalk application to VisualAge?

Answer The recommended approach is to import (through a DAT file) or file in (through an ST file) the model code. The GUI code should be rebuilt in VisualAge because the effort to port may exceed the effort to rebuild.

Sources of Support

Question Once I purchased VisualAge for Smalltalk, what kind of support can I expect?

Answer A varietyof support is available, including:

- Sixty days of free IBM direct telephone support
- TalkLink forums
- CompuServe forums
- Web (for DAP members)
- Technical support, consulting, education, training, and more—from third parties

TalkLink

Question Which forums are available on TalkLink?

Answer Talklink′s public forums for VisualAge for Smalltalk are:

• IBMPARTS	IBM Parts for Visualage for Smalltalk
• VADIST	Distributed feature
• VAIMS	CICS/IMS feature
• VAINSTAL	Install questions
• VACOMLNG	Communications and Language features
• VA400	AS/400 Connection feature
• VAOTHER	Other topics not covered
• VAHOWTO	How-to questions
• VADB	Database questions
• VABUGS	Bug reports

CompuServe

Question Which forums are available on CompuServe?

Answer The CompuServe forums for VisualAge for Smalltalk are:

- Free-for-all
- Install/Begin
- Communications/Languages
- Database
- AS/400 Feature
- How-to
- Everything Else
- Bug Reports
- Third Party
- IBM Smalltalk
- Classifieds

Usenet

Question Is there a Usenet group dedicated to Smalltalk?

Answer Yes: COMP.LANG.SMALLTALK.

Resource Catalog

Question How can I get the resource catalog about vendors of components for VisualAge for Smalltalk?

Answer The document number of the Resource Catalog is G325-0813. If you work for IBM, you can order it through Puborder. The catalog is also accessible from the VisualAge home page on the World Wide Web:

http://www.software.ibm.com/software/ad/vastub.html

Be sure you are getting the latest catalog if you order a hard copy.

Learning Courses

Question Where can I find out about a suggested sequence of courses for application programming with VisualAge and IBM Smalltalk?

Answer The Object Technology University (OTU) offers a curriculum for VisualAge for Smalltalk. For more information visit their Web site at:

http://www.training.ibm.com/ibmedu/roadmaps/client/ot/

New Features in Version 3

Question What are the new features in VisualAge for Smalltalk Version 3?

Answer The functions of Version 2 plus:

- Development and run-time support for AIX Version 4.1.3.
- Open Database Connectivity (ODBC) support. This feature enables you to access data from a variety of databases and file systems which support the ODBC interface.
- Performance improvements through tuning and enhancements to the underlying virtual machine (VM): up to 5 times faster execution; machine code execution for Intel platforms
- Usability improvements through improved startup, consistency in dialogs, settings, font usage, additional parts, improved help, better and more complete examples.
- Enhanced database parts for greater usability and performance
- Upgraded IBM Database support for DB2/2 V2.1 with support for user-defined functions (UDFs), user-defined types (UDTs) and binary large objects (BLOBs).
- New portable controls (notebook, container, slider, and spin button) for multiplatform support and portability
- Portable drag-and-drop support
- VBX support (Windows only). This feature enables you to wrapper existing VBX controls and use them on the VisualAge for Smalltalk palette.
- Multimedia enhancements, including kiosk authoring enablement (OS/2 and Windows only), which includes support for hot spots and scene-to-scene navigation.

- VisualAge for Smalltalk, Distributed, enables you to develop a VisualAge for Smalltalk application on a single development platform and split the logic of the application to run on multiple execution platforms. Facilities are provided for remote debugging, performance profiling for optimal distribution, and other features that complement the development environment.
- VisualAge for Smalltalk, Database for Oracle for OS/2, for AIX, and for Windows, V3.0 provides access to Oracle data through native APIs. (Access to Oracle and other databases through ODBC is also provided in the base products.)
- VisualAge for Smalltalk, Reports for OS/2 and for Windows, V3.0 enables you to lay out reports, using the facilities of the VisualAge for Smalltalk construction-from-parts paradigm.
- Message Queue Series (MQSeries) support and remote procedure call (RPC) support as part of VisualAge for Smalltalk, Communications/Transactions for OS/2, for AIX, and for Windows V3.0.
- System Object Model (SOM) and Distributed SOM (DSOM) 2.1 Client support (OS/2 and AIX only).
- VisualAge for Smalltalk, AS/400 Connection for OS/2 and for Windows, V3.0 enables you to access additional AS/400 services, objects, and data, along with additional parts for VisualAge for Smalltalk.
- VisualAge for Smalltalk, IMS Connection for OS/2, for AIX, and for Windows, V3.0 enables you to wrap existing IMS transactions to be used as parts within the VisualAge for Smalltalk environment.

New Features in Version 4

Question What are the new features in VisualAge for Smalltalk Version 4?

Answer VisualAge for Smalltalk is IBM′s award winning enterprise application development tool that is the cornerstone of IBM′s VisualAge family of development products. VisualAge for Smalltalk allows software developers to create highly portable, scalable, multitier business applications that take advantage of emerging technologies such as the World Wide Web, ActiveX, OLE, and OLE controls (OCX) as well as seamlessly integrate existing legacy C, COBOL, CICS, DB2, IMS, and Tivoli systems. Version 4.0 takes the next step in enabling global enterprises to quickly construct line of business applications that are portable, highly scalable, and simple to maintain and fit easily into existing IT infrastructures.

VisualAge for Smalltalk is now available on more UNIX, Windows and OS/2 platforms than ever before. In addition to the existing OS/2 Warp V3.0, AIX V4.1.4/4.2, Windows 3.11, Windows 95, and Windows NT 3.51, new platforms supported are HP-UX V10.10, Sun Solaris V2.5, S/390 MVS, OS/2 Warp V4.0, and Windows NT 4.0. On Windows 95 and NT, OLE and ActiveX are now supported as well as new native user interface controls. Version 4.0 is now registered and certified for the ″Designed for Microsoft Windows NT and Windows 95″ logo.

Version 4.0 brings further performance enhancements, usability improvements, tool enhancements, and enhanced packaging support. Performance has been improved through the introduction of image components (ICs), which allow developers to create dynamically loadable modules that partition a Smalltalk application into smaller components that can be loaded and unloaded during program execution. ICs enable a more fine-grained packaging system and allow multiple applications to ″share″ the client′s run-time image, which improves performance and footprint and further promotes reuse. Performance of database applications will improve under Version 4.0, and DB2 applications can now be improved substantially by using static SQL. Tool enhancements include a more robust debugger, allowing for static and conditional breakpoints, and profiler enhancements that provide for easier fine tuning of an application′s performance. VisualAge for Smalltalk has also improved its external callout and callback support, allowing non-Smalltalk applications to communicate more easily with Smalltalk applications. Several Version 3.0 optional features have been moved into the base, including native Oracle support, Connection for Lotus Notes, Reports, a leading-edge object visualization tool, and Pro Server, providing the best value in the industry. Management of applications is improved by including parts that allow your application to interact with the Tivoli system management products. Also included with each copy of Version 4.0 is AppletAuthor, which allows you to create eye-popping Java applets that lead to dynamic, visually appealing Web sites without the need to know Java programming.

Additional Platforms and Windows Platform Exploitation

Platform choice has been a hallmark of VisualAge for Smalltalk, and Version 4.0 increases that choice fivefold. New UNIX platforms HP-UX (V10.10) and Sun Solaris (V2.5) have been added, and additional versions of PC operating systems are now supported: OS/2 Warp V4.0 and Windows NT 4.0. Finally, Smalltalk applications can be written for the S/390 MVS platform by using the new VisualAge Smalltalk Server for MVS product in conjunction with VisualAge for Smalltalk Version 4.0. With Smalltalk on the S/390 MVS system, mainframe programmers can achieve the same level of productivity and capability as those using Smalltalk on a PC or workstation. Using

Server Smalltalk for MVS, you can quickly build business logic in true object-oriented form on the MVS operating system. Support for IMS, CICS, DB2, native file system access, and batch programing are just a few of the features included. State-of-the-art cross-development tools allow you to develop the bulk of the application on a PC and later deploy it to MVS for the robust run-time performance we all expect from that platform. You can now create end-to-end enterprise objects, from client to mid-tier server to third-tier server, using a single language and development environment. For more information, please see the white paper titled ″Introducing VisualAge Smalltalk Server for MVS.″

For those of you who use the Windows platform, Version 4.0 adds additional power and function. A new OLE application framework provides support for incorporating OLE (OCX) and ActiveX controls in your application and makes it easy for you to write OLE clients in VisualAge for Smalltalk. Off-the-shelf OLE and ActiveX controls can be placed on the VisualAge palette, dropped on the application design surface, and visually connected to other parts, using the award winning construction-from-parts technology. If you want to write OLE clients, the framework provides high-level abstractions for the OLE container concepts, allowing you to focus on your Smalltalk application and not on the low-level OLE programming interfaces. Using the framework, you can write OLE container applications that support embedding, linking, in-place activation, drag-and-drop, and structured storage. You can also write OLE automation controllers in Smalltalk that can control third party OLE automation servers, such as Microsoft Word or Lotus 1-2-3 for Windows.

Windows 95 and NT 4.0 support many new or visually redesigned user interface controls, and so does VisualAge for Smalltalk Version 4.0. You will find support for the new system slider, progress bar, tab strip, toolbar, tree view and status bar. These are included in a new palette category in the VisualAge application builder for easy visual application construction, as well as in the widget class library.

Image Components

Developers of Smalltalk code are very familiar with the concept of an image. An image is a file that contains all of the user′s objects, and it is loaded and executed by the Smalltalk run-time system. In VisualAge for Smalltalk Version 3.0, single, stand-alone image files are the norm. Typically these image files are quite large, and copying and storing them requires a lot of resources because these images are not shareable, code that is common between applications must be duplicated in the image files that represent each application and in memory when each application is running. Although techniques have been developed to apply fixes to application programs in an image, the process is time consuming and difficult, especially when only a few applications, classes, or methods must be replaced in a large system.

In Version 4.0, ICs enable you to overcome these and other difficulties. ICs promote reuse, reduce memory requirements, make application maintenance easier, and promote porting. In Version 4.0, you can divide an image into separate IC files, which can be reused among independent applications. ICs reduce the memory required when multiple applications are running because the ICs are shared among applications using the shared memory facilities of the underlying operating system. Also, ICs use a compact format which results in smaller images that load and run faster. An additional benefit of the compact format is shorter time spent in garbage collection. Because ICs are replaceable, they make it easier to maintain and enhance applications. replaceable. You can update an IC with bug fixes or patches and replace the existing IC, not the entire image file. Provided the Smalltalk code in an IC is platform independent, an IC is portable, because the same IC file will load and execute on all platforms supported by VisualAge for Smalltalk Version 4.0.

Smalltalk Language and Tool Enhancements

Support for ICs is just one of several improvements that have been made to the Smalltalk language and tools in Version 4.0. Breakpoints have been added to the debugger, enabling you to more easily find, isolate, and remove bugs from your code. You can set a breakpoint on any line of Smalltalk code, and you will see a red dot in the margin of the browser indicating that a breakpoint is set. The executing program stops when the breakpoint is reached, and you can begin debugging at that point by stepping through the code, inspecting objects, and so forth. A conditional expression can be associated with a breakpoint, and the execution will stop only when the condition is true. Thus you can isolate those hard-to-find bugs that occur when the state or content of an object becomes incorrect, by setting a breakpoint that will only trip when that occurs. You can also set the number of iterations a breakpoint must be reached before the execution will be stopped, so that you can debug pieces of code that are executed repeatedly without stepping through each iteration manually. From the development environment you can display a list of all methods in which breakpoints are set and browse the method code from the list.

Namespaces have been added to the language, which give you control over the scope of globals in your application. Namespaces allow you to use a language construct that logically groups globals and assigns them a name. You can later selectively gain access to and include the collection of globals in the scope of your application by referring to its name. Thus namespaces provide the same type of scoping control for globals as application prerequisites provide for class references. They also provide an easier to use and more robust technique than was provided by ″toBeLoadedCode″ and ″wasRemovedCode″ in previous releases. A new query tool has been added to the development environment that lets you display all references to objects, methods, and globals that are outside the scope of your application, its namespace references, and its prerequisite applications. This tool helps you find and correct scoping and namespace problems and correctly prepare your application for packaging, especially when using ICs.

The performance analysis and tuning tools now monitor and collect more memory allocation information and enable you to easily create, run and compare the results of performance benchmarks. The memory monitor in Version 4.0 now shows individual pieces of allocated memory segments. You can quickly identify the portions of your application that are using a lot of memory, making it much easier for you to tune and improve that code. The benchmarking workshop now lets you collect and store performance benchmark test cases. You can make performance test runs using any and all of these test cases. Results can be graphically displayed and compared with previous runs so that you can quickly see the impact of your performance tuning efforts.

Better callback support from C to Smalltalk has greatly improved the interoperability between C and Smalltalk. This entry point can be passed to an external program when calling-out to it from Smalltalk. It gives that program an entry point to the Smalltalk object and selector it represents. In C you can map the entry point to a function pointer, making it very easy and natural to callback from C to Smalltalk. When combined with ICs, you now have powerful new options for packaging and delivering Smalltalk dynamically loadable application code that must call to and be called from programs written in other languages.

Database Performance Improvements and Enhancements

Major performance and database security improvements are in store when you use DB2 static SQL support, a new addition to VisualAge for Smalltalk in Version 4.0. Other database improvements include unified fields and samples, ″headless″ server support, and improved performance in applications using dynamic SQL.

In Version 3.0, dynamic SQL was used for all database access. IBM′s DB2 static SQL essentially allows you to compile your SQL statements and store them in the database. What is stored is called a *plan,* which contains optimized database access and manipulation code to execute your SQL statements. Comparing static SQL and dynamic SQL is very similar to comparing compiled and interpreted programming languages. Static SQL gives you compiled, optimized, high performance database access, while dynamic SQL is in some cases slower to execute but more flexible and easier to change. The Version 4.0 database support enables you to create, test, and debug your DB2-based Smalltalk application, using dynamic SQL, and then flip a switch and convert it to static SQL. Thus, you have the rapid application development style of an interpreted environment early in the development cycle, and the speed of a compiled environment when you roll your database application out into production. Dynamic and static SQL can be used in combination in the same application, allowing you to stage or phase in your cutover to static.

The performance gains you will realize when switching to static SQL vary and are application dependent. Database security for any application that uses static SQL is vastly improved over dynamic SQL. For users to execute dynamic SQL, he must have complete authority to the tables and data affected by the query. This can present an exposure if, for example, one of the tables contains sensitive data not included in the SQL SELECT statement. Users can use interactive query tools to view the sensitive data and bypass the SELECT statement in the application program, because they must have access to the entire table in order to run the SQL statement dynamically. This is an even greater exposure for other SQL statements, like UPDATE and DELETE. Not so with static SQL. User authorization may be given only to specific plans (compiled SQL statements), thus preventing the user from viewing or changing any data not included under the scope of the plan. Static SQL is one of the strongest features of IBM′s DB2 family, and now VisualAge for Smalltalk allows you to exploit that strength.

Unified fields now allow you to create applications that are largely database neutral. In Version 3.0, the database field classes, which represent data types in the database, are database vendor specific. The DB2 field classes differ from the ODBC field classes, both of which differ from those supporting native Oracle. The field classes have been refactored and unified, allowing you to easily switch the underlying database system without changing your Smalltalk application code. The database samples have been redesigned to exploit this, yielding a unified, database-neutral set of samples. A migration tool is included that will help you convert your application from using the old database specific field classes to the new unified field classes.

A ″headless″ application is one that typically runs on a server and does not have a GUI or an operator console. Headless application support has been added to the Version 4.0 database support so that you can package an unattended database server application that does not require a GUI or a console window. Concurrency has also been added to the database support. You can now develop unattended database server applications that simultaneously serve multiple clients.

Runtime performance of dynamic SQL applications has been improved up to 70% over the same applications running on Version 3.0. In Version 4.0, many database access related computations have been moved to application development time from run time. This change in Smalltalk has yielded significant performance improvements in overall database application run time. Run time performance improvements of 70% in SELECT, 30% in DELETE, and 25% in UPDATE statements have been realized by some applications.

Web Connection

With the Web Connection feature you can use VisualAge for Smalltalk to build applications that are accessible through the World Wide Web. You use the composition editor to build dynamic Web pages that provide the user interface for your application. You implement the program logic by making connections to visual and nonvisual parts and writing Smalltalk scripts, just as you would with any VisualAge application. Thus you can quickly add a Web client interface to your existing VisualAge for Smalltalk applications, or rapidly develop a new Web client from scratch that leverages your existing relational data and transactions. To the Web client support, Version 4.0 adds a grid visual part, ActiveX controls, and JPEG images. The server now supports DLL or shared library based interfaces for improved performance.

The new grid visual part enables you to build HTML tables. The tables can contain an arbitrary number of rows and columns, and cells of the table can contain other visual parts such as text, buttons, list boxes, and forms. Using the grid you can create clean, well-organized, visually appealing Web pages that provide table-based user interface layouts. You can easily embed ActiveX controls on a page, using the object tag. JPEG images are also supported. When these Version 4.0 enhancements are taken in combination with the existing Web client development capabilities, you can create dynamic, visually attractive Web clients.

In Version 4.0, server application performance is greatly improved over the CGI using DLL or shared library interfaces. These interfaces streamline the startup of the server application because an operating system process does not have to be created for each request made of the server, as it does with CGI. The DLL or shared library interfaces are available in Web server products supporting IBM′s ICAPI, Netscape′s NSAPI, and Microsoft′s ISAPI on any platform that is supported by both the Web server product and VisualAge for Smalltalk. The CGI interface continues to be supported.

Usability

Drag-and-drop or direct manipulation of objects in a GUI is one of the most intuitive and efficient methods for interacting with a software application. Version 4.0 adds platform drag-and-drop, which allows you to drag objects between applications written in Smalltalk and any other drag-and-drop enabled application or any desktop object, on any platform. You can now let the users of your VisualAge for Smalltalk applications avail themselves of the improved usability and efficiency of drag-and-drop. Enabling drag-and-drop for visual parts is as simple as changing a property (setting) on the part.

Speaking of property changes, the most obvious and pervasive usability improvement in Version 4.0 is the new property view for parts. This view appears whenever you want to display or change the properties or settings for a part, typically by double-clicking on the part. Gone is the tedium of flipping notebook pages to find and change various properties and their values. Instead you use a scrollable two column table, which displays the property name on the left and its value on the right. Your productivity is improved greatly when changing several properties for a part as the table is easy to use.

Optional Features

Several Version 3.0 optional features have been moved into the base, including native Oracle support, Connection for Lotus Notes, Reports, two of the tools from the Distributed feature, and Pro Server, providing the best value in the industry. Management of applications is improved by including Tivoli Connection parts in the base that enable your application to interact with the Tivoli system management products.

The two tools from the Distributed feature that are now included in the base enable you to analyze your application′s performance. The Object Visualizer helps you analyze object activity and interaction in your application. You can see how many instances of a class exist at any time, watch a visual representation of message traffic between objects, identify which objects are most and least busy, and determine which clusters of objects should be local to each other based on the amount of message traffic between them. The cluster view dynamically displays the objects and clusters as they form and re-form, so you can see graphically the cohesion between your objects. The Event Profiler is a browser that shows all of the events (method calls) sent among selected objects. You can use it to browse each event and trace the method call stack that led up to it. Using these tools you can better understand the message traffic and interaction between the objects in your application and then tune your application for peak performance.

IBM′s Tivoli family of system management products helps you deploy, monitor, update, and manage software applications in your enterprise. Tivoli Connection, provided in the base, includes two facilities for enabling the development of VisualAge for Smalltalk applications that are management-ready for Tivoli. The first is a set of parts that enable you to easily build an application that generates events and sends them to the Tivoli/Enterprise Console. The new parts are available on the parts palette when you install Tivoli Connection. The parts allow you to instrument your application with events that communicate information about the status and health of your application. For example, you can send events that identify the startup and shutdown of tasks and the occurrence of error conditions. These events are sent to the Tivoli/Enterprise Console, which provides centralized monitoring and management of system information and events.

The second facility is a stand-alone program, the Tivoli Developer Kit, that enables you to define the management characteristics of your application to the Tivoli Management Environment. The Developer Kit assists you in defining management information about your application by using Tivoli′s Application Management Specification. You can specify your application′s components, dependencies, distribution and installation details, and operational tasks that are needed by the Tivoli Management Environment.

For developers of VisualAge applications in a complex, network computing environment, Tivoli Connection provides the support for building management-ready applications.

AppletAuthor

AppletAuthor is included with each copy of VisualAge for Smalltalk Version 4.0. AppletAuthor is a Java authoring tool that lets you create applets without writing a line of Java code. By combining components (Java beans) from a palette using simple mouse movements, Web site authors can bring to life their site. Once you create your applets, you can include them in a Web site created with any number of Web page or site creation tools, including the Web Connection feature of VisualAge for Smalltalk.

Chapter 2. General Information

It seems that every Smalltalk programmer looks for tips on good Smalltalk image maintenance. This chapter covers questions and answers about the image and general housekeeping.

Cleaning Up the Image

Question How do I clean up my image?

Answer The never-fail approach to reducing your image size is to rebuild it. Export all of your stuff from the old image. Then get a virgin image (you may need to get one from the installation CD) and load all of your applications. It′s good practice to keep a base image around that has all of the IBM and third-party applications already loaded. When you need to rebuild, simply load your applications to the base and save under another name. Be sure to keep your base image updated with any fixes.

Creating a New Image

Question How do I create a new image?

Answer

- With the Standard Edition of VisualAge for Smalltalk, if you have backup copies of the files sources.es, changes.log, abtdev.pom and image, then you can copy these files into your VisualAge for Smalltalk subdirectory and launch. If you do not have backup copies of these files, then it will probably be best to reinstall to get a new image. To avoid reinstalling in the future, make backup copies of these four files and keep them where you can access them. Be careful to file-out applications you would like to keep!
- If using VisualAge for Smalltalk Team Version 2, change into your VISUALAG directory on the server. From there, change into either the OS2 or WIN subdirectory, depending on whether you are using OS/2 or Windows. Then change into the client subdirectory. Within this subdirectory, there is an image file that you can copy into your client VisualAge subdirectory. You can either rename your old copy or delete it, depending on whether you want it backed up or not.
- VisualAge for Smalltalk Professional Version 3.0 no longer supports two-step centralized installation, so you must ensure that you always

have a backup copy of the virgin image (or of the image with only the features installed).

Reducing Image Size

Question My image has grown significantly in size, yet I have added little to it. At the same time, performance seems to degrade. Why is this happening, and can I fix this?

Answer Close all VisualAge windows except the Transcript. Type

Smalltalk

```
Dependents
```

in your Transcript, select, and inspect it. If more than *self* and *Behavior* is in the leftmost window pane, type

Smalltalk

```
Object abeReinitializeDependents
```

in your Transcript, select, and execute it. Then save your image. The size of your image should reduce back to what it was initially.

Object abeReinitializeDependents frees up the resources associated with any existing dependents. When you use a part that is not subclassed from AbtObservableObject, and you establish connections to this part (so that other parts are notified when this part changes), you create dependents.

Here′s something else to check. From the Transcript, select **Smalltalk Tools** → **Open Debugger**. Then, from the debugger, select **Processes** → **Debug Other**. Typically, there are two. If there are many more, select and terminate them.

If you try these things and your image is still quite large, try this from the Transcript (save parts and close all windows first!):

Smalltalk

```
EmSystemConfiguration new abtScrubImage
```

This reinitializes dependents, removes and terminates active processes, and reinitializes common widgets. It closes all VisualAge windows and returns the System Transcript window back to its default message.

Preventing Image Growth

Question What should I look for if my image size is increasing over time?

Answer Execute *System abtScrubImage*. Does your image size shrink considerably? If so, it is time to figure out where you are hanging onto resources. There are several possibilities here. Check the following:

- Open a debugger. You can open a debugger from the Transcript by selecting **Smalltalk Tools** → **Open Debugger**, or from the VisualAge Organizer, **Tools** → **Debugger**. To see the Debugger item in the Tools pulldown menu, select **Options** → **Full Menus** in the Organizer. From the Debugger′s pull-down menu, select **Processes** → **Debug other**. There should be only two processes, the CwAsyncIOProcess process and the Idle process. If there are others, you should attempt to figure out what objects are suspending them and holding onto them.
- With no windows but the Transcript and VisualAge Organizer open, inspect the AbtEventDependents classVariable of AbtCLDTAdditions. There should be only two entries. If there are more, then you have dependencies that are not being released. This most often occurs when sending *closeWidget* as opposed to performing the action *closeWidget.* The selector corresponding to the *closeWidget* action is *closeWidgetCommand*. Sending *closeWidget* will prevent dependencies from being freed.

AbtScrubImage

Question I′ve heard many times about *abtScrubImage*, that it helps me reduce my image size. What does it do, actually?

Answer First of all, it clears out the dependents dictionaries. Then, it terminates all the running processes except the currently activeProcess. It then resets the Organizer applications view, and restarts it. And, finally it does *CommonWidgets reinitialize*.

Ghosts

Question How do I get rid of unwanted ghosts (multiple instances of my parts hanging around).

Answer I don′t know what you are doing to cause them to hang around (do you save your view instances in a global or class variable?), but you can scrub VisualAge with

Smalltalk

```
System abtScrubImage.
```

It clears out the dependents′ dictionaries and a few other things that can get messed up after lots of walkbacks. Look at EmSystemConfiguration>>#abtPrimScrubImage: for details. If you still have instances after running the scrub, check who is referencing them with #allReferences. In addition, make sure that, if you are closing your views programatically, you are sending *closeWidgetCommand* and **not** *closeWidget*.

Image Growth after Packaging

Tip I would bet that your image size has grown as the result of saving your image after doing the packaging. I′ve noticed that something created by the packager gets a global reference and is therefore never garbage collected. To avoid this, I would advise saving your image immediately before packaging, then package and exit without saving the image.

Keeping Multiple Versions on the Same System

Question Can I keep both VisualAge for Smalltalk 2.0 and 3.0 on the same machine?

Answer Yes, they can coexist on the same machine. You did not say whether you were on OS/2 or Windows. However, since Version 2 and 3 do not share DLLs of the same names, keeping both is fairly easy. Just put the ″.;″ (period and semicolon) in your path (or libpath) as the first thing so that files are picked up from your current subdirectory. This may mean consolidating some of the files in the same directory. Also watch the *abtpath*, it can reference several subdirectories as well. Libraries can be kept on the same machine, just make sure you are pointing the clients to the right one.

Replacing the Default Icon

Question How do I replace the default VisualAge icon with my own?

Answer If you do not want a splash screen everywhere *abt.exe* is mentioned, replace it with *nodialog.exe*. Abt.exe can be found in your VISUALAGE subdirectory while nodialog.exe can be found in your VISUALAGE\DIALOG subdirectory. Copy the abt.rc, abtrc.h, and the abt.exe (or nodialog.exe) files to your run-time application directory from the dialog directory. Do an edit of your abt.rc and change the following line

```
POINTER ID_ICONRESOURCE visAge.ico
```

to

```
POINTER ID_ICONRESOURCE iconName.ico
```

- If you *do not* want a splash screen, you can erase the rest of the abt.rc file. (The rest of it pertains to the splash screen.)
- If you *do* want the VisualAge splash screen, leave the rest as is.

Once you have saved the changes to your abt.rc, compile your resource script file and merge this into your abt.exe (or nodialog.exe) with the following command:

```
rc abt.rc abt.exe
```

or

```
rc abt.rc nodialog.exe
```

You can find a resource compiler in the OS/2 toolkit. Once this is done, you can run your application against your image.

Escaping an Infinite Loop

Question How do I find a way out of an infinite loop? Does VisualAge have something like CTRL-C?

Answer VisualAge allows you to hit Alt+SysReq. Hold this until you hear a low beep. A debugger will appear, signaling that the process has been terminated or interrupted, with the error string *″User Break.″*

File-In and File-Out

Question How do I file-in and file-out code programmatically?

Answer You can write Smalltalk scripts that will file-out code for you. Doing this, you can make the scripts as sophisticated as you want. Use class CfsWriteFileStream to create your file. Examine the class EmFileOutInterface. This class has several methods which you will find useful, such as *fileOutOn:*, *fileOutSourceOn:from:*, and so on. Do not forget to generate archival code when you save the part.

Separate Threads

Question Can I run VisualAge for Smalltalk from separate operating system threads?

Answer True multithreading has been tested in AIX, but no specific release plans are available. OS/2 and Windows releases do not support true multithreading. OS/2 and Win95/NT may support multithreading in a future release, but I have been unable to find out specifics so far. The only ways to do parallel processing today are:

- Launch multiple copies of VisualAge for Smalltalk on the same or different machines
- Use the VisualAge Distributed feature along with launching multiple copies.

Distributed would be much easier to coordinate and is quite elegant, but does require a larger image and adds complexities such as TCP/IP setup if you don′t already have it.

Capturing Stack Information

Question How can I capture the stack information when I have a walkback?

Answer In the debugger window, use the *Write Stack Trace Text To File* option from the Stack pull-down. This writes the information to a file of your choice.

Renaming a Class

Question How do I rename a class ?

Answer The *VisualAge for Smalltalk User′s Guide* discusses this a little bit. Filing out and filing in classes to rename them works fine.

Another way to rename classes is this:

- Change to the Script Editor for the class.
- Change the class name in the class definition in the bottom window of the Script Editor to the desired name.
- Save the change. A new class with the desired name will be added to the application browser. It is an empty class for now.
- Select all methods showing in your method window, choose **Move Methods** → **To a new Class** from the **Methods** pull-down menu. Type in the new class name when prompted.
- Be sure to move all class and instance methods—both public and private—over to the new class.
- Test your new class to make sure it behaves as the other one did.
- When you are positive that your new class works, you may delete the old one if it is not needed anymore.

Environment Variable

Question How do I get the value of an environment variable defined in config.sys?

Answer There is a private method called *abtScanEnv* in EsString that should do what you desire. To use it, try executing the following code:

Smalltalk

```
′PATH′ abtScanEnv
```

Registered Connection

Question How do I remove a registered event-to-script connection?

Answer Try sending *destroyPart* to the connection you dynamically created.

Loading Different Image

Question How do I specify a different image from a command line?

Answer You can use the -i parameter, for example:

```
abt.exe -id:\va30\image
```

Deferred Update Part

Question How do I use the Deferred Update Part?

Answer **Opinion One**

Deferred update parts exist only to buffer editing of control widget values. Although there is sufficient demand for things like buffering, the practical use of the deferred update part is rather limited. The limitations are twofold.

First, the part assumes that your objects are just objects that contain data attributes. That is, assume you have the object RightAngleTriangle with attributes a, b, and c, where c is calculated based on a and b. Using a deferred update part, you will never see attribute c calculated until the user hits the apply button, whereas without the deferred update part, the attribute c will be updated as either a or b changes.

Second, if you allow your users to open different views on the same object, they may have made some changes on a view and not applied them. When they then open a second view on this object, it comes up without the changes they have just made on the first view, and things look inconsistent.

In my opinion, you have a couple of strategies depending on your application, but the best is to have the view ask for a copy of the object (this may even be a specific copy for this view). Have the view be a view of the real objects and then you get all the behavior right away that the object should exhibit. If the user cancels, it is the view′s responsiblity to set the object back into its original state.

Opinion Two

To the naked eye, the deferred update parts can be of little use once your objects consist of a bit more than primitive attributes. But, believe it or not, we found ways around the problem. You just have to look closely at the AbtDeferredUpdatedManager and what it does. Your domain object can implement two different class methods which the Manager sends to the class during the creation of the deferred update part instance. These methods respond with a class name which the Manager uses to instantiate the deferred part, and there you could implement some of the behavior such as changing attribute b when a attribute changes. The object you will be dealing with at that time is the update operation whose target is your object which is undergoing a change. Hence, you can ask your object to recalculate some other value based on the attribute changed.

My recommendation would be to attempt to use deferred update parts, and enhance them when needed. The benefit is that there is so much infrastructure there. The Command or the Momento pattern is nice, but not as easy to implement from scratch. Also, if one uses VisualAge correctly— that is, having a form work with a single object that includes nested forms which in turn edit other complex attributes of your object— deferred update parts work great.

As for seeing noncommited object changes when a different editor is opened on the object currently being edited, I say it is up to your personal preference. One must have a pretty smart application-based transaction framework to support such behavior. What about different views editing the same object? Getting the transient state of an object is wonderful, as long as you are willing to pay the price for the added complexity.

Renaming a Visual Part

Question How do I rename a visual part?

Answer The current implementation of VisualAge for Smalltalk provides no mechanism for renaming a part once it has been created. The copy facility is flawed, in that certain connections retain a reference to the previous class name.

In Smalltalk, you can rename a class by filing it out, exchanging the names in an editor, and filing it back in. A little more work must be done in VisualAge (even Smalltalk classes with a public interface will not work using the file-out, file-in technique).

Solution 1

Open the Composition Editor for your class and select **Generate Archival Code** from the file menu. File out your class and in an editor find and replace occurences of the old class name with the new class name. Save the file and then file it into your image.

An unfortunate side effect of this technique is that the archival code will stay connected to your class forever. Whenever you save the part in the Composition Editor, you will be asked if you want to regenerate archival code.

Solution 2

Create a new visual part. In the new part dialog, change the superclass to the name of your existing class.

When the Composition Editor editor opens, make a small change (move some widget a bit to enable the **Save** menu item) and save the part. This generates the magic *abtBuildInternals* method and all the invisible stuff needed. Change to the Script Editor. In the class definition, change the superclass from the name of your old part to *AbtAppBldrView* and save the definition.

Just to be safe, whenever you change the superclass of a visual part, close the editor and reopen it.

Hover Help for Buttons

Question How do I customize hover help for my push buttons?

Answer Push buttons with a graphical label don′t contain textual information. This textual information is what is displayed by hover help and, in the absence of textual information, it displays the part name. The part name, unfortunately, cannot contain special characters, such as accented characters for French language.

When defining the push button, specify both the label text and the graphics descriptor. In the settings editor for the push button, first define the label type as text and enter the text you want to see for hover help. then, define the label type as graphical and specify the bitmap you want to display.

Customizing Connection Menu

Question Can I customize the connection pop-up menu for a visual part?

Answer Yes. You must specify which options should be shown on the context menu and which can be accessed using more. The *preferredConnectionFeatures* class method for the specific part that needs this menu adjusted must be modified as follows: (using a, b, c, d as the list of connections)

Smalltalk

```
preferredConnectionFeatures

 ^#(a,b,c,d)  ″this shows a,b,c,d and More on the menu″
```

Live Updating of Images

Question I am creating a settings page that features a form in the center that will be used to display a GIF file. Right above the form is a file picker that returns the path and file name of the graphic the user picked. (It is basically a recreation of the background image selector on the form part in Web Connection). The system fails to update the graphic properly however. I have torn off the backgroundDescriptor from the form and tied its module name attribute to the output of the file picker. When I open the page for the first time the image is correct, (whatever the user selected last time) but I am unable to get the image to update live. Any ideas? How can I get the *updateOperation* to fire before the user actually clicks the OK button?

I tried to examine the logic, but I am stymied by what look like incomplete connection diagrams in the settings view parts. The edges are just floating with no apparent sources or targets. What can be done to get around this?

Answer You cannot see the form in the settings page view because the bottom and right edge attachment offsets are zero. To fix this, do the following:

Edit the part in question, for example, AbtIntegerDatatypeSettingsView. Normally you would select the form and open its settings page to change the offsets. However, as you cannot see the form, you cannot select it, forcing you to change the offsets manually.

First, open an inspector on the *partBuilder* as follows. Press Alt-Shift-Mouse button 1 anywhere on the Composition Editor. Then inspect the *partBuilder*, then the *subPartBuilders*, then the entry named Form1 in the AbtOrderedDictionary. This should lead to an AbtInternalSubpartBuilder. Inspect the attribute settings and then the *framingSpec* entry. Open inspectors on the *bottomEdge* and *rightEdge*. The AbtEdgeConstraint should have an offset of 0. Change it to something like 100.

After closing all the inspectors save the part. This requires tricking VisualAge into thinking that you have changed something, so you will need to do something innocent like move the center of a connection slightly or move the *updateOperation* variable a pixel or two. After saving, exiting, and reentering the part, the form should now be visible allowing you to work with it.

The problem with the stale image may be related to the fact that the *updateOperation* variable is just a deferred part. Deferred parts are not instances of the class they are deferring and are just a buffer for public interface attribute values. Thus, any behavior you may have imposed on your part to respond to a method such as image will never affect your object. The solution is to add your own variable and code as though the settings page were a normal Composition Editor.

Passing Parameters Between Windows

Tip Your VisualAge design should use the dependency mechanism. For example, you could have an attribute-to-attribute connection from a nonvisual (model) object attribute, using its public interface, to the list in the first GUI window. The second window could have a variable promoted to its public interface that receives the model object from the first window. It is this object that is receiving the edits in the second window. When the changes occur (as when an OK button is clicked), the model object state changes and its dependents are notified (including those in the first window). The refresh will then occur after the changes to the model object state.

Generating Archival Code

Question What application should you specify when generating archival code ?

Answer Create a new application, such as MyArchivalApplication, and generate archival code to it. You don′t want to generate archival code to the application that you plan to package and distribute to your customers (it will be larger than it needs to be). You can always delete the generated methods if necessary.

To create your archival application, do the following:

1. From an application manager, create the archival application. Change its prerequisites to include AbtBuildViewsApp, plus the application you are generating code for.
2. Open the part in the Composition Editor or Public Interface Editor.
3. Select **Generate archival code** from the **File** pull-down menu.
4. Select your archive application′s name as the target for the generated code.

Note: We recommend a one-to-one correspondence (an archival application for each one of your applications).

> **Remember**
>
> If you want to share code, you will have to file out *both* applications— the original and the archival.

Deleting Parts with Instances

Question How do I delete a class or part that still has instances?

Answer To remove instances of a class, execute:

Smalltalk

```
<MyClass> allInstances do: [ :anInstance | anInstance become: nil ].
```

Although this method of removing instances does work, and all of us have resorted to it at some time or another, we should warn you that it is dangerous. You will usually get away with this, but it can also cause walkbacks and even system crashes or meltdowns. Use it at your own risk.

Change Table Format

Question How can I change the dashes on a table to a line?

Answer You can use the following code:

Smalltalk

```
setGridLineStyleAndColor
 ″Set the line style and color appropriate for the grid″

 self foreground: self clrGrid.
 lineStyle == AbtTWLineStyleGrid
    ifTrue: [ ^self ]. ″already done″

 self gc setLineAttributes: 1 ″line width″
    lineStyle: (self useDashedLines
       ifTrue: [ LineOnOffDash ]
       ifFalse: [ LineSolid ])
    capStyle: CapButt
    joinStyle: JoinMiter.
 lineStyle := AbtTWLineStyleGrid.
```

The table defaults to having solid lines. To change the table to have dashed lines, *true* needs to be sent to the *useDashedLines* method. This can be accomplished by connecting the *aboutToOpenWidget* event of the table to a script connection with similar code:

Smalltalk

```
dashedLines
  (self subpartNamed: ′Table′) useDashedLines: true.
```

Error Messages

Question How do I find information on fatal errors?

Answer From the System Transcript, display the following code:

Smalltalk

```
SystemPrimitiveErrors keyAtValue: errorNumber

Example: SystemPrimitiveErrors keyAtValue: 33
```

If you highlight this code and choose Display, the following is returned: *′PrimErrImageFileOpenError′*.

Removing Archival Code

Question How do I remove archival code from my base application if I generated it here instead of to an archival application?

Answer Generating archival code to a new application will not move methods to the new application if they have previously been generated to a different application. This is important, especially when packaging your application to an image. You should not package archival code to a run-time image.

You may either delete the methods below from all classes in your application, or move them to the new archival application. If you delete them, be sure to generate archival code to your new archival application after deletion.

To move the archival methods, select the **Move methods** choice from the **Classes** pull-down menu of an Application Browser or Classes Browser. When prompted where to move the method, type the name of the target application and class. The following methods are archival:

- Public Class Method
  - *calculateInterfaceSpec*
  - *calculatePartBuilder*
- Private Class Methods
  - *add...PartBuilder:*
  - *codeGenerationParameters:*
  - *connect...*
  - *converter...PartBuilder:*
  - *test*

Apply Push Button

Question There are unlisted actions for both an okay and cancel push button on a part′s custom settings view, but how do you handle an apply push button ?

Answer If you are creating a VisualAge part, and want the custom settings view to have an apply push button, add an event-to-script connection from the push button *clicked* event to the following script:

Smalltalk

```
apply

  | op newOp |

  op := ((self subpartNamed: ′updateOperation′) value) execute.
  newOp := op target customSettingsOperation
     stackHolder: op stackHolder;
     defaultValuesSource: op defaultValuesSource;
     target: op target.
  (self subpartNamed: ′updateOperation′) value: newOp.
```

Inspector

Question How can I quickly open an inspector on a part in the Composition Editor?

Answer Do Alt-Shift-Click1 on a part in the Composition Editor.

Multimedia

Question Where do I find examples of using multimedia in VisualAge?

Answer There are multimedia applications on the VisualAge CD. Also, when you load the Multimedia and Multimedia Samples features, you will get an application named AbtMultimediaSamplesApp in your Organizer window. In this application, you will find multiple sample parts that you can run. For example, you can run MtTravelAgencyMainView. There are also parts for browsing videos (look for .AVI files in the VISUALAG\SAMPLES directory), and browsing sounds (look for .WAV files in the same SAMPLES directory).

Default Memory

Question Is there a way to change how much memory gets allocated by VisualAge? It appears that 2 MB is the default.

Answer The following will work for Version 2 (see page 38/39 of the Smalltalk User′s Guide for memory options in Version 3). When you call the packaged image *abt.exe -iMyImage*, add the following parameters:

-mi500000 This gives the memory an increment of 0.5 MB at a time, and is what was defaulting to 2 MB.

-mo5500000 This gives the initial memory allocation to the image. As a rough guide, take the previous *mi* figure away from the initial size of the image, for example, 5.5 MB.

The full call would then be: *abt.exe -iMyImage -mo5500000 -mi500000*

Storing Settings

Question I want to be able to store current settings into a file just prior to closing an application. The goal is so that next time the same VisualAge application is opened, it will provide the user with the same settings as the last access by first reading the file with the settings. How would I do this?

Answer One way to do this is by using the objectDumper and objectLoader. Create a class called *settings* which contains all the settings attributes you will store persistent in the file. To save the settings in the file, use:

Smalltalk

```
ObjectDumper new
   unload: <settingsInstance> intoFile: <aFile>
```

To load the settings from the file use:

Smalltalk

```
settings := ObjectLoader new loadFromFile: <aFile>
```

Connect the *save* action with *aboutToCloseWidget*, the *load* action with *aboutToOpenWidget*. The objectDumper unloads the complete instance into the file in a noneditable format. For redistributing, you also need the ABTSWP30.DLL.

Extending Classes

Question Why do I get a prompter asking me to create a scratch edition of the base VisualAge application when I want to extend an existing method?

Answer Extensions can only add behavior to an existing class (for instance, adding new methods to add functionality). You cannot alter existing methods as an extension. The only way to alter existing behavior (alter existing methods) is to subclass and alter the methods you want to do something different in your subclass.

Running VisualAge from External SCSI Drive

Question How can I copy the entire VisualAge installation to an external SCSI drive and run it from there?

Answer Just copy the whole VISUALAG directory structure to the new drive and use a command file to setup VisualAge to run from a directory (so you don′t have to mess with CONFIG.SYS), for example

```
@echo off
SET BEGINLIBPATH=x:\VISUALAG\DLL
PATH=x:\VISUALAG;%PATH%
SET HELP=x:\VISUALAG\HELP;%HELP%
SET ABTPATH=x:\VISUALAG\ABT
```

In addition, you′ll need to update ABT.CNF and NLS.CNF (I make the path references relative to the current directory).

Damaged Classes

Question How do I get rid of damaged classes?

Answer Version and release your classes, applications, and configuration maps. Then, get a fresh image and reload your stuff.

Extending Classes

Question How do I extend classes in the Organizer?

Answer You can extend classes in the Organizer in one of two ways. Both ways require you to be using full menus. To get full menus, click **Full Menus** from the **Options** pull-down menu.

- Option 1:

  Select the application you want to extend the class into. From the **Parts** menu, click on **New** → **Extension** and enter the name of the class you want to extend.

- Option 2:

  Select the class (part) you want to extend. From the **Parts** menu, click on **Extend** and enter the name of the application you want to extend the class into.

Modifying the System Menu

Question I would like to modify the System Menu, the menu you get when you click on the VisualAge icon at the upper left corner of a window, for an application. The problem I have is how to connect an appropriate action to the new menu items.

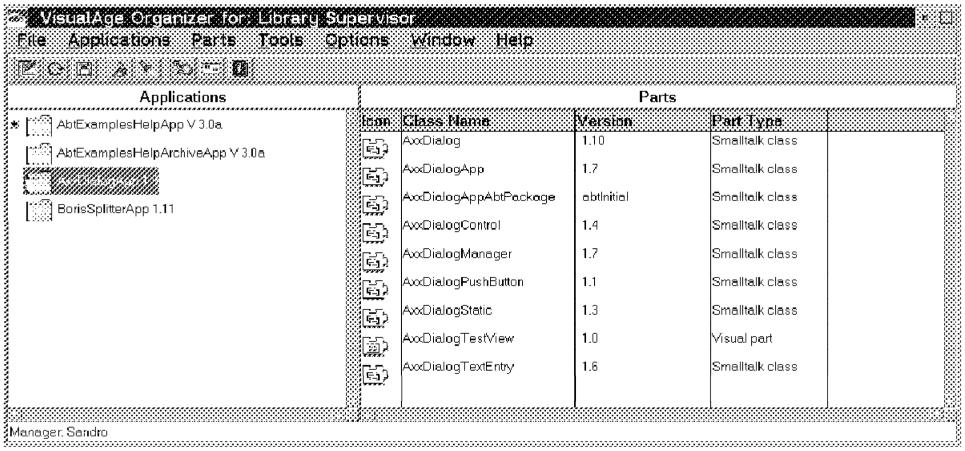
Answer Here′s an example that shows how you can change the system menu. Make sure to add PlatformWidgetsConstants and the CwConstants pool dictionary to your class definition first. Note that the second parameter of *hook:with:* is the method that will run when the menu item is selected, and it must take one parameter (even though you do nothing with it). The example also highlights the use of an accelerator key.

Smalltalk

```
| menuItem key |
menuItem := (self subpartNamed: ′Window1′) yourself
  primaryWidget
  osWidget
  systemMenu
  createMenuItem.
menuItem label: ′my item′;
  owner: self;
  accelText: ′Alt+F2′;
  hook: OSxSelect with: #myAction:;
 yourself.
key := VkToKeySym
  keyAtValue: XKF2
  ifAbsent: [nil].
menuItem
  alt: true
  shift: false
  control: false
  virtual: true
  key: key.
```

Organizer Views

Tip You can switch to different views in the Organizer. For example, it can look like this:



Applications

- AbtExamplesHelpApp V 3.0a
- AbtExamplesHelpArchiveApp V 3.0a
- [illegible]
- BorisSplitterApp 1.11

Parts

Icon	Class Name	Version	Part Type
	AxxDialog	1.10	Smalltalk class
	AxxDialogApp	1.7	Smalltalk class
	AxxDialogAppAbtPackage	abtInitial	Smalltalk class
	AxxDialogControl	1.4	Smalltalk class
	AxxDialogManager	1.7	Smalltalk class
	AxxDialogPushButton	1.1	Smalltalk class
	AxxDialogStatic	1.3	Smalltalk class
	AxxDialogTestView	1.0	Visual part
	AxxDialogTextEntry	1.6	Smalltalk class

Manager: Sandro

File in Use

Question What might cause a ″File In Use″ error when trying to read (only) a file in VisualAge?

Answer Normally, this is caused by an access rights problem outside the VisualAge domain. Make sure the server access allows concurrent read-only rights to clients. Check this out by making sure you can see the files and file contents from any program on the client system. In VisualAge, make sure the code uses the ORDONLY access mode. Do not use any of the following: OWRONLY, ORDWR, OCREAT, OAPPEND, OTRUNC, or OEXCL.

OS/2 Shutdown

Question Why doesn′t OS/2 shut down until all running applications have terminated?

Answer In general, OS/2 won′t shut down until all running applications have terminated to prevent the user from possibly losing data. This is an OS/2 behavior for any application, and is not unique to VisualAge.

BRW Format Generator

Question Is there a generator that produces a .BRW format for help?

Answer Use an ASCII editor—or Smalltalk Workspace—to develop a .BRW file. We gave the source files the extension .BRW for browser to distinguish the files from other types of files). The BROWSER.BRW file describes the tags used to specify fonts, lay out a page, display a graphic, and so on. To see how we used the tags, open an ASCII editor on BROWSER.BRW or VAXAMPLS.BRW. To see the formatted text (or view your own file), test AbtHelpBrowserView and load the file. AbtExamplesHelpBrowserSubApp contains the Smalltalk code that parses the tags. We′ll probably change our code so it reads HTML tags, and you might consider doing something similar for your own browser.

Loaded Method

Question Is an application′s *loaded* method invoked when it is run as an executable? Do you need to do anything special if you want to ensure that it will be run?

Answer The *loaded* method is a class method called only when the class is loaded into the image. Its usual purpose it to complete the task of hooking the class into the environment and setting up class variables and such. It is not called at execution time.

Missing Character Set

Question What should I do if I get a ″missing characterSet translation table, ibm-NNN″ error?

Answer You need to customize the ABTRULES.NLS file to include the 862 code page. Do this by copying and pasting all 862 code page table definitions from ABTRULES.ALL file to ABTRULES.NLS. This is a null table that doesn′t really do anything. Eventually, you will need a table with the proper values if you really want code page translation to work for your system.

Event Triggering

Question I created a Smalltalk class and defined in it an event in its public interface. In this class, I have only one method that signals this event under a certain condition. However, when running the code, this event is being fired and I am pretty sure I didn′t fire it. Tracing the code in the debugger reveals that VisualAge has a collection of Abt***Spec and it is firing the event from this collection. Is it the design of VisualAge or am I not defining the event correctly? I do not want the event to be triggered unless I use the *signalEvent:* method.

Answer Since this is a Smalltalk class (that is, a subcass of Object as opposed to AbtObservableObject or AbtPart), you may want to look at the method *featuresAffectedByAction:*. In order to handle classes that normally don′t signal events in all cases, this method defaults to signal all events for any action. To change this behavior, override the method to indicate exactly which attributes should be signaled for a given action.

Part not Visible

Question What does it mean for a part to not be visible from an application?

Answer If you get a message about some class not being visible, it means that you are either trying to extend or subclass it in an application that does not prerequisite the application containing that class.

Removed Class

Question I got the following debug message from a script: “Removed Class: ClassName does not understand method.” What did I do wrong and how can I solve this problem?

Answer This problem in VisualAge is caused by deleting a class without removing all method references to it. You should be able to do ″Browse References...″ and find all the references to the class that no longer exists (if you deleted, then recreated the class, simply recompile the methods referencing it).

If this does not work for you and if you are in the Team environment, it is probably best to get a new image and reload your application. We have seen this before, and that is the fastest and best way to get rid of it.

Synchronizing Attributes

Question I have two classes, A and B, and I make an attribute-to-attribute connection of one attribute in class A to another attribute in class B.

Suppose I have methods in both classes that reference these two attributes. When the attribute of A or B changes, do these methods get executed simply because they reference (by using the *getSelector*) these two attributes?

Answer With an attribute-to-attribute connection, class A will signal in a method that its attribute has changed using the *changeSymbol* defined for the attribute:

Smalltalk

```
(self signalEvent: #attributeOfAChanged)
```

This signaling of the event will cause the *AbtAttributeToAttributeConnection* to get the value of class A′s attribute (using the *getSelector* defined for it) and set the value in the instance of class B (using the *setSelector* defined for class B′s attribute).

Keeping the Packaged Image Small

Tip

Image size is a fundamental problem when it comes to running on restricted resources. Keep in mind that VisualAge uses substantial amounts of DLL code— most probably more than the image size.

Anyway the packager will do its best to get rid of any superfluous classes and methods, but you can do a few things that might be helpful:

- Have a small image to run packaging from, containing only the absolute minimum classes and methods (no other applications, no samples, no unnecessary infrastructural support).
- Do not use ″standard″ method names, as the packager seems to determine class and method exclusion by name, disregarding the class context for method names. For example, calling a packaged and needed method *doSomething* will package every method called *doSomething* that is found in your entire development image, which will include every class.
- Get the prerequisites right; having unnecessary or wrong prerequisites can be fatal, possibly resulting in an 8 MB package instead of the normal 2 MB.

Including Classes in a Package

Question

We create a complex object outside of our application and dump the object to disk using the ObjectDumper class. This dumped object contains a collection of Catalog objects, and each Catalog object contains a collection of Part objects and collections of Vehicle objects. After dumping this object to disk, the object remains static, so our application can use it.

I discovered that when we packaged our application, objects such as Catalog, Part and Vehicle are not included in the package. To get around this, I am creating an instance of these classes, and finally letting them be collected as garbage. Not doing this results in an error after using the ObjectLoader, saying the ″Class definition was missing.″

To me this looks like a dirty solution, is there a better way to force classes to be included in the package?

Answer Implementing the method *packagerIncludeClasses* (on your applications) to answer a collection of classes that you want to force the packager to package will solve this problem. Browse implementors of *packagerIncludeClasses* for more details.

Packaging for AIX from OS/2

Question We are using VisualAge for Smalltalk for OS/2 V3.0. We need to distribute both OS/2 and AIX clients of the application. Can we package the AIX version of the client from our OS/2 development environment? Or do we need to export from OS/2 and then import into VisualAge for AIX, and then package the AIX version from VisualAge for AIX.

Answer You must package in the target environment. By the way, to access a single manager on an AIX machine, you can use TCP/IP and the EMSRV support for all environments. If you do that, no import/export is necessary, just a load and package.

Loading Application in a Run-time Image

Question Is it possible to load .APP files into a runtime image, not a stock image, if the ApplicationLoader and Swapper are included in it? It seems that the reduced runtime image (AbtEpStandardRuntimeSpecification) that comes with VisualAge includes both.

Answer The reduced runtime image does not have the ApplicationLoader and Swapper applications that you will need to bind .APP files to it. The stock image does.

Why did it seem that the reduced run-time image had these applications? It is likely that you reviewed the applications in the Packaged Image Browser associated with the reduced run-time image specification. If this is true, you should note that these applications were not *base* applications, and as such, may not be completely added to the run-time image.

Packaging an Application for Windows

Question Is it possible to package an application that should run under Windows with VisualAge under OS/2?

Answer You cannot run the OS/2-packaged application under Windows regardless of whether that′s native Windows or a Win-OS/2 session. In order to package your application for execution on a Windows platform, you must have a copy of VisualAge for Windows from which to create the run-time.

Icons and DLLs

Tip The VisualAge run-time does not use ABTICONS.DLL for any of its resources. In fact, the edittime doesn′t either. Only the SampleParts and VisualAge Examples Help use the resources contained in this DLL. The DLL is provided simply to provide some common images for you to use in your application. Now, when shipping your packaged application you have to include the .BMP files and not the ABTICONS.DLL.

Packaging a Workspace

Question Is it possible to package an EtWorkspace with an application?

Answer You cannot package an EtWorkspace because it is part of the edit-time development environment (see the \visualag\install\visualag.api file for a list of redistributable classes and methods); this is partly because those workspaces have menu options for evaluating code using the compiler, which is definitely not packageable. If you are looking to simply reproduce an editor window, just drop a multiline edit part in a window. .

Image Reduction

Question How do I strip down my development image?

Answer The VisualAge for Smalltalk packager is functionally equivalent to the third party and OEM *strippers* that are available for other Smalltalks. They all analyze your image, and remove unneeded classes and methods. The major difference is that some of the better third party strippers can dynamically analyze the running system, where the VisualAge packager does only a static analysis. Even so, the packager does a pretty good job of getting the image size down.

Ensuring that Needed Classes are Packaged

Question How can I make sure that all classes and methods I need for an application are packaged, given that the classes are spread over various applications? Would it be sufficient to include two private methods: *packagerIncludeSelectors* and *packagerIncludeClassNames*?

Answer The packager excludes all applications that are not part of the prerequisite list, even if you implement one of the methods (*packagerIncludeSelectors*, *packagerIncludeClassNames*). By using one of these methods, you are telling the packager to include the selector, class, and so on from the application no matter whether the packager thinks it should be included or not. However, you still need to tell the packager what application to draw from.

Class Modifications and Packaging

Question Suppose you extend a class, such that you need to create a new edition of the application to which that class belongs, will there be anything special you need to do when you package your application?

Answer No, if you have your changes loaded in the image when you package, then the changes will be picked up by the packager. For what it is worth, if you extend a class you only add methods and do not directly modify the existing code so you should not have to modify the owning application. You can create a new extention of that class in your own application by selecting **New** → **Extension** from the **Parts** menu in the Organizer.

Merging Application Files into Run-time Image

Question The VisualAge 3.0 on-line manuals state that it is possible to merge APP files into run-time images, even if the image is not the stock image, but a reduced one. How do I enable the reduced image to load the APP files?

Answer A stock image is no longer shipped since Version 3. You can either generate it using the Packaged Images Browser and appropriate packaging specification (AbtEaRuntimeSpecification) and use the ABTAPP.CNF approach as with Version 2, or you must add the appropriate logic within your own application which supports this type of behavior (that is, uses ApplicationLoader to bind .APP files under certain conditions). If you add the logic yourself, it is your responsibility to ensure that the reduced run-time image contains all the necessary methods to support the methods required by your application modules (.APP files).

Missing Icons

Question Why do I have missing icons after packaging the application?

Answer VisualAge caches the window handles of the icons used in the message prompter. The cache is cleared when the image is saved. If you package your application before saving your image, the current window handles of the icons get packaged into the run-time image. When you run the packaged image, these window handles are no longer valid and, as a result, you get missing icons or a trap. A quick fix to the problem is to save your image and then package your application (don′t test it after saving the image). A more appropriate solution is to implement a private class method in AbtPromptersAppWithUI called *packagingRulesFor:* which will clear the cache prior to packaging. Underneath you find the code for this method:

Smalltalk

```
packagingRulesFor: aPackagedImage
″Define rules for the given packaged image.″
aPackagedImage initializeToNilClassVariable: ′AbtIconPixmaps′
  inClass: AbtExtendedMessagePrompter.
```

Required .CAT Files

Question How do I find out what .CAT files are required for my run-time image?

Answer There′s a little script to scan the messages during packaging and show you catalogs your code referenced. That′s the minimum set of *.cat files you need to distribute. By the way, most nonerror items are bound to the image using the CAT entries from the development environment (English). These files are needed if the locale changes because it is on a non-US machine or a run-time error occurs. See EsPoolDictionary>>indexedMsg: for details.

Smalltalk

```
| stream tokens file |

file := (CwFileSelectionPrompter new)
    title: ′Packaging Messages′; prompt.
(file isNil) ifTrue: [ ^nil ].
stream := CfsReadFileStream
 open: file.
[stream atEnd] whileFalse: [
  tokens := stream nextLine subStrings.
  ((tokens size > 0) and: [((tokens at: 1)
    indexOfSubCollection: ′NlsCat′ startingAt: 1) > 0]) ifTrue: [
        Transcript cr; show: ′References messages in: ′,
          ((Smalltalk at: (tokens at: 1) asGlobalKey)
         at: ′CATALOGNAME′)]].
stream close.
```

Packaging Pool Dictionaries

Question How can I tell the packager not to change my pool dictionaries when creating a run-time image?

Answer To prevent the packager from changing your pool dictionaries, add the following class method to your class application:

Smalltalk

```
packagingRulesFor: aPackagedImage
″Pack the NLS pool″

aPackagedImage includeGlobal: #MyClassPool.
```

Question I′m having difficulty getting the simple packaging process to include all keys and values of two application specific pool dictionaries that are used in one of the classes in my application. The packager seems to be excluding all keys/values in these pools unless I code a rather ugly method which explicitly refers to all keys/values in these two pools. What do I do?

Answer

Are you referring to your pool dictionary entries by name (for example, are you using #at: to look them up)? If you are, you′ll need to inform the packager not to reduce the pool. By default, the packager will remove any entries that are not directly referenced. To prevent this, add a packaging rules class method to your application class like the one below. The packager will not do pool reduction if a rule has been explicitly set.

Smalltalk

```
aPackagedImage includeGlobal: #MyPoolDictionary.
```

Application Prerequisites

Tip

Of the problems we have encountered with packaging, 90% have been problems with prerequistes. With the team version of VisualAge, it is possible to partition your deliverable across multiple applications.

Each application has certain requirements that must be met for an image to make sense. For example, if an application introduces a new class, the superclass must be available, either in the same application or in a prerequisite application. It can be said that one application needs others to be in memory to work.

It is also worth mentioning that when an application is listed as a prerequisite for another, no mention of the appropriate version is made. Before packaging, ensure that the appropriate versions of all applications have been loaded. The packager uses the prerequisites when building a runtime image.

Missing Classes in the Run-time Image

Tip

The packager starts with the collection of all classes in the application and all prerequisite applications. From that starting point, it extracts all classes and methods which are not explicitly referenced in code.

If you use any metaclass tricks, like asking a class for its subclasses, those subclasses would not be explicitly referenced, and would therefore be excluded.

The packager can be coerced into including certain classes by adding a new class method to the class with the same name as your application.

Smalltalk

```
packagerIncludeClassNames: aString

    ″Answers a collection containing the names of the
     classes which must be included when packaging.″

    #(MyFirstClass MySecondClass)
```

The class with the same name as your application can hold behavior that is specific to your application (in its class methods). This class is a subclass of the class called Application, which is itself a subclass of SubApplication. In the class SubApplication, you find similar methods that force specific methods or classes to be included, or excluded when packaging.

Missing Method in Package

Question I try to package a reduced run-time image. After the packaging process has finished it seems that the package is missing the method *convertToCodePage:* of EsString even though it is referenced several times. Is there a way to force a method to be packaged?

Answer You can force the method to be included by the packager by adding a class method called *includedMethods* to the class with the same name as your application. This method returns an array of methods to be included in the packaged image. An example of this method is listed below:

Smalltalk

```
includedMethods
″Return a list of methods to be included in the packaged image″

^Array with: ToDoListView >> #test
```

Run-time Image Packaging Problem

Tip

Messages about OS2.INI not being able to be saved have usually been the result of insufficient disk space (which would also lead to slow execution since the swapper cannot grow), so check that first. Another cause of slow execution is not having ABTBYTES.DLL in the LIBPATH (or PATH under Windows).

As for the run-time image itself, more than 5 MB is way too large (unless you have a truly huge application). This is typically caused by having the wrong prerequisites for your application. A common example is when your application requires AbtBuildViewsApp (part of the application builder development environment). This usually happens if you generate archival code into your application rather than having a separate archival application (which is strongly recommended). If this is the case, you should create the separate archival application, move all archival code into that application, and update your prerequisites so that they do not include any of the development-time applications.

To determine which methods are archival, just regenerate again and note the messages in the Transcript. An API document is shipped in softcopy form with the product, which details what applications are considered part of the runtime and thus redistributable.

Generating Getters and Setters

Tip

You can use the TrailBlazer browser to generate getters and setters for an application. Select a class, navigate to instance variables (or class variables), select a variable, pop up the menu, and select **Generate Accessors**.

High Performance Trace Logging

Tip

For complex systems, often trace logging must be built in. Unfortunately, you can incur a high performance penalty if you do not build in trace logging properly. One way of avoiding this penalty is to send blocks rather than strings to your logging subsystem. For example:

Smalltalk

```
  self traceShow: ″I am entering method #blah for ″, self printString,
     ″ with: ″, myArg printString.
```

Using a string can be expensive because you have to build the entire string even if tracing is not active. Therefore use a block:

Smalltalk

```
  self traceShowBlock: [″I am entering method #blah for ″, self
printString,
     ″ with: ″, myArg printString.].
```

Then in the trace block code, do this:

Smalltalk

```
  traceShowBlock: aBlock

    self isTracing ifTrue: [
       traceLog cr ^ show: aBlock value ].
```

In this way, you can leave your trace code in the system without paying a steep performance penalty. The string is only built when tracing is active.

Closing a DLL from Smalltalk

Tip

If you are developing a Smalltalk application that calls C or COBOL functions in a DLL, when you initially call the DLL function, Smalltalk opens the DLL and leaves it open. If you need to change the DLL code, the DLL must be closed. To avoid having to shut down Smalltalk, execute the following:

Smalltalk

```
    (PlatformLibrary logicalName: ′myDllName′) close″
```

Unloading When Instances Exist

Tip

To unload an application when instances exist, execute the following:

Smalltalk

```
MyClass basicAllInstances do: [ :each |
  each become: Object new ].
```

Using *basicAllInstances* is faster than using *allInstances* because a garbage collection is not performed.

Testing When Looping Walkbacks Occur

Tip

When testing while a looping walkback occurs (such as in the midst of a losingFocus callback), use the following:

Smalltalk

```
GlobalIsTesting ifTrue: [
  GlobalIsTesting := false.
  self halt. ].
```

Then execute this before each test:

Smalltalk

```
Smalltalk at: #GlobalIsTesting put: true.
```

A walkback will occur only the first time the halt is encountered rather than every time.

Quickly Disabling Methods

Tip

To quickly make a method do nothing, add this line:

Smalltalk

```
true ifTrue: [ ^nil ].
```

To quickly make a method forward its behavior to its superclass use this line:

Smalltalk

```
true ifTrue: [ ^super myMethodName ].
```

In this way you do not have to delete or significantly modify a method when you want to change its behavior or bypass an unfinished portion.

Specifying Public or Private with TrailBlazer

Tip

To quickly specify the public or private policy for a class, select the class and in the lower area (the properties area) select the instance interface (or class interface). You can then easily make public or private assignments.

Browse and Find in the Pop-Up Menu

Tip

Here is a little tool that extends the development environment. The tool adds two extra items to the pop-up menu: *Browse* and *Find.*

Browse is an intelligent browse; you select the code in the pane and then make your choice. The code you have marked is evaluated, and the resulting class is determined. If the class is a part, the *editPart* message is sent and the VisualAge for Smalltalk editor is opened. If the class is not a part, the ordinary Smalltalk browser is opened.

Find brings the marked text into the Find window (of VisualAge for Smalltalk) and opens it. If the Organizer is not minimized, the resulting class (if you search for a class) is marked.

Here is the code for filing into your image. When the application NhaTool is loaded, you have the extension.

Smalltalk

```
Application create: #NhaTools with:
   (#( AbtViewApplication EtTools)
      collect: [:each | Smalltalk at: each ifAbsent: [
         Application errorPrerequisite: #NhaTools missing: each]])!

NhaTools becomeDefault!
AbtViewApplicationAbtPackage subclass: #NhaToolsAbtPackage
   instanceVariableNames: ′′
   classVariableNames: ′′
   poolDictionaries: ′′!

NhaTools becomeDefault!
Application subclass: #NhaTools
   instanceVariableNames: ′′
   classVariableNames: ′′
   poolDictionaries: ′′!

NhaTools becomeDefault!

!EtBrowser publicMethods !

defaultTextMenu
  ″Answer the modified textMenu″

^super defaultTextMenu
     addLine;
     add: #menuBrowse      label: ′Browse′;
     add: #menuFind        label: ′Find′;
     yourself! !

EtDebugger description: ′Developer:  Niclas Havrup
Date:  96-06-07
APAR:

Description:
Menu items add to the debugger′ in: NhaTools!
```

Smalltalk (continued)

```
!EtDebugger privateMethods !

defaultTextMenu
  ″Answer the modified textMenu″

^super defaultTextMenu
    addLine;
    add: #menuBrowse    label: ′Browse′;
    add: #menuFind     label: ′Find′;
    yourself! !

!EtInspector publicMethods !

defaultTextMenu
  ″Answer the modified textMenu″

^super defaultTextMenu
    addLine;
    add: #menuBrowse    label: ′Browse′;
    add: #menuFind     label: ′Find′;
    yourself! !


!EtWindow publicMethods !

menuBrowse
  ″Browse the class for the selection″

  | selection className |
  selection := self targetTextWidget getSelection trimBlanks.
  className  := (Smalltalk  classAt: selection
      ifAbsent: [(self evaluateSelectionIn: targetTextWidget
       ifFail: [ ^self ])
       class.]).

  (className inheritsFrom: AbtPart)
    ifTrue: [ className editPart.]
    ifFalse: [ ((EtTools browser: #class) on: className) open ].
!
```

Smalltalk (continued)

```
menuFind
  ″The Find window″

  | selection className |
  selection := self targetTextWidget getSelection.
  selection isNil
    ifFalse: [selection := selection trimBlanks].

(AbtApplicationsOrganizerView current find
    secondaryViews at: #find ifAbsent: [nil]) findText: selection.

! !

!EtWorkspace publicMethods !

defaultTextMenu
  ″Answer the modified textMenu″

^super defaultTextMenu
    addLine;
    add: #menuBrowse     label: ′Browse′;
    add: #menuFind       label: ′Find′;
    yourself! !

!NhaTools class privateMethods !

runtimeStartUp

  self abtViewApplicationPackage runtimeStartUp! !

NhaToolsAbtPackage initializeAfterLoad!
NhaTools initializeAfterLoad!

NhaTools loaded!
```

Versioning Applications Not Owned

Tip

If you make changes to a class that is in an application you do not own, it can be difficult to version and release the application.

Here′s the incantation (which assumes that you do not have password control of user IDs or know the passwords):

1. Open the Application Manager.
2. Select the troublesome application.
3. Select the open editions of the class or classes. The owner of that change will be highlighted in the user list in the Application Manager.
4. Double-click on the highlighted name to change the user to that person.
5. Click the right mouse button over the class list and select **Version** → **Name each**, **One name**, or **Use defaults**. This will version the classes. Now you need to release them.
6. Double-click on the name list to change the user to the manager of the application (name with the > next to it).
7. Click the right mouse button over the class list and select **Version/Release All** →, select **Name each**, **One name**, or **Use defaults.** This will version and release all the classes, some of which may already have been versioned. Another way to do this is to just select the classes that have not been released yet (marked with a >), click with the right mouse button to open the menu, and select **Release.**
8. When all classes in the application are versioned and released, change the user to the the manager and version the application.

Client Not Supported

Tip

If your connectivity is through Microsoft Windows for Workgroups (3.11) MSDLC protocol and Client Access/400 V3R1 router, be aware that the CA/400 DOS Extended client is not supported for use with either MSDLC or VisualAge AS/400 Connection (see AS/400 Connection User′s Guide SC34-4544-0 under Chap.1, ″Getting Started...″ section on ″Prerequisites for Windows″) In this case, you must use the Client Access/400 for Windows 3.1 client.

AbtRecord

Question How do I use AbtRecords?

Answer There are some references to AbtRecord in the online *Communications/Transactions Guide and Reference*. One way to create an AbtRecord instance is by sending the message #newRecord to an AbtCompoundType. Here is an example I found:

Smalltalk

```
| ctype rec |
ctype := ( AbtCompoundType new
  addField: (AbtCOBOLDisplayField new name: ′string1′; length: 8 ) ;
  addField: (AbtCOBOLDisplayField new name: ′string2′; length: 8 );
  yourself ).
rec := ctype newRecord.
rec at: ′string1′ put: ′abcde′.
rec at: ′string2 ′ put: ′ddddd′.
Transcript cr ; show: (rec at: ′string1′) ;
  cr; show: (rec at: ′string2′).
```

Indexed Message Editor

Tip I find that, whenever I made changes in the Indexed Message Editor, I need to

1. Save the changes.
2. Execute application *abtGenerateExternalizedStringsIfNecessary.*
3. Execute *AbtNLSCoordinator forceRelocalizationOfSeparatedConstants.*

After I do this, all my messages are immediately available. I never have to unload or reload the application.

Corrupted MRI File

Question The messages in my MRI user message file appear corrupted or have bad data. What can be wrong?

Answer The MRI file in VisualAge for Smalltalk, Version 1.0 has a limit of 64 KB, so messages above this limit may appear corrupted. To get past the limit, in application AbtNLSSubApp, change class AbtMRIGroupHeader, method offsetSize as follows:

Smalltalk

```
offsetSize
 ″Public - answer size of file offset used in header.

 ^offsetSize isNil
  ifTrue: [ offsetSize := 4 ]
  ifFalse: [ offsetSize ]
```

The only change in this method is to change the number 2 to 4. For VisualAge for Smalltalk, Version 2.0, the changes have already been made. In the method above, the offsetSize now defaults to 4 instead of having to change it to 2. So, corruption should no longer be a problem in Version 2.0

Class Naming Convention

Question What naming convention should I use?

Answer Our naming convention complements existing literature[1] on the subject to facilitate code reuse and to provide a layered architecture for the development environment.

A naming convention in your development environment greatly helps reusing existing classes because it allows searching by class name based on a filter.

[1]

- *VisualAge for Smalltalk, Programmer′s Guide to Building Parts for Fun and Profit*, IBM
- *Smalltalk with Style*, David Thomas et al, Prentice Hall
- *Smalltalk Best Practice Patterns*, Kent Beck, Prentice Hall

Class Name Prefix

The prefix is arbitrarily defined with the following three characters:

1. The first letter stands for the company name.
2. The second letter describes the type of components:

   A Application

   C Class

   M Configuration map

3. The third letter is used to categorize the component within a layered architecture:

   A Defines classes of the top layer, which is the application layer. All classes are visual, representing the screens of the application. At the application layer, the primary part of a class is always a window (for example, an instance of AbtShellView) rather than a form (an instance of AbtFormView).

   F This optional layer contains reusable visual classes. The main difference between this layer and the application layer is the fact that the primary part of a class is always a form rather than a window.

   M The model layer contains the implementation of the object model reflecting the business domain of the enterprise.

   G As part of the infrastructure, the GUI layer contains classes that provide additional features to the base product, such as converters, security defined at the widget level, controller services common to any instances of Model-View-Controler (MVC), or a window manager that keeps track of the view that owns the current unit of work.

   E As part of the infrastructure, this layer contains class extensions made to the base product.

   T As part of the infrastructure, this layer contains in-house development tools for developers. These classes are never packaged with the end-user application.

   S As part of the infrastructure, this layer contains common services defined in the Common Object Request Broker Architecture (CORBA), for example, the persistence object service, concurrency, and transaction service.

Class Name Suffix

For a class, the category of component is defined with the third letter of its prefix and the use of a suffix.

Model Class

Model classes (for example, business domain classes) do not need a suffix.

Reusable Visual Classes

Reusable visual classes are derived from model classes. There is a one-to-many relationship between one model class and its reusable visual classes, and that is why we always use the name of the model class as the name of the reusable visual class. Its suffix consists of two parts:

- The first part characterizes the information being displayed. Use a name that characterizes the reusable visual class based on the attributes displayed from its model class. Consider using terms like the following:
  - Summary
  - Detail
- The second portion is the word *View*.

Visual Classes

Visual classes are task oriented and should reflect in their name their main task. Use *View* for suffix.

Facade Classes

Structuring a system into subsystems helps reduce complexity. A common design goal is to minimize the communication and dependencies among subsystems. The Facade class provides a single and simplified interface to the more general facilities of a subsystem[2]. Use *Facade* for suffix.

Controller Classes

Controller classes handle the sequence of events found in a use case. They deal with user interface logic or business processes. Jacobson defines them as control objects[3]. Use *Controller* as suffix.

Agent Classes

Agent classes are used to encapsulate the physical implementation of a portion of or whole service by using one or more service objects. One agent class plays an intermediate role between one or more model classes, which

[2] *Design Patterns*, Erich Gamma et al, Addison-Wesley

[3] *Object-Oriented Software Engineering*, Ivar Jacobson, Addison-Wesley

do not know anything about the outside world of a Smalltalk run-time image, and the service object, which in turn does not know anything about the business domain[4].

Use the word *Agent* followed by the kind of service object used. Here are some examples:

- AgentDB2: Classes implementing service, using DB2/2 service objects
- AgentMQ: Classes implementing service, using MQSeries service objects
- AgentTCPIP: Classes implementing service, using TCP/IP service objects

Manager Classes

Manager classes keep track of all instances of a specific class or a subset of a class. They are also used to select some instances of one specific class based on some criteria[5]. Use *Manager* as suffix.

Category Naming

How should I group methods within a class in order to improve readability?

Category names have to be explicitly created. We propose category names for instance and class methods.

Question

Answer

Categories for Instance Methods

- Get and set selectors

  Contains all accessor methods for instance variables

- Extensions

  Used when an extension is made to an existing class of the development tool

- Initializations

  Contains initialization methods

- User interface logic

  Used for validation or navigation logic

[4] *VisualAge and Transaction Processing in a Client/Server Environment*, Andreas Bitterer et al, Prentice Hall PTR

[5] *Enhancing the Object Design Process Using Stereotypes*, paper presented by Rebecca Wirfs-Brock at the Smalltalk User′s Conference in New York, 1996

- *ResponsibilityName*

  A category is created for each responsibility (for example, contract) taken by the object.

- Services

  Contains methods having a general purpose such as boolean or copy methods

We do not recommend defining a method to more than one category even if the tool allows it.

Categories for Class Methods

Some categories for class methods are the same as those described for instance methods.

- Get and set selectors

  Contains all accessor methods for class variables

- Creations

  Contains methods returning one instance of the class

- Constants

  Contains constant methods

- Example

  Contains methods explaining how to use the class. A common usage is showing how to create an instance of the receiver.

- Extensions

  Used when an extension is made to an existing class of the development tool.

- Initializations

  Contains initialization methods

- *ResponsibilityName*

  A category name is created for each responsibility (for example, contract) taken by the object.

- Services

  Contains methods having a general purpose such as boolean or copy methods.

- Tests

  Contains unit testing methods

Archival Code is Missing

Question I had an application on VisualAge for Smalltalk on OS/2 2.11. Now I installed VisualAge for Smalltalk on a Warp system and filed in my application and the archive file. I get all parts, but the view wrapper parts are empty (the Composition Editor shows only an empty window). Has anyone an idea what could be wrong?

Answer The problem you are experiencing (for example, no visual parts in any of the windows) is usually an indication that the archival code is missing. Open a Composition Editor on one of your visual parts and then switch to the Script Editor. Check and see if you have any class methods. If you generated and then filed in your archival code, you should see the following class public methods:

- *calculateInterfaceSpec*
- *calculatePartBuilder*

You should also have some class private builder methods. Do you see any of these methods? Open a browser on the archival application. Do you see class names for all your visual parts? When you generate archival code for your visual parts, the class gets extended in the archival application. The class extensions are the archival code methods.

If you don′t have the archival code methods, I suggest you go back and regenerate the archival code, file it out, and then file it back in.

Meaning of Application Errors

Question Where or how can I find an explanation of the fatal application errors?

Answer From the System Transcript, display the following code:

Smalltalk

```
SystemPrimitiveErrors keyAtValue: errorNumber
```

If, for example, you highlight and display this statement with argument: 33, you get the following result: ′PrimErrImageFileOpenError′.

Error: ′Undefined Object does not understand messageString′

Question I get the following error every time I try to save a part: ′Undefined Object does not understand messageString′ How can I correct this?

Answer Inspect AbtShellView (the class). You can do this by executing

Smalltalk

```
(AbtShellView inspect).
```

From the inspector, double click on the entry ′classPool′ This should open a Dictionary Inspector with two entries: ′AbtInImageRestore′ and ′InLongOperation′. Select the entry called ′InLongOperation′.

If it says *true* then change it to *false* and save (using the pop-up menu) the change.

If *false* appears in the right pane, then you have a different problem.

Error on Saving a Part

Question When I try to save a part, I get an error that states that there is no primary part, why?

Answer This can happen assuming the following order of actions:

1. Open a new part (the default window is the primary part)
2. Add, for example, a notebook (the default window is still the primary part).
3. Delete the window (no primary part)

From here, an attempt to save will generate a warning until you specify which part is to be the primary one.

However, if the order is the following, the result is different:

1. Open a new part (the default window is the primary part).
2. Delete the window (no primary part).
3. Add the notebook (the notebook becomes the primary part).

From here, saves works fine.

In summary, if the primary part is deleted, VisualAge makes no assumptions about which part should now become the primary one. If, however, you add a new part when no primary part is currently defined, the newly added part becomes the primary part. It must be a visual part when editing a view class, otherwise the primary part will stay undefined. The rationale for automatically setting a primary part when one is not currently set is to simplify the typical scenario where one wants to substitute the window for an embeddable window (Form).

Inappropriate VisualAge Icon

Question In Windows, I click on the VisualAge icon and nothing happens or my VisualAge icon has DOS in it instead of the eye, or I receive ″cannot run in DOS mode.″ What should I do?

Answer You have probably not installed Win32s support needed for Windows or do not have the correct version of Win32s. VisualAge for Smalltalk requires level 1.15.111.0. You can tell what version you have by looking at the Win32s.ini file in the \windows\system subdirectory. Refer to the readme.txt file in the \wradcdw1\vaprodn1\readme subdirectory on the VisualAge for Smalltalk CD and search for Win32s instructions on installing Win32s support.

Double Execution of Script

Answer Why does my script get executed twice when a user hits the enter key and I use the *defaultActionRequested* event?

Answer This is a known problem. You′ll need to change the method AbtTextView>>#postCreationInitialization. Locate the following piece of code in the method and remove it:

Smalltalk

```
widget addCallback: XmNactivateCallback
  receiver: self
  selector: #defaultAction:clientData:callData:
  clientData: nil.
```

Error 203

Question When trying to start VisualAge for Windows, I get Error 203. Why?

Answer Older versions of VisualAge for Smalltalk did not run on Windows 95 or NT. You need to upgrade to at least version 3.0a if you want to run VisualAge for Smalltalk under Windows 95 or Windows NT.

Removing Elements from a View

Question How do I remove elements from a tree view?

Answer We opened APAR PN82758 on a problem that sounds very similar to this problem. Here is a workaround or fix: In EwHierarchy>#refreshItems:childrenBlock: Change the lines that read:

Smalltalk

```
oldExpandedItems add: (self items at: index).
oldExpandedNodes add: node!.
```

To the following:

Smalltalk

```
(index between: 1 and: self items size)
ifTrue:
   oldExpandedItems add: (self items at: index).
   oldExpandedNodes add: node!!.
```

Class EmLibraryStatistics

Question Where is the class EmLibraryStatistics that was in IBM Library Tools configuration map?

Answer It was left out of the Version 3.0a refresh manager. You can import the configuration map from your Version 3.0 manager into your Version 3.0a manager. The code hasn′t changed between the two.

Error: rc=12

Question What does a rc=12 for abt.exe mean?

Answer Try executing the following code:

Smalltalk

```
Object primitiveErrorStringFor: 12     → "Not enough memory"
```

Error: SYS317x

Question Can I avoid SYS317x in ESVM30.DLL, PMMERGE.DLL, or xxx.DLL on OS/2?

Answer OTI believes that many of the SYS317x messages our customers are seeing are due to stack overflow. When the stack overflows, some reasonably important data is trashed in ABT.EXE. The effect of trashing this data can be immediate or delayed (depending on how much is trashed and what the trash looks like). Also, according to OTI, there is no way for customers to either protect against a stack overflow or know when one has occurred. Many of our customers run quite successfully with the original stack size of 32 KB. However, as VisualAge for Smalltalk applications have become more complex, stack usage has increased. Therefore, before the Version 3.0a refresh, ABT.EXE was rebuilt with a 64 KB stack.

It has been noted that a 64 KB stack is not always enough either. So, I have developed a version of ABT.EXE that runs Smalltalk on a thread. This allows customers to set any stack size they need. It also keeps Smalltalk-called DLLs from writing past the end of the stack (based on the different mechanism used for managing thread stacks vs. the main thread stack).

When you get a report of a SYS317x on IBM Smalltalk or VisualAge for Smalltalk that occurred while running the product (not during install, for example), here are some steps to diagnose and correct the problem:

1. Does the trap occur when the user is doing lots of user interface interaction? If so, suggest that users update their video drivers to the newest level and retry. In the past, we′ve seen this kind of problem on Windows, but it is also beginning to show up on OS/2. It seems to be most prevalent with S3-based video cards, but has been seen with others. Diamond Stealth drivers seem to be particularly notorious for causing this problem.

2. Is the user still on Version 3.0? If so, suggest that they upgrade to Version 3.0a. In the meantime, you could send them a replacement for ABT.EXE to bring it up to the Version 3.0a level.
3. Does the trap occur when bring up a Version 3.0a image with database features loaded? Have the customers saved their image with a database connection open? If so, suggest that they reload into a clean image and not save with open database connections. There is a problem in database with finalization code that is trying to free operating system objects that have bogus memory pointers at startup.

If these don′t help, and if the SYS317x occurs based on a repeatable scenario, then we can turn the problem over to OTI support (after verifying that it can be recreated) if the customer supplies: a) a scenario for recreating the failure, b) the failing image, and c) any associated files, DLLs, and the like needed to recreate the failure.

Error: Primitive Error Codes 52 through 55

Tip

Primitive error codes 52 through 55 indicate a general protection fault or trap. They are described in the table at the end of Chapter 11 in the *IBM Smalltalk Programmer′s Reference*. These error codes were introduced in Version 3.0a. The specific meanings are:

52 - General Protection Fault
53 - General Protection Fault - read from invalid memory location
54 - General Protection Fault - write to invalid memory location
55 - General Protection Fault - invalid instruction executed

If the primitive error is associated with a *callWith: type* method, then the general production fault occurred in the DLL being called (or in a function that it called). If the primitive error is associated with a *xxxAt:* or *xxxAt:put:* type method, then an invalid address was supplied to the method.

Error: Does Not Understand Format

Question

After filing in code from Version 2 Standard to Version 3 Team, I get a debugger with ″AbtIbmDate384Field does not understand format:″ when generating runtime code for an access set.

Answer Implement a *format:* instance method for AbtIbmDate383Field that does nothing.

Menu on Windows NT 4.0

Question VisualAge for Smalltalk Version 3.0a for Windows has a small menu problem when run on Windows NT 4.0. The context menu for text fields appears after a default windows menu. What do I do?

Answer This behavior can be easily fixed by executing the following code:

Smalltalk

```
OSWidget eventTableAt: 16r7B ″WmContextmenu″
  put: #wmContextmenu:with:
```

Back-tabbing in Container Details Views

Tip On some UNIX platforms (HP-UX and Hummingbird Exceed X server, for example) shift-Tab is mapped to key symbols other than back tab. The container details view expects shift-Tab to back tab (or tab to the left). Therefore, when running VisualAge on these platforms, you may need the following workaround in order to use Shift-Tab to back tab through container cells. Edit the key mappings for your X server. If you are using X server software on a Windows or OS/2 workstation, the X server software may provide dialogs to edit the keyboard mappings. Use these dialogs to change the setting for ″Shift″+″Tab″ to generate the keysym ″Tab.″ For X servers running under UNIX, you can run X command *xmodmap* to change the mappings. For example, to correct the mapping on an HP-UX workstation, run the following command:

```
xmodmap -e ″keysym Tab = Tab Tab″
```

You can run this command from your .profile or other startup or logon script, but it must be run after your display is specified. If you are already running xmodmap, you may have a resource file, possibly called *.xmodmaprc,* to which you can add the above keysym expression.

Properties Views and Notebook Settings Views

Tip

To switch from Properties views to Notebook settings views, do the following:

1. From either the System Transcript or the Organizer, bring up the Load/Unload dialog.
2. Choose to load VisualAge Notebook Style Settings Views.
3. In the VisualAge Organizer Preferences, make sure Notebook Style is selected for the Preferred Settings View.

Code Page Conversion Errors

Tip

If you are receiving errors related to code page conversion, select **Tools** → **NLS** → **Rebind Image Strings** from your Transcript.

Determining Required ICs

Tip

The following script can be run to determine the ICs that are needed at run time. It will show a list of all IC packaging instructions that are currently loaded. When you select a packaging instruction, the required run-time ICs will be listed on the Transcript.

Smalltalk

```
| icCollection instructionNames instructionName instructionClass work |

 instructionNames := (EpAbstractPackagingInstructions
   withAllSubclasses collect:
    [ :ea | ea name asString])
       asSortedCollection asArray.
 instructionName := CwListPrompter new
       title: ′Find IC files required by the selected IC′;
       items: instructionNames;
       prompt.
 instructionName size = 0 ifTrue:[ ^self ].
 instructionClass := Smalltalk classAt: instructionName
    ifAbsent: [^self].
 icCollection := Set new.
 icCollection add: instructionClass dumperOptions imageFileName.
 instructionClass prerequisiteICs do: [:eachICInstructionClass|
              icCollection add: (eachICInstructionClass asClass)
                 dumperOptions imageFileName].
 work := Transcript bringToFront.
       work cr;
       nextPutAll: ′******* The required ic files for ′,
          instructionClass name,′ *******′;
       cr;
       cr.
 icCollection isEmpty
       ifTrue: [work tab; nextPutAll: ′No IC files needed.′; cr]
       ifFalse: [icCollection asSortedCollection do: [:each| work tab;
          nextPutAll: each; cr]].
```

Required CAT and MPR Files for an IC Run-time Image

Tip The following script can be run to determine the .cat and .mpr message files that are needed by an IC image at run time. It will show a list of all IC packaging instructions that are currently loaded. When you select a packaging instruction, the required run-time message files will be listed on the Transcript.

Smalltalk

```
| appCollection catFiles mprFiles instructionNames
  instructionName instructionClass work |

instructionNames := (EpAbstractPackagingInstructions
  withAllSubclasses collect:
  [ :ea | ea name asString])
      asSortedCollection asArray.
instructionName := CwListPrompter new
      title: ′Find .cat and .mpr files required by the selected IC′;
      items: instructionNames;
      prompt.
instructionName size = 0 ifTrue:[ ^self ].
instructionClass := Smalltalk classAt: instructionName
  ifAbsent: [^self].
appCollection := Set new.
catFiles := Set new.
mprFiles := Set new.
instructionClass actualIncludedSubApplicationNames do: [:eachApp|
    appCollection add: (eachApp asClass rootApplication)].
instructionClass prerequisiteICs do: [:eachICInstructionClass|
    (eachICInstructionClass asClass)
      actualIncludedSubApplicationNames do:
        [:eachApp|
           appCollection add: (eachApp asClass rootApplication)]].
appCollection do: [:eachApp | |poolNames mprName|
      poolNames := eachApp definedPoolNames.
      poolNames do: [ :eachPoolName| |pool catName|
          (pool := (Smalltalk at: eachPoolName ifAbsent:[]))
            epIsPoolDictionary
                 ifTrue:[(catName := (pool at: ′!CATALOGNAME′
                   ifAbsent: [])) isNil
                 ifFalse: [catFiles add: (catName,′.cat′)]]]].
      (mprName := eachApp abtNlsRawFilename) isEmpty
            ifFalse: [mprFiles add:
              (AbtNLSCoordinator resolveLanguageMappingCharacter:
                 mprName)]].
```

Smalltalk (continued)

```
work := Transcript bringToFront.
work cr;
      nextPutAll: ′******* The required message files for ′,
         instructionClass name,′ *******′;
      cr;
      cr;
      nextPutAll: ′Cat files:′;
      cr.
catFiles isEmpty
      ifTrue: [work tab; nextPutAll: ′No .cat files needed′; cr]
      ifFalse: [catFiles asSortedCollection do:
        [:each| work tab; nextPutAll: each; cr]].
work cr;
      nextPutAll: ′Mpr files:′;
      cr.
mprFiles isEmpty
      ifTrue: [work tab; nextPutAll: ′No .mpr files needed′; cr]
      ifFalse: [mprFiles asSortedCollection do:
        [:each| work tab; nextPutAll: each; cr]].
```

Required CAT and MPR Files for a Reduced Run-time Image

Tip

The following script can be run to determine the .cat and .mpr message files that are needed by a reduced image at run time. It reads the spusage.es file produced by packaging, so you should run the script right after your packaging run so that you have the correct spusage.es file available. The required run-time message files will be listed on the Transcript.

Smalltalk

```
| spusageFile fileStream class classCollection
appCollection catFiles mprFiles work icxFileName |

classCollection := Set new.
appCollection := Set new.
catFiles := Set new.
mprFiles := Set new.
spusageFile := ′spusage.es′ asFileSpec.
fileStream := spusageFile cfsStreamReadOnlyIfError:
  [ :aCfsErr| aCfsErr printString].
(fileStream skipToAll: ′Output statistics for: ′)
    ifTrue: [icxFileName := (fileStream nextLine)].
(fileStream skipToAll: ′#     Size    #     Size    #     Size′) ifTrue: [
    fileStream nextLine; nextLine.
    [fileStream atEnd] whileFalse: [
        (class := fileStream upTo: $ )isEmpty
            ifTrue: [fileStream setToEnd]
            ifFalse: [
                classCollection add: (class asClass).
                fileStream nextLine]]].
fileStream close.
    classCollection do: [:eachClass | |app|
    app := eachClass controller.
    appCollection add: app rootApplication].
appCollection do: [:eachApp | |poolNames mprName|
    poolNames := eachApp definedPoolNames.
    poolNames do: [ :eachPoolName| |pool catName|
        (pool := (Smalltalk at: eachPoolName
          ifAbsent:[])) epIsPoolDictionary
            ifTrue:[(catName := (pool at: ′!CATALOGNAME′
              ifAbsent: [])) isNil
            ifFalse: [catFiles add: (catName,′.cat′)]]]].
    (mprName := eachApp abtNlsRawFilename) isEmpty
        ifFalse: [mprFiles add: (AbtNLSCoordinator
         resolveLanguageMappingCharacter: mprName)]].
```

Smalltalk (continued)

```
work := Transcript bringToFront.
work cr;
      nextPutAll: ′******* The required message files for ′,
        icxFileName,′ *******′;
      cr;
      cr;
      nextPutAll: ′Cat files:′;
      cr.
catFiles isEmpty
      ifTrue: [work tab; nextPutAll: ′No .cat files needed′]
      ifFalse: [catFiles asSortedCollection do:
        [:each| work tab; nextPutAll: each; cr]].
work cr;
      nextPutAll: ′Mpr files:′;
      cr.
mprFiles isEmpty
      ifTrue: [work tab; nextPutAll: ′No .mpr files needed′]
      ifFalse: [mprFiles asSortedCollection do:
        [:each| work tab; nextPutAll: each; cr]].
```

Format Change for CTLs

Tip

The format of the .CTL files used during feature load and unload has changed between Version 3 and Version 4. This change will affect you if you create .CTL files for your components.

Sample V3 CTL File Format

Smalltalk

```
 SOMsupport ″AbtCommonProductInstallerApp products: #(′va′)
   platforms: #( ′aix′ ′os2′ ′win95′ ′winnt′ ) sys: ′es′″
′SOMsupport′ ′V 3.0a′ abtesm30.dat
′AbtSOMsupport′ ′V 3.0a′ abttsm30.dat
```

Sample V4 CTL File Format

Smalltalk

```
 SOMsupport, Base ″AbtCommonProductInstallerApp
    productId: ′sdcs00000023′
    platforms: #(′win95′ ′winnt′ ′os2′ ′unix′)″
include=abttvv40.ctl
include=abtesm40.ctl
′AbtSOMsupport Run′ ′ts=3032438045′ ′abttsm40.dat′
′AbtSOMsupport Run V4.0′
′AbtSOMsupport Edit′ ′ts=3032437984′ ′abttsm40.dat′
′AbtSOMsupport Edit V4.0′
```

Here is what has changed:

- Feature identification
  - Text up to the first double-quote forms the feature name that will display in the Feature load and unload dialog
  - The products: and sys: parameters are no longer used.
  - The new productId: parameter is used to determine whether a feature is for evaluation or for production. If you use sdcs00000023 for the value of this parameter, your evaluation status will track with the evaluation status of the base VisualAge product. If you use an unrecognized string for this parameter, your component will always be treated as for evaluation.
  - For the platforms: parameter, *aix* has been replaced with *unix.*
- Prerequisite loading

  Prerequisite features are indicated with the include statement referencing the .ctl file of the prerequisite. There is one include statement for each direct prerequisite feature. In this example, SOMsupport, Base for IBM Smalltalk, and VisualAge Base are the direct prerequisite features of SOMsupport, Base for VisualAge. These two lines are equivalent to the second line in the Version 3 example.
- Feature loading

  The final two lines in the example identify the configuration maps comprising the feature. Each line is composed of four parts:
  - Name of the configuration map to be loaded
  - Time stamp of the configuration map to be loaded
  - Name of the .DAT file containing the code to be loaded

  – A comment (typically the name and version of the configuration map to be loaded)

These two lines are equivalent to the third line in the Version 3 example.

To have your component listed in the VisualAge section of the load and unload dialog, make the fourth letter of your DAT file name a *t*. To have your component listed in the IBM Smalltalk section of the load/unload dialog, make the fourth letter of your DAT file name an *e*.

Unloading the Visualization Feature

Tip

To unload the Visualization feature, first execute the following code from a workspace:

Smalltalk

```
DtEventProfilerApplication eventQueueProcessor.
```

Then unload the Visualization feature.

Manager Access

Tip

Watch out for the following manager access issues:

Solaris Server

- Error 1212 ″Transaction terminated prematurely″ message

  When running EMSRV on Sun Solaris 2.5.1, you may get an Error 1212 ″Transaction terminated prematurely″ message when loading a large config map. This is an access denied error due to an incorrect return value from one of the Solaris lock checking functions. Temporary patches from Sun are available to fix this problem. You can download these patches from Sun′s FTP site: ftp://sunsolve1.sun.com//pub/patches/T103640-06.tar.Z.

- Starting EMSRV

  EMSRV for Solaris does not determine the device number correctly. In most cases, images that talk to it fail to connect with the message ″File resides on unsupported device.″ This error also prevents the use of the EMCOPY command.

To resolve the problem, start EMSRV with all the options you normally would use in addition to option -xd0. For instance, if your normal EMSRV startup options are:

```
emsrv -v -b50 -lt240
```

then you should use:

```
emsrv -v -b50 -lt240 -xd0
```

Note that when you use -xd0, you lose the protection mechanism that prevents you from starting EMSRV on an NFS drive. To avoid corruption, take extra care to ensure that you run EMSRV from a local hard drive when you start with this option.

Windows 95 Client

From a Windows 95 client, you must use Novell′s Client32 for Windows 95 requesters and associated patches. If you use the wrong requesters, library corruption is possible.

The Client32 for Windows 95 requester and the required patches can be obtained from Novell′s Web site.

From the Core OS Updates section, select *landr9.exe*; from the Client Kits & Updates section, select *cltdr2.exe*. Because fixes may be updated or files renamed over time, you may find that these files are no longer available. In this case, follow the instructions at the top of the page for accessing Novell′s Product Support area.

If these two patches are not used, users may experience severe performance problems when reading from attached servers and may also find that locks are not correctly released if more than one image is running concurrently on the same machine. The two patches should correct both of these problems.

The Client32 for Windows 95 requester has many parameters that may be changed in the Advanced Settings dialog. The default settings provided on installation have been tested; in most cases there is a trade-off between data integrity and performance (for example, True Commit set to off). Although other settings have not been tested, changing any of the settings in the Performance – Cache parameter group to less optimistic values should not present any problems.

When using this configuration, set ENVY/Manager′s release lock mode parameter to *true*.

Windows NT Client

Windows NT clients must only use the Microsoft Client Services for Netware (CSNW) that is provided with Windows NT 3.51. The Novell Netware Client for Windows NT should not be used as it has not been tested with VisualAge for Smalltalk.

When using this configuration, set ENVY/Manager′s release lock mode parameter to *true*.

OS/2 Warp 3 Client

From an OS/2 Warp 3 client, use Novell′s OS/2 2.11 requester for library access.

OS/2 Warp 4 Client

Direct access to Novell Netware servers from Warp 4 is not currently supported. OS/2 Warp 4 customers should use EMSRV for library access.

Opening Multimedia Devices

Tip Open only one multimedia device (for example, Digital Video Player) at a time. Make sure you close one device before opening another, otherwise unexpected results may occur. This is an OS/2 Warp limitation.

Using Digital Video Player on OS/2

Tip To use the Digital Video Player, move the MMOS2 directory to the beginning of your LIBPATH and PATH statements in your CONFIG.SYS file on OS/2 before starting VisualAge. This is an OS/2 Warp limitation.

Video Disk Player

Tip The wrap function does not work for the video disk player.

Crystal Wave and MWave Audio Cards

Tip

When running video or audio under OS/2 on a Crystal Wave Audio or MWave audio card, the video or audio might not terminate and thereby lock up VisualAge. Installing more recent audio drivers might help solve this problem. Using a different audio card, such as SoundBlaster, will solve the problem.

AVA Playback Windows

Tip

An AVA Playback Window in a scene of a story enables the user to use Ctrl+Esc to escape that scene in a full screen presentation even if the *Prevent user exit* option is on. This is an Ultimedia Builder limitation.

Lotus Notes

Tip

On DBCS systems, a Notes database title might be truncated when displayed in a Notes database icon. You can view the entire title in the database properties dialog. This problem occurs when the database name is longer than 32 bytes which can occur with as few as 11 Lotus multibyte characters (LMBCS). Depending on the type of characters in the database title, such truncation might lead to an unviewable character at the end of the title.

Chapter 3. Graphical User Interface

In this chapter we cover issues related to graphical user interface (GUI).

GUI Architecture

Question What GUI architecture does IBM Smalltalk use?

Answer IBM Smalltalk has many advantages in its GUI implementation.

ParcPlace VisualWorks Smalltalk emulates all the supported operating systems GUIs in Smalltalk—that is, it opens a window, and does all the bit manipulation itself. This makes it very flexible for cross-system development but there is a performance overhead. Digitalk (now ParcPlace) Visual Smalltalk uses more of the underlying operating system GUI, for better performance but at the price of only supporting a common subset of GUI controls; it′s better with Version 3, but complex controllers still require quite a lot of work.

IBM Smalltalk maps the Motif GUI widgets onto the underlying operating system widgets. This means you get a very rich kit to build your GUI, but the widgets operate at near native speed. You have the best of both worlds.

Mapping Widgets Across Systems

Question How do I map Motif widgets to Smalltalk widgets?

Answer Each Smalltalk widget that maps to a Motif widget class must implement two class methods:

- platformWidgetClass
- platformWidgetGadget

The XPlatformAdministration class>>#initializeWidgetClasses method sets up global variables in PlatformGlobals used by each of these methods (for example, XmPushButtonWidgetClass). These map a Motif widget class (an address) using the CwWidgetClassMappings variable of CwBasicWidget into a Smalltalk widget class (see CwBasicWidget class>>#widgetFor:).

As a result, you can map only one Motif widget class into a single Smalltalk class. (Note: Multiple Motif widget classes can be handled by the same

Smalltalk class. Several classes do that, see implementors of *platformWidgetClass*.)

One possible particular scenario is: You create a subclass of CwPushButton, and when the class #initialize method is run (CwBasicWidget), the CwWidgetClassMappings table is rebuilt. This causes all Motif push button widget class instances (an address) to be mapped into the new subclass. The new CwPushButton subclass could add #platformWidgetClass and #platformWidgetGadget methods to make them unique (not inherit from CwPushButton), although I don′t know what address they could specify, since it is truly a Motif push button.

Despite all this verbose explanation, I cannot explain why this particular walkback occurs, but I argue it is largely academic because even if there wasn′t a walkback, simple subclassing of a CwBasicWidget would not work. Instead, subclass a CwExtendedWidget where the primary widget is a push button.

Motif Documentation

Question Is there any formal documentation as to which Xm... entries are available?

Answer VisualAge for Smalltalk does not use any application-specific X resources, but it does inherit the standard Motif/Xt/X11 resources by virtue of being an X client and using the native widgets. For a reference on available resources, check out the standard Motif/Xt/X11 manuals, such as those from O′Reilly and Prentice Hall.

Interface to Windows 95

Question Will VisualAge for Smalltalk provide an interface to Windows 95 integration features?

Answer VisualAge for Smalltalk is meant primarily as a standard-compliance-oriented tool. For example, even though you have the Windows 95 control support in Version 4, I strongly suggest that you abandon Windows 95-specific controls in the interest of portability through Common User Access (CUA) 91 standard compliance.

Applications with Multiple Windows

Question How do I create applications with multiple windows?

Answer View Wrappers can be extremely useful when building complex applications with many windows because of the ability to conserve screen real estate in the Composition Editor. The View Wrapper performs better than the variable approach because the reusable part is not constructed by VisualAge for Smalltalk until the openWidget connection executes.

The View Wrapper is essentially an icon that represents another view that you have created. Instead of taking up valuable Composition Editor space, you can place this icon on the free-form surface and make connections to and from it. After adding a View Wrapper part to the Composition Editor, change its type to the class name of a view that you have already created. Since the View Wrapper is a place holder for a view, you make connections to it as if it were the actual view class.

Exit Dialog Message Box

Question I am attempting to create an exit dialog message box that prompts the user to respond to the question ″Are you sure you want to exit?″ when they either select **Close** from the system menu or double click on the system menu.

Answer Create an event-to-script connection from closeWidgetRequest (of the window) to the following new script:

Smalltalk

```
okayToClose: aDoit

   aDoit doit: (true == (System confirm:
        ′Are you sure you want to exit?′)).
```

Window Location

Question How do I specify where my window will appear on the screen when open?

Answer There are two ways to do this if your window is a visual part.

- Think of the free-form surface of the Composition Editor as your display; if your window is at the top left of the free-form surface, it will be at the top left of your display.
- Assuming that you want to place it specifically at test or run time (and not affect its position in the Composition Editor), you could connect from *aboutToOpenWidget* of the window to a script like the following:

Smalltalk

```
(self subpartNamed: ′Window′) yourself primaryWidget
    x: 200;
    y: 400.
```

Also, you could use the view′s *finalInitialize* method to do something like the following:

Smalltalk

```
finalInitialize

  self initWidgetSize: self computeSomeSizeAndPosition.
  ^super finalInitialize
```

Positioning Window at Cursor

Question How do I initially position the window where the cursor is?

Answer Look at the following code:

Smalltalk

```
finalInitialize
| rootXReturn rootYReturn |

  CgDisplay default
  queryPointer: CgWindow default
  rootReturn: (ReturnParameter null)
  childReturn: (ReturnParameter null)
  rootXReturn: (rootXReturn := ReturnParameter new)
  rootYReturn: (rootYReturn := ReturnParameter new)
  winXReturn: (ReturnParameter null)
  winYReturn: (ReturnParameter null)
  maskReturn: (ReturnParameter null).
  self initWidgetSize:
     (Rectangle origin: rootXReturn value @ rootYReturn value
         extent: 300 @ 200).
  ^super finalInitialize
```

Maximizing a Window

Question Is there a way to maximize a window programmatically?

Answer There are platform-specific ways to maximize and minimize a window:

- In Windows:

Smalltalk

```
″Code for maximizing a window:″
   self primaryWidget osWidget handle
       showWindow: SwShowmaximized

″Code for minimizing a window:″
   self primaryWidget osWidget handle
       showWindow: SwShowminimized
```

- In OS/2:

Smalltalk

```
″Code for maximizing a window:″
(self primaryWidget osWidget handle winQueryWindow: QwParent)
winSetWindowPos: nil x: 0 y: 0 cx: 0 cy: 0 fl: SwpMaximize

″Code for minimizing a window:″
(self primaryWidget osWidget handle winQueryWindow: QwParent)
winSetWindowPos: nil x: 0 y: 0 cx: 0 cy: 0 fl: SwpMinimize
```

Opening Window Maximized

Question How can I cause a window to be opened maximized?

Answer Let′s apply what we just learned. Create a script like the following:

Smalltalk

```
maximizeWindow
″Maximize window in OS/2″

  (self primaryWidget osWidget handle winQueryWindow: QwParent)
     winSetWindowPos: nil x: 0 y: 0 cx: 0 cy: 0 fl: SwpMaximize
```

Then, create an event-to-script connection from the #openedWidget event of the window to the script you just wrote, and enjoy.

Display Resolution

Question How can I retreive the display resolution?

Answer The following code snippet buys you the height and width of the screen:

Smalltalk

```
screenHeight := CgScreen default width.
screenWidth := CgScreen default height.
```

Opening Window Centered

Question How do I open a window centered on the screen depending upon its resolution?

Answer Connect the *aboutToOpenWidget* event of your window to a setPosition method which determines window position based on screen size:

Smalltalk

```
setPosition
| w |
 w := self primaryWidget.
 w
   x: ((CgScreen default width - w width) / 2) asInteger;
   y: ((CgScreen default height - w height) / 2) asInteger.
```

Window Space Planning

Question How do I calculate the space taken up by the window border, title bar, and so on?

Answer In OS/2, you can query the title bar height, menu bar height, and border width using OSCall>>#winQuerySysValue:, for example,

Smalltalk

```
 (PlatformGlobals at: ′HwndDeskTop′)
   winQuerySysValue: (PlatformConstants at: ′SvCytitlebar′)
```

should return you the title bar height.

Iterating over Widgets

Question How do I iterate over all widgets in a window?

Answer Use #recursiveViewsDo: instead of #components do: against the main window. This will ensure all widgets are touched; it is necessary when adding callbacks to a window in Windows.

Modal Dialog

Question How can I get my modal dialog to return a value?

Answer I suggest adding an instance variable called *result* to your view, and implementing a *prompt* method:

Smalltalk

```
prompt

  result := nil.
  self suspendExecutionUntilRemoved.
  ^result
```

You would open the view like this:

Smalltalk

```
(view := MyView new) openApplicationModalWidget
myResult := view prompt.
```

Capturing the Minimize Request

Question Is there a way to capture the minimize request similar to the CloseWidgetRequest? I do not want a window to minimize to the desktop, I would like to create a pseudo desktop within my application and minimize to it.

Answer You get notified, but cannot disallow it.

Window Synchronization

Question How do I open a window from one method and wait on a return from the window in the method?

Answer Try the following code snippet:

Smalltalk

```
myWindow openWidget
 ″or use any other message, such as openOwnedWidget″
myWindow suspendExecutionUntilRemoved.
  ″code to be executed, when the window is closed goes here″
```

Closing a Window

Question What is the difference between closeWidget and destroyPart for a window?

Answer The *closeWidget* action will destroy the widget. The *destroyPart* action will destroy the widget and release all connections. You want to destroy the part when closing the owning window to be sure that all connections are freed; otherwise dependencies will be left around and your image size will grow.

Resolution-Independent Screen

Question When I change the screen resolution, say from 800*600 to 1280*1024, the text does not seem to scale properly. If there is a text label before an input field, the text ends up overlapping the input field. Is there a platform independent solution to fix this?

Answer The best way to do this is by using widget attachments. Using the widget attachments, attach the right edge of the label field with either no attachment or attach it to its own left edge with 9 to 11 offset points per character of text in the label. Then, attach the left edge of the text field to the right edge of the label field with an offset of 2 or 3 pixels. To be safe, attach the top and bottom of the text field to the top and bottom of the label field with an offset of zero. If you want to get really fancy, group controls on a form part or in a group box and offset the controls from each other and the sides of the form or group and then offset the forms and group boxes.

Setting Minimum Window Size

Question How can you set or specify the minimum size for a window?

Answer The following script checks, after each window resize, whether it has exceeded a predefined minimum size. If so, it sets it to the predefined minimum size. Do not forget to connect the *resized* event of the window to this script.

Smalltalk

```
checkResize
| tempWidget minimumWidgetSize |
  minimumWidgetSize := #(400, 200).
  tempWidget := self primaryWidget.
  ( tempWidget width < minimumWidgetSize first ) ifTrue: [
    tempWidget width:minimumWidgetSize first].
  ( tempWidget height < minimumWidgetSize last ) ifTrue: [
    tempWidget height:minimumWidgetSize last]
```

Closing a Widget from a Script

Question I have created a view in which I show some customer information from an ordered collection that I dropped on the free form. I also dropped a button and connected it to the following script:

Smalltalk

```
return
″Close the widget and return.″

self closeWidget.
```

When testing the script, the widget closes, but a dependent of ordered collection remains in the Dependent dictionary. Is this not the proper way of closing a view? I have noticed that if you close the view by using ALT-F4 from the system menu, the Dependent dictionary does not contain the ordered collection.

Answer You are leaving dependents lying around because you are using the private *closeWidget* method. You should use the *closeWidgetCommand* method instead. You could use the Script Editor to paste the proper action *closeWidgetCommand* into your method.

Removing the Close Menu Option

Question How do I disable or remove the Close menu option on the system menu for a window?

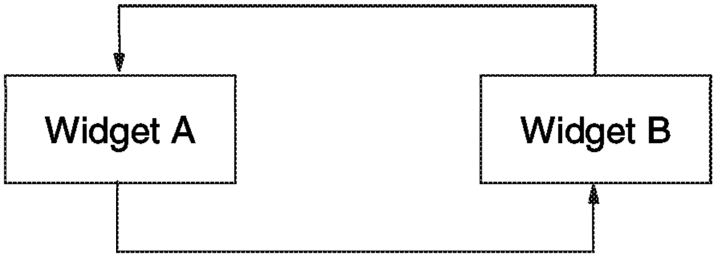
Answer You can′t, but you can add a *closeWidgetRequest* callback. That is a callback that is invoked before a shell is closed. If you set the ″doit:″ to *false*, the window is not closed. Normally this is used to display an ″Are you sure?″ message before closing the window. See AbtApplicationsOrganizer>>#confirmCloseRequest: for an example. It hooks the *closeWidgetRequest* event to a script.

Widget Warning: Circular Reference

Question I am getting a warning in the Transcript: ″CwForm circular reference ...″ My application works fine, but I would like to get rid of the warning. What should I look for?

Answer This means that you have one widget whose attachments are dependent on another widget, and the other widget′s attachments are dependent eventually (if not immediately) on the first widget.

Here′s an example of circularity. Widget A′s bottom edge is attached to Widget B′s bottom edge, while Widget B′s left edge is attached to Widget A′s right edge.



The nonobvious restriction with attachments is that in order for Widget B to successfully attach to Widget A, Widget A′s edges must be fully resolved. That is, they can′t be directly or indirectly (by additional widgets) based on any of the edges of Widget B.

Always-on-Top Window

Question We have built a tool bar for our application. We would like the tool bar window to always stay on top. Is there a way in VisualAge for Smalltalk to set a window to always stay on top? I know there is a windows application programming interface (API) that will keep a window on top, but I do not know how to get to if from VisualAge.

Answer An easy way to do this is to use CwOverrideShell rather than CwTopLevelShell as the shell class for the window. This will cause it to float on top of all other windows. If you want the tool bar to float on top of a specific window, then make the tool bar a child of the other window (by setting its parent to be the other window). Either of the above methods requires digging into IBM Smalltalk a bit.

Changing the Mouse Pointer

Question When the mouse pointer enters into the scope of an editable field, the pointer changes from the arrow to the I-beam. Is it possible to accomplish a similar behavior while crossing over some arbitrary other control, like a form or button?

Answer To change the shape of the pointer when moving over a button, do the following:

1. Create a new visual part.
2. Put a push button in the window.
3. Go to the Script Editor. In the class definition for your part, add an instance variable named *cursor* and save the class definition.
4. Implement the following instance methods:

Smalltalk

```
changeCursor
  ″Change cursor for Push Button1.″
  | display |

  display := (self subpartNamed: ′Window′) yourself
     primaryWidget display.
  self freeCursor.
  self cursor: (display createFontCursor: (CgConstants at: ′XCGumby′)).
  (self subpartNamed: ′Push Button1′) yourself
     primaryWidget window
        defineCursor: self cursor.
```

Smalltalk

```
cursor
  ″Answer the currently selected CgCursor, or nil.″

  ^cursor
```

Smalltalk

```
cursor: aCgCursorOrNil
  ″Set or clear the currently selected CgCursor.″

  cursor := aCgCursorOrNil
```

Smalltalk

```
freeCursor
  ″Free the cursor if one is allocated.″

    self cursor notNil
      ifTrue: [ self cursor freeCursor ]
```

Smalltalk

```
undefineCursor
  ″Undefine the cursor for the window.″

  (self subpartNamed: ′Push Button1′) yourself primaryWidget
     window undefineCursor.
```

5. Go back to the Composition Editor and make the following connections.

   a. Connect *openedWidget* of the window to the *changeCursor* script.

   b. Connect *aboutToCloseWidget* of the window to the *freeCursor* script.

   c. Connect *aboutToCloseWidget* of the window to the *undefineCursor* script.

   Note that the order of the two last connections is important.

6. Test your part and enjoy!

Busy Cursor

Question Can #showBusyCursor in a development application be packaged with the user application?

Answer The *Programmer′s Guide to Building Parts for Fun and Profit* suggests that you create two different applications, *Edit* and *Run* You package only the *Run* applications when creating your packaged image. The same applies to VisualAge for Smalltalk applications and parts themselves.

As AbtShellView>>#showBusyCursor is defined in AbtRunViewsApp, it is packaged into your run time image.

Note this!

Please don′t be confused by similarity in the names of *showBusyCursor* and *showBusyCursorWhile:.* The method *showBusyCursorWhile:* is an instance method of the class EmSystemConfiguration. Its comment says, ″Show the busy cursor during execution of aBlock. The cursor is not changed if the graphics system has not been started.″

The following code shows you another way you can change the cursor. The example can be evaluated in a Transcript window:

Smalltalk

```
| window cursor |

window := Transcript shell window.
cursor := CgDisplay default busyCursor.
window defineCursor: cursor.
(Delay forSeconds: 5) wait.
window undefineCursor.
```

By the way, do not explicitly free this cursor. As for disabling the mouse you might want to consider one of the following:

1. The easiest way to disable the mouse pointer is to use the message grabPointer:. (This method is an instance method in CwComposite.) The argument is a cursor. So, you can send it the clock cursor and you won′t have to set that up yourself. This method sends all mouse input to the shell, regardless of where the pointer is on the screen. This is also the bad part, as the user can′t use the mouse for other things. When you are done, send the message ungrabPointer.

2. Another way is to add an event handler when you start the processing. You can do something like this:

   Smalltalk

   ```
   shell addEventHandler: ButtonPressMask
     receiver: self
     selector: #eventHandler:clientData:event:
     clientData: nil.
   ```

   The advantage of this method is that it applies only when the mouse pointer is in the window. Call the above when you want the input blocked. When you are done, call the above again, except change *addEventHandler* to *removeEventHandler.* Make sure to add CwConstants to the class′s pool dictionaries. Now you need to add the method eventHandler:clientData:event:, which gets called when there is input. You can either leave this method blank to do nothing, or maybe use the following option.

3. To implement a beep, add the PlatformFunction pool dictionary to the class. The following is valid for OS/2:

Smalltalk

```
eventHandler: widget clientData: clientDate event: event
″There has been user input.  Beep.″
DosBeep callWith: 100 with: 200.
```

Caret Shape Cursor

Question I have some text entry fields where the cursor is positioned at the right edge. If I tab into these (empty) fields, I nearly cannot detect in which field the focus is actually set because of the look of the cursor which is a thin line. How can I define another caret-look for entry fields?

Answer You can′t change the caret look for your entry field. The entry field input cursor, known also as the caret (″I″ bar) is defined by the operating system and cannot be changed by an application. Some operating systems have configuration options to change the mouse cursor, resize cursor, and the text input cursor. If you want a VisualAge for Smalltalk solution, you could connect the *gettingFocus* and *losingFocus* event to a label′s *show* and *hide* actions. Put the label next to the entry field, as

Address ________ *

The asterisk will show when the field has the focus, and hide when it doesn′t.

Mouse Pointer Location

Question How can I find the position of the mouse cursor on the screen?

Answer You can detect the cursor location in one of two ways: You can call the OSWidget class method *widgetUnderCursor* to see where the cursor is, or you can add a Pointer Motion Mask event handler to find where the cursor is and see if it′s in the desired area (see page 192 of the *IBM Smalltalk Programmer′s Reference* for an example of the latter). You can change the font of a cursor by following the example on page 124 of the *IBM Smalltalk Programmer′s Reference*.

Icon as Cursor

Question How can I use an icon as cursor?

Answer The follwing code snippet reads an icon out of a DLL and uses it as a cursor.

Smalltalk

```
| window data pixmap cursor shell label icon gc image |

shell := CwTopLevelShell
  createApplicationShell: ′shell′
  argBlock: [:w | w title: ′label example′].
  shell width: 200;
  height: 200;
  x: 1;
  y: 1.

label := shell
  createLabel: ′label′
  argBlock: nil.

label labelString: ′this is a label.′.
label manageChild.
shell realizeWidget.

window :=  shell window.

″get an icon from a file″
icon := CgIcon
  fromResource: 153
  fileName: ′d:v3clientdllabticons.dll′.
```

Smalltalk (continued)

```
″create a pixmap from the icon″

pixmap := window
  createPixmap: icon width
  height: icon height
  depth: 1.

″get device independent image″
image := icon shapeImage.
gc := pixmap
  createGC: 0
  values: nil.

″create a rectangular area of data for the cursor″
pixmap
  putDeviceIndependentImage: gc
  image: image
  srcRect: (0@0 extent: image extent)

  destRect: (0@0 extent: image extent).

″Create the cursor.
 I hardcoded a hotspot at 20,20.
 A better idea might be to use
 image width and image height.″

cursor := pixmap
   createPixmapCursor: pixmap
      x: 20
      y: 20.
pixmap freePixmap.

″Display the cursor.″
window defineCursor: cursor.
(Delay forSeconds: 5) wait.
window undefineCursor.
cursor freeCursor.
gc freeGC.
```

Animated Busy Cursor

Question How can I display an animated busy cursor similar to the activity indicator on the print preview? I know I can use the System *showBusyCursorWhile:* method that displays the system clock, but sometimes the users may think the system is locked.

Answer Check out the spinning clock in the VisualAge for Smalltalk Help Examples for an example of how to make an animated progress indicator.

Adding Pull-Down Menu Choices Dynamically

Tip Listed below is some sample code that you can use to dynamically add pull-down menu choices. You will need to add a callback method to do some processing when the user clicks the new pull-down choice.

Smalltalk

```
addMenuChoice
″Dynamically add a new menu choice to a menu ″

| newMenuChoice |
newMenuChoice := ((self subpartNamed: Menu1′) widget)
  createPushButton:  New Choice′
  argBlock: nil.
newMenuChoice addCallback: XmNactivateCallback
  receiver: self
  selector: #newChoicePushed:clientData:callData:
  clientData: nil.
newMenuChoice manageChild.
```

Toggle Buttons in Menu

Question I want to have a menu with toggle buttons *Sort by Date* and *Sort by Name* (and possibly some other sort criteria). I want to have a check mark in one of these buttons only. I tried to connect the *selectionChanged* events to a script but for some reason this script runs only once. What would be a way of making this work?

Answer You′ll need to write a script for each toggle button and connect it to the selectionChanged event. Here′s what the scripts might look like:

Smalltalk

```
sortByNameSelectionChanged: selectionBoolean
″Sort by name.″

selectionBoolean == true
  ifTrue:
    [ (self subpartNamed: ′SortByName′) disable.
    (self subpartNamed: ′SortByDate′) clear; enable.
    <perform sort by name statements> ]
```

Smalltalk

```
sortByDateSelectionChanged: selectionBoolean
″Sort by date.″

selectionBoolean == true
  ifTrue:
    [(self subpartNamed: ′SortByDate′) disable.
    (self subpartNamed: ′SortByName′) clear; enable.
    <perform sort by date statements>]
```

The selectionBoolean value needs to be checked because the statements should only be executed when the menu item is checked, not unchecked.

Disabling Push Buttons on Pop-Up Menus

Question How do I disable push buttons on pop-up menus?

Answer You can use a *setSensitive:* method to disable a push button. The default setting for the value of the XmNsensitive resource is *true*. This method will allow you to set it to *false*.

If you want the push button to be disabled initially, try using the *createPopupMenu:argBlock:* method. When you create a menu through a Smalltalk script, you first create the menu, and then add individual buttons to it. Each button can then be enabled or disabled. An example of how to do this is listed below:

Smalltalk

```
createPopup: parent
| menu |

menu := parent createPopupMenu: ′popup′ argBlock: nil.
(menu createPushButton: ′Add′ argBlock: nil)
   addCallback: XmNactivateCallback
   receiver: self
   selector: #addSomething^clientData^callData:
   clientData: nil;
   setSensitive: false;
   manageChild.
^menu
```

Pop-up Menus for Icon Gadgets

Question How do I dynamically create pop-up menus for dynamically created icon gadgets?

Answer To dynamically create a pop-up menu for the dynamically created icon gadget, use the following:

Smalltalk

```
″Create an icon gadget for each item and set its attributes.″

iconGadget := (AbtIconGadgetView new
  label: ′MyIcon′;
  largeIcon: icon;
  parentView: container;
  openWidget;
  yourself).
```

Instead of making the association with the menu before the *openWidget* message, add the following code afterward to associate the pop-up menu with the icon:

Smalltalk

```
iconGadget menu: ((self subpartNamed: ′Menu1′)
      parentView: container;
      openWidget).
```

Sharing a Menu Between Containers

Tip If you want to share a menu, you will need to create a reusable menu part and add multiple instances of it for connecting to each container. See page 413 of the *VisualAge for Smalltalk User′s Guide* for details.

Dragging Versus Pop-Up

Question What is the difference between obtaining a pop-up menu and dragging?

Answer Drag does not start unless the mouse is moved a threshold amount. If we are not in drag mode, the pop-up is displayed. (This is a general algorithm; it varies slightly, depending on the platform you are running on.)

Icons on a Menu Bar

Question I tried adding an icon as a menu bar item and a menu pull-down item. To do that, I created a menu as usual, then selected **Open Settings** from the **Menu Choice** pop-up menu, then clicked on **Label Type - Graphic, Graphic Type - Icon, Module name: abticons, ID: 1**. That gave me a file cabinet icon on the menu, but the icon was displayed as a bitmap—that is, the background was white, not transparent. The comment in the AbtLabelView>>#calcGraphicLabelType says ″For OS/2 and Windows, only bitmaps (not icons) are allowed as menu items, so if the parent will be a menu, then force the graphic to be a pixmap.″ Is there a remedy?

Answer I′ve not seen a menu bar at the bottom of the frame, but that is not native widget behavior. The word processor must be implementing this (or perhaps, to use PM-terminology, overriding WM_FORMATFRAME to move it to a new location). You may want to investigate using an AbtRowColumnView for building your tool bar. It can be attached to the bottom, the top, or wherever on your window. It is a special part that automatically attaches buttons to each other. To see an example, do the following:

1. From the **Options** menu select **Add Part** in the Composition Editor.
2. Type in **AbtRowColumnView** as the class and select **Part** in the radio button set.
3. Drop the part within your window, attaching it at the top and sides to the window.
4. Drop several push buttons within the row column part. Use some that are graphic and some that are text, just to see how it works.
5. If you want a scrollable tool bar, you can drop a scrolled window part (from the Canvas palette) and connect the workView of the scrolled window to the row column part (moving the row column off to the side).

Creating a Tool Bar

Question What would be an easy way to create a tool bar for an application? Can you simply add a menu bar to the window and have the items displayed as icons?

Answer Basically, what you need is a part that you can put onto a window and have the attachments made automatically to the parent. This part would also have to accept buttons that are dropped onto it in edit mode, and automatically make the attachments from the button to the tool bar frame.

To do this you could use the AbtFrameView and the AbtRowColumnView. These parts are not on the palette, but you can still drop them onto the Composition Editor. Drop the row column into the frame view. You need to open its settings and make sure that it is connected on all sides to the frame with preferably a offset of zero. Now, when you drop your push buttons inside the row column, they automatically space apart, making a tool bar a little easier to deal with.

Synchronizing a Combo Box

Question Suppose you have two combo boxes dropped on a window. The first combo box contains a set of names, while the second combo box contains valid descriptions for each of these names. How can you synchronize the two in such a way that when you type an invalid code in the first combo box, the second one will be cleared?

Answer One of the ways of doing this is just to connect the *selectedIndexChanged* event of the first combo box to a script, where you validate the index, and modify the second combo box accordingly.

Adding Items to Combo Box

Question How do I allow a user to add new items to a combo box?

Answer Try this:

- Hook a keyboard event handler to the comboBox
- Within the event handler, sort out any Enter keys
- Whenever you encounter an Enter key you do something like

Smalltalk

```
<combobox> items add: (<combobox> object).
```

In order to invoke the keyboard event handler, you hook a method to the *aboutToOpenWidget* event of your main window; in this script you include:

Smalltalk

```
<combobox> widget addEventHandler: KeyReleaseMask
  receiver: self
  selector: #keyRelease:clientData:event:
  clientData: nil.
```

Then you can add a *keyRelease:clientData:event:* method to your class which will be called for each key entered while the combo box has the input focus.

Drag-and-Drop Support on List Boxes

Question Is it possible to have drag and drop support for list boxes? The documentation only describes the situation for containers. I would like to give the user the ability to move an item in a list box to a new position in the same list box using drag and drop.

Answer

The drag-and-drop concepts are fairly universal. The nice things containers will do for you is let you specify what sort of object can be moved, copied, or linked.

Basically, the drag-and-drop callbacks in IBM Smalltalk are exposed as events for your list box in VisualAge for Smalltalk. If you open the settings for the list box and select drag/drop on the notebook page, the events are added to the connection pop-up menu. These events are exactly the same as the callbacks listed in Chapter 9 of the *IBM Smalltalk Programmers Reference*.

The following example illustrates how to drag and drop items within a list box:

1. Add a list box to any window.
2. Open its settings and go to page Drag/Drop: select **Allow Drag** and **Allow Drop and Move** as the valid default operation in both cases.
3. Use any collection as the items of the list, as usual.
4. Connect the *dropped* event of the list to the following script:

Smalltalk

```
moveItems: aCallback
| items itemsDropped position index |

items := aCallback sourceModel.
itemsDropped := aCallback sourceItems.
itemsDropped isEmpty ifTrue: [ ^nil ].
index := items indexOf: itemsDropped first.
position := aCallback event y // aCallback offsets first y negated.
itemsDropped do: [ :item | items remove: item ifAbsent: [ ] ].
index < position ifTrue: [ position := position - 1 ].
position > items size ifTrue: [ position := items size ].
items addAll: itemsDropped afterIndex: position.
aCallback sourceWidget userData items: items.
```

The example works for a list and for a multiselect list.

List Box Behavior

Question When the focus is on the list box part and the user clicks on a character key, the list box should scroll to the first item begining with this character. When typing another character, the list box should scroll to the next item that starts with the character that was typed first, followed by the character that was typed second. Is it possible to get this behavior from the list box parts, and if so, how?

Answer On OS/2 and Windows, pressing a character scrolls to the list box entry that starts with that character. You might override this behavior by creating a form around the list box and using CwComposite>>interceptEvents: to trap the events going to the list box.

But an easier solution (and one you see see many times) is to add a **Search** entry field above the list box and scroll the list box with CwList>>topItemPosition: as the user types in it.

Identifying Last Selected Item

Question For our application, we would like to know what the last selected item in our multiple-select list was. We need this, because the user can press a button that will open a detailed view on that item. I would like to see an attribute such as *last item selected*. Any ideas how to achieve this?

Answer There is no public attribute available on the multiple-select list part that will tell you the last item selected. However, with a tiny bit of code, you can get this information. Here is what you can do. First, connect the *aboutToOpenWidget* event of your window to the following script:

Smalltalk

```
addCallbackToList
″Set up a callback for the multiple selection list″

(self subpartNamed: ′Multiple Select List1′) widget
   addCallback: XmNmultipleSelectionCallback
   receiver: self
   selector: #multipleSelect:clientData:callData:
   clientData: nil.
```

Then, add an instance variable to your visual part′s class definition called *lastItemSelected*, with the appropriate get and set methods. Now, write the following script to maintain your *lastItemSelected* value:

Smalltalk

```
 multipleSelect: widget clientData: clientData callData: callData

 ″Save the last item that was selected.  The ′includes:′ logic will
 exclude any items that were deselected, since itemPosition will
 return the last item either selected or deselected.  Caching the
 itemPosition value for performance.″

 | currentPosition |

currentPosition := callData itemPosition.
   (callData selectedItemPositions includes: currentPosition)
   ifTrue: [
       self lastItemSelected: currentPosition].
```

Default Action Requested

Question How do I get an event that could work as the DefaultActionRequested available in a text part for a combo box?

Answer This question indicates an apparent oversight, and I′ve opened DCR 6872 for a future release. In the meantime, you can add an activate callback to the widget. For example, I hooked the following code to the combo box′s *openedWidget* event:

Smalltalk

```
addComboBoxActivateCallback: aComboBoxView
aComboBoxView widget addCallback: XmNactivateCallback
receiver: self
selector: #myComboBoxDefaultAction:clientData:callData:
clientData: nil.
```

The *myComboBoxDefaultAction:clientData:callData:* will be invoked when the user presses Enter. Or, if you prefer, you could create a subclass of AbtComboBoxView that adds the new event See also AbtAbstractTextView for an example, especially the methods *postCreationInitialization*, *defaultActionEvent*, and *defaultAction:clientData:callData:*. I suspect they could be copied to your subclass with only minor modification (and the addition of the *defaultEventRequested* to the public interface, of course). For an Multiline Edit (MLE) part, the activate callback for CwText applies only when it is single-line (since the Enter key means ″add line″ to a multiline CwText).

Combo Box Behaviour

Question I have dropped a combo box on my form. I populated it with a number of items and set one of them as the initial item. I grab the selected item and pass it to a variable. When I type a value in the combo box, it seems to search the list of items for a match (for example, on the first letter typed). Of course, it does not display anything. When I tab out of the field and inspect the variable, it has not captured the data entered in the field, but instead, it has the list item that matched on the first character. Is this the default behavior of a combo box?

Answer That is indeed the default behavior of a combo box. The entry field is just for you to be able to type in an existing entry as an alternative to using the scroll bar to find it in the list. However, you can add the behavior you describe with a combination of connections and scripts.

Skipping through a Notebook

Question I am writing an application demonstrating various features of VisualAge for Smalltalk. One of the things I want to do is to skip through a notebook without user intervention, as in a slide show. The question I have is, how can you automatically skip to the next page?

Answer You can turn notebook pages by setting the *currentPageIndex* or *currentPage*. Here is an example using delays, so that the pages turn a bit slower:

Smalltalk

```
| delay numberOfPages |

delay := Delay forSeconds: 2.
numberOfPages := (self subpartNamed: ′PMNotebook1′)
    components size.
2 to: numberOfPages do: [ :i |
  (self subpartNamed: ′PM Notebook1′) currentPageIndex: i.
  delay wait.
  ].
```

Bitmap on Notebook Tab

Question Is it possible to display a bitmap on the tab of a notebook page? And if so, how do you do this?

Answer If you would like to have a bitmap appear on the tab of a notebook page, you can use the *tabContents* attribute for the notebook page. It is looking for an AbtBitmapDescriptor. Here is a piece of sample code that puts a bitmap on the tab of a page. Connect the *aboutToOpenWidget* event of the window to the following script:

Smalltalk

```
bitmapTab

  | bitmap |
  bitmap := ((AbtBitmapDescriptor new
     moduleName:  AbtBmp20′;
     id: 101) baseGraphic).
  (self subpartNamed:  NotebookPage′) tabContents: bitmap.
```

It is also a good idea to free the bitmap after you are done using it. Use of the *baseGraphic* and the *gpiDeleteBitmap* (see below) methods is highly platform specific. The way I would go about doing this is to assign an instance variable to the baseGraphic in the bitmapTab method and connect the following script to the closeWidget event of the window:

Smalltalk

```
bitmapFree
   instanceVariableName gpiDeleteBitmap
```

This causes resources used up by this bitmap to be freed when this window is closed.

Hiding and Showing Pages

Question How do I hide and show pages of an OS/2 Windows notebook?

Answer I believe hide is not supported for notebook pages, you must destroy the page. Here is how I add a page dynamically (basically, I copy from the *abtBuildInternals* method and add an *openWidget* send):

Smalltalk

```
addNotebookPage: aNotebook named: aName
| page label |

aNotebook subpartNamed: aName
put: (page := AbtPortableNotebookPageView newPart).
page
   framingSpec: (AbtViewAttachmentConstraint new
   leftEdge: (AbtEdgeConstant new offset: 34);
   topEdge: (AbtEdgeConstant new offset: 90)).
page subpartNamed: ′label′ put: (label := AbtLabelView newPart).
   label object: Time now printString;
   framingSpec: (AbtViewAttachmentConstraint new
   leftEdge: (AbtRunEdgeAttachmentConstraint new attachment:
      XmATTACHFORM;
      offset: 65);
   rightEdge: (AbtRunEdgeAttachmentConstraint new attachment:
      XmATTACHNONE);
   topEdge: (AbtRunEdgeAttachmentConstraint new attachment:
      XmATTACHFORM;
      offset: 95);
   bottomEdge: (AbtRunEdgeAttachmentConstraint new attachment:
      XmATTACHNONE)).
page openWidget.
label openWidget.
```

To remove the page, I call *destroyPart* on the page. This is just a simplified example; in your case, you would substitute *label* for an embeddable view that represents the entire page.

Porting Notebooks across Platforms

Question If you try to port an OS/2-Windows notebook from an OS/2 platform to Windows NT or 95, the notebook is replaced by a VisualAge icon. Why?

Answer The OS/2-Windows notebook is not available on Windows NT or 95, since it is built from a 16-bit class library. However, you can migrate the notebook to the portable notebook (which is supported under all VisualAge platforms) by selecting **Morph** from the pop-up menu of the icon that you are seeing. This will morph the OS/2 notebook to the portable notebook.

Navigating between Columns

Question How do I navigate between columns in a container without using the mouse?

Answer Use CELLSINGLESELECT selection technique.

Selecting Valid Items

Question I am using a container details part. There are some rows that cannot be selected by the user (depending on the data in the object), whereas other rows may be selected. Is there some kind of callback I can capture that would allow me to deselect a row when I do not want the object to be valid?

Answer The container details will signal a *selectedItemsChanged* event every time the user modifies the selection. You can then examine the selected items, and change the selection to include only the valid objects (using *selectedItems:*). Be careful of getting stuck in a loop (some parts will resignal *selectedItemsChanged* even if the new selection is the same as the previous one). You may have to keep track of this yourself.

Validating Input Fields

Question We have a written a notebook application where data needs to be entered on different pages. After the data has been entered, I need to see wether it is valid. The following problem occurs when the user skips to the next or previous page. A prompter might pop up on the new page, displaying a message that shows the validation error from the previous page. What can we do to prevent this from happening?

Answer To solve the problem, it sounds like a good idea to use a form checker:

1. Drop a form checker part onto the free-form surface.
2. Following the example on page 149 in the *VisualAge for Smalltalk User′s Guide*, connect the *verificationRoot* attribute of the form-checker part to the self attribute of each notebook page you want to verify.
3. Add an instance variable called *lastPageSelected* to your visual part class
4. Connect the *switchedToMe* event for each page to the following script:

Smalltalk

```
saveLastPage
 ″Save the current page as the last page
 selected before the page is turned″

self lastPageSelected:
  (self subpartNamed: ′OS/2-Windows Notebook′) topPageIndex.
```

5. Connect *switchFromMe* to the *check* action of the form-checker part.
6. Create a script called *keepPageOnTop* as follows:

Smalltalk

```
keepPageOnTop
″Keep the last page selected on top of the notebook by
switching to it.″

(self subpartNamed: ′OS/2-Windows Notebook′)
    topPageIndex:  self lastPageSelected.
```

7. In the set method for *lastPageSelected*, be sure to add a catch for the special case when the notebook is first opened and the index is 0 instead of 1:

Smalltalk

```
lastPageSelected: anInteger

| realPage |
realPage := (anInteger = 0)
  ifTrue: [1]
  ifFalse: [ anInteger ].
lastPageSelected := realPage
```

8. Connect the *checkFailed* event of the form-checker part to the *keepPageOnTop* script. Also connect that same event to the *showErrorToUser* action.

Container Column List Box

Question How do I create a container details column that displays a drop-down list box for each item?

Answer Take a look at AbtContainerTreeChildrenAttributesPage and the methods *initializeHasChildrenAttributeList:* and *initializeChildrenAttributeList:*. Also take a look at EwEditPolicy. Another good place to look for examples is the EwExampleLauncher and CwExampleLauncher. You can load these by selecting the Smalltalk programmning examples from the **Load Features** menu option. EwExampleLauncher is the one that you want to examine. Type *EwExampleLauncher new open*, select the code and execute it to bring up the example launcher. I think the EwxTableListExample contains exactly what you are looking for and the Abt stuff listed above will help you incorporate the code into the AbtLayer. EwxEditPolicy also contains similar examples.

Columns with GUI Elements

Question Can I create columns that contain GUI elements, in particular check boxes, so that the user can click a button to toggle a boolean?

Answer Yes, but only for the container, as AbtTableView does not support this. Intercept the *aboutToBeginEdit* event and set the edit policy. See the *initializeChildrenAttributeList* method in AbtContainerTreeChildrenAttributesPage for an example. You would specify EwToggleButtonEditPolicy as the edit policy and initialize it appropriately.

Adding Records to Container

Question How do I add more records to a container after the user presses the page down key?

Answer The *packetRequested* event is not signaled until an *at:* request comes into the BufferedCollection and is indexing into a slot that does not have a value. Do you know if this is occurring? What is happening? How many rows have you told the container that the virtual collection has?

Changing Labels

Question How can I change the color of the label that appears with an icon?

Answer Below is an example of using the AbtContainerIconListView (hover help on parts palette shows ′Container Icon List′):

1. Add a container icon list to your visual part.
2. Create a script *aboutToOpenProcessing* to add a callback handler to do the background drawing. (See the sample code at the end of this message.)
3. Implement the appropriate callback method *drawBackground*. This method will be called twice, once for the icon image and once for the label.
4. Add a connection from your visual part *aboutToOpenWidget* event to the script *aboutToOpenProcessing*.

You are now ready to test your part; just connect the *items* attribute of the container icon list to the data items you want to display.

Smalltalk

```
aboutToOpenProcessing
| iconListwidget |

″Get the container icon list wigdet (at the Ew* layer)″

iconListwidget : ′Container Icon List1′) widget.

″Set up our own background drawing callback″

iconListwidget applicationDrawnBackground: true.
iconListwidget addCallback: XmNdrawBackgroundCallback receiver:
   self selector: #drawBackground: clientData: nil.
drawBackground: aWidget clientData: clientData callData: callData

| oldColor anRC |

″Only draw the label background in this case″

(callData region == XmLABELREGION)
   ifTrue: [ anRC oldColor anRC setForeground:
      (aWidget colorMap palette nearestPixelValue:
         (CgRGBColor red: 16rFFFF green: 16rFFFF blue: 0)).
      anRC drawable fillRectangle: anRC gc x:
        anRC x y: anRC y width: anRC width height: anRC height.
     anRC setForeground: oldColor ].
```

Deselecting Details Tree Items

Question I have set up a container details tree to display two levels of information. A second list box displays the selected entries from the first. When a parent is expanded, and its child is selected, all works fine. However, if I collapse the parent without deselecting the child, the child is automatically deselected anyway. This does not seem to be the proper behavior upon collapsing its parent. Is there a work-around to keep a child selected once its parent is collapsed?

Answer

The philosophy of the tree views is that the children do not exist in the container unless they are displayed. The reason for this philosophy is that the container may contain a large number of items that have children that have children that have children. To optimize the tree views for performance, we do not hold onto the children and they change from collapse to expand (that is why we request them each time an item is expanded). Because they do not exist for the container they are removed from selection when the item is collapsed.

Smalltalk

```
addNotebookPage: aNotebook named: aName
| page label |
aNotebook subpartNamed: aName
  put: (page := AbtPortableNotebookPageView newPart).
page
  framingSpec: (AbtViewAttachmentConstraint new
  leftEdge: (AbtEdgeConstant new offset: 34);
  topEdge: (AbtEdgeConstant new offset: 90)).
page subpartNamed: ′label′ put: (label := AbtLabelView newPart).
label object: Time now printString;
  framingSpec: (AbtViewAttachmentConstraint new
  leftEdge:
    (AbtRunEdgeAttachmentConstraint new
      attachment: XmATTACHFORM;
      offset: 65);
  rightEdge:
    (AbtRunEdgeAttachmentConstraint new
      attachment: XmATTACHNONE);
  topEdge: (AbtRunEdgeAttachmentConstraint new
    attachment: XmATTACHFORM;
    offset: 95);
  bottomEdge: (AbtRunEdgeAttachmentConstraint new
    attachment: XmATTACHNONE)).
page openWidget.
label openWidget.
```

Resizing Container Columns

Question How can I resize container columns on the fly?

Answer I have resizable container columns by using the following snippet in an initialization method (*columnWidth* is a value that I calculated):

Smalltalk

```
newColumns add: (AbtContainerDetailsColumn newPart
  heading: columnHeading;
  width: columnWidth;
  abtWhen: #cellValueRequested
  perform: (DirectedMessage new
  receiver: self;
  selector: #fillCell:;
  arguments: (Array with: nil)
  );
  parentView: (self subpartNamed: ′Row Container′);
  openWidget;
  yourself).
```

This is the essence of the fillCell: method.

Smalltalk

```
fillCell: callData
″Fill in cell data.″

| columnIndex item |

columnIndex := callData column index.

″columnIndex is the column number″

item := callData item.

″item is the object associated with the row″

callData value: (item someMethod: columnIndex).

″set the cell value by getting some data from the object″
```

Changing Scroll Bar Size

Question I am working on a touch-screen-only application and the scroll bars on the container details part are too small for my fat fingers. Is it possible to change the size so even I can use it?

Answer It appears that the only way to do this is to set *scrollingWithHiddenScrollBars* to *true* on the container details view, create the large push buttons with arrows (as in the multimedia support), and connect those buttons to the different scrolling events on the container details view.

Container Details Tree Children

Question Can you use a derived attribute for the children of the parent in a ContainerDetailsTree?

Answer There is a limitation in container tree view. You need to ensure that the children of a parent are identical (==) each time they are accessed, otherwise the tree will refuse to expand.

In other words, you cannot use a derived attribute for the children of the parent, where there is the possibility that at different times, the children of the parent will not be identical objects. Instead, the first time you get the children of the parent you need to store them in an instance variable of the parent, so the next time the parent is asked for its children, it returns precisely the same children as it did last time. This of course precludes the children from ever changing.

The problem comes about because the tree view checks the children it had last time when it does the expand operation and, if the current children are not the same as the previous children, it assumes that the previous children have been deleted and does not expand them.

Hiding the Heading on a Container Details View

Tip To disable the heading of a container details view, make a connection from the column′s *aboutToOpenWidget* event to the column′s *heading* action. This must be done for every column but it works. It should not be hard to add something to the settings page of the container to encapsulate this behavior.

Disabling a Notebook Tab

Question How do I disable the tab of a notebook?

Answer There are several kinds of notebook types, but one of them offers an ′ *aboutToChangePage*-type event that you can hook up to do the very thing you want to do.

List with Icon Column

Question How do I create a list with an icon column and several detail columns?

Answer You need to use a details-tree container part. First, the column parts should have an attribute to display value specified in their settings view. Second, the objects that you are using as the items should have methods with the names you specified in the attribute to display fields of the column part′s settings. For the column where you want the icon, its Attribute to display method should answer (return) an instance of *CgIcon*. Since the details tree container has multiple columns, you can specify an icon for any of the columns, not just one column and not just the first column. This is why the icon attribute of the visual information requested is not used. This also works with bitmap resources if you return them as pixmaps.

Hiding a Column

Question How do I hide a container details column using the hide action?

Answer Hiding a container details column is not supported even though the actions to hide and show the column are in its public interface. The reason is that the interface actually inherits these actions. We are looking for a way avoid having unsupported features show up in the public interface.

Removing an Icon

Question How come, when you remove an icon from the icons collection of a container, it is still visible in the container view?

Answer It is likely that the container does not get refreshed. There are a couple of solutions to that. You can send a *refreshAll* to the container which will update the view. You can also tear off the *items* attribute and use the following code to do the same:

Smalltalk

```
(self subpartNamed: ′itemsCollection′′)
   performActionNamed: #remove
   with: anItem.
```

Adding Icons Dynamically

Question How do I dynamically add icons to a container and use context menus in different variations?

Answer To learn how to dynamically add icons to a container during run time, instead of statically during development time, see the example in the *VisualAge for Smalltalk User′s Guide and Reference*, pages 229-236. This example allows you to use a context menu while clicking the second mouse button anywhere in he container surface except on top of the icons.

There are other ways to use the context menu. For instance, you may want to be able to bring up the context menu while clicking the second mouse button both on the container surface and on top of the icons. To do this, you just have to modify the following method:

Smalltalk

```
fillContainer: itemsCollection defaultAction: aScriptName

 iconGadget := (AbtIconGadgetView new
 label: each;
 largeIcon: icon;
 parentView: container;
 openWidget;
 menu: ((self subpartNamed: ′Menu′) yourself)   ← add this line
 yourself).
```

By adding that one line, you can now click anywhere inside the container, including on top of the icons, and see a context menu.

Another variation to the use of context menus within a container is to click the second mouse button on top of the icon and see a context menu, but not see a context menu when you click the second mouse button from anywhere else in the container. To do this, edit the same method, except that the same segment of code now looks like this:

Smalltalk

```
iconGadget := (AbtIconGadgetView new
label: each;
largeIcon: icon;
parentView: container;
openWidget;
menu: (((self subpartNamed: ′Menu′) yourself)
  createWidget;
  setPostCreationSettings;
  yourself);
yourself).
```

In addition to changing this piece of code, you also must delete the connection between the container′s menu attribute and the menu′s self attribute.

After these few changes, you should be able to see a context menu by clicking the second mouse button on the icons only, not on the container′s surface.

Refreshing a Container Details View

Question What is the refresh design for a container details view?

Answer You can use the *add:* API or perform the *add:* action. If you perform the action, the model will signal to its views (in this case the details view) that an item was added. The code works this way so you can add an item to the collection without having to refresh the entire list. You can set the *refreshEntireListOnChange* attribute to *true* to cause the entire list to be refreshed with each addition. The default for the containers is to have *refreshEntireListOnChange* set to *false* while the lists′ default is *true*.

Access to Container Object Attribute

Question What event do I need to trigger from so that I can have access to an attribute of an object in a container icon area?

Answer If you are using

Smalltalk

```
(self subpartNamed: ′???′) items addAll: (aCollection)
```

you are dealing with the ordered collection, which holds the items, and you are using their basic protocol. That collection′s protocol does not know anything about event notification, which is VisualAge stuff. When using the second way

Smalltalk

```
(self subpartNamed: ′???′) performActionNamed: #addAll
   with: aCollection.
```

you are dealing with a variable that contains that ordered collection. It comes down to a method

Smalltalk

```
AbtObservableWrapper>>abtPerformActionNamed: actionName
   ifAbsent: aBlock arguments: args
```

Within that method a *signalEvent: #whateverYourActionIs* occurs.

Removing an Icon

Question How do I remove an icon from the container icon view?

Answer Your problem may be that the container does not get refreshed. There are a couple of solutions to that:

1. Send a *refreshAll* to the container.
2. Tear off the items attribute and use something like this:

Smalltalk

```
(self subpartNamed: ′items of blahblah′)
  performActionNamed: #remove with: anItem.
```

That causes a refresh of the container as well. You want to signal that the items have been updated and then anything connected will refresh.

Error: ″Attribute Does Not Exist″

Question In the container details part, what causes an ″attribute does not exist″ error?

Answer Each of the objects in the items collection must have the attribute shown in the column header to get the data to display. If any of them doesn′t, you will get an error.

Adding an Icon to A List Part

Question How do I set up a list part that can show an icon in the first column, followed by several detail columns?

Answer You need to use a details-tree container part. First, the column parts should have an *attribute to display* value specified in their settings view. Second, the objects that you are using as the items should have methods with the names you specified in the *attribute to display* fields of the column part′s settings. For the column that is to display the icon, its *attribute to display* method should answer (return) an instance of CgIcon.

Drag-and-Drop

Question I have developed a drag-and-drop-intensive application under OS/2 that works just fine. When I tried to port it to Windows, I expected it would work like the native style (right-button-drag on OS/2, left-button-drag on Windows) but I see this is not the case—I still have to drag with the right mouse button on Windows. Why?

Answer

The drag-and-drop function built into VisualAge is completely emulated and has nothing to do with the operating system settings. Which button is used internally is completely under Smalltalk control. The default is right-button drag under all operating systems.

If you want to use the left button on Windows to make the user interface on Windows conform to its standard, give this a try:

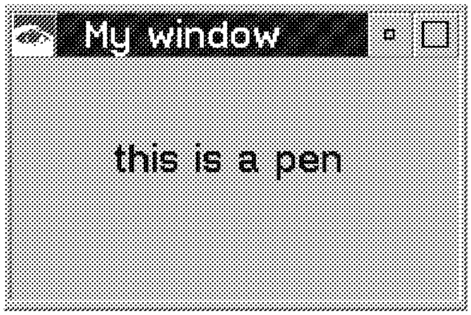
Smalltalk

```
EwDragAndDropManager default button: Button1
```

Multiline Text

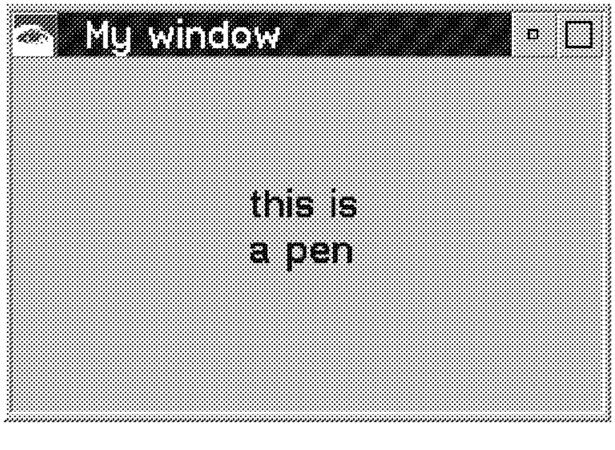
Question

I am developing my GUI via the Composition Editor. I added a text label, but now I want it to be multiline. I have a window like this:



this is a pen

but I want it to look like this:



this is
a pen

Is there a way to achieve this?

Answer Yes. Connect the *labelString* attribute of the label to this script:

Smalltalk

```
label
| cr |

cr := CldtConstants at: ′LineDelimiter′.
^′this is′, cr, ′a pen′.
```

Window Titles

Tip If a window is going to appear in the OS/2 task list, the title is clipped (by OS/2) to 64 characters. When we create a window in OS/2, we automatically add the title to the task list. The only windows whose titles do not get added are dialog windows. However, if you were programming this in C, you would have to add the window title to the task list explicitly. It should be possible to create a window in Smalltalk and not add the title to the task list.

Querying the Caret Position in Text

Question I have an entry field, such as a text edit field, or multiline edit field. I want to know where the caret (the blinking I-beam cursor) is within the text.

Answer The parts you are talking about have the *queryCursorPosition* action. You can use it in a script or through a visual connection. It is described in the *VisualAge for Smalltalk User′s Reference*.

Color Bitmap Resolution

Question How can I improve the resolution on 256 color bitmaps in Windows?

Answer Basically, you have to set the palette of the window to be equal to the palette of the bitmap. To do this, connect the *openedWidget* event of your window to a script like the following:

Smalltalk

```
  fixColors
    ″Ask the image for its palette and
     set the window to have the same palette.″

  | imageDesc |

  imageDesc := (self subpartNamed: ′Form1′)
      backgroundGraphicsDescriptor.
  (self subpartNamed: ′Form1′) primaryWidget shell window setPalette:
     imageDesc image palette.
  (self subpartNamed: ′Form1′) backgroundGraphicsDescriptor:
imageDesc.
```

Note: This example assumes that you are setting the background of a form.

Automatic Tab Moving

Question How do I implement automatic tab moving in a VisualAge text field?

Answer You can implement auto-tabbing in VisualAge by writing some script and making a connection to that script. Here is an example: Suppose you have two text fields. Go into the settings of TextField1 and set its text limit to 5. Then, connect the *object* event (make sure it is the event) of the text entry field to the following script:

Smalltalk

```
  autoTab

  | position limit |

  position := ((self subpartNamed: ′TextField1′) queryCursorPosition).
  limit := (self subpartNamed: ′TextField1′) textLimit.
  (position = limit)
     ifTrue: [ (self subpartNamed: ′TextField2′) setFocus ].
```

Now, if the user types five characters in the first text field, focus is automatically set to the second text field.

Question Is there a way to have it just tab to the next field? (I have around 10 fields and would rather avoid hard-coding all the names.)

Answer Use some code such as:

Smalltalk

```
| widget list index next |

widget := (self subpartNamed: ′Window′) framingWidget.
list := widget tabList.
index := list indexOf: ((self subpartNamed: field name)
   primaryWidget).
next := list at: (index + 1).
(next userData) setFocus
```

We hooked this to a text field and passed in the self of the text field as the parameter.

This code is given *as-is*. That is, it has some drawbacks. First, when you get to the end, you will have to know the number to reset it to. We don′t have that check in the code above. And, when it gets to a group of push buttons, I′m not sure of the behavior (it may just go to the first in a group). This is a good start, however, and the concerns mentioned can be checked for in a script.

User Input Error

Question How can I get the *userInputConvertError* event to fire if the user does not change an invalid field?

Answer You must modify the method *AbtTextConverterManager>>#primCommitUserInput:* to handle this or run this script (using aDateString as an example) when losing focus:

Smalltalk

```
(AbtDateConverter new) primDisplayToObject: aDateString
  ifError: [ ^error ]
CwAppContext default asyncExecInUI: aView setFocus.
```

Closing the Window

Question When I press the Esc key, the window closes. What event is triggered by pressing the Esc key? Can I capture this event to alter the behavior?

Answer Yes. Connect the shell′s *closeWidgetRequest* event to a script like this:

Smalltalk

```
myShellCloseWidgetRequest: aConfirmationCallbackData

  aConfirmationCallbackData doit: false.
```

The window won′t close as long as *doit:* is set to *false*. When you use *openOwnedWidget* as opposed to *openWidget*, you are actually creating a dialog box as opposed to a standard frame window. It is the dialog box that provides the default behavior of closing when the Esc key is pressed (and this is operating system behavior, not part of VisualAge).

The solution above hooks to the system *close* action. Since you are bringing up a dialog, the Esc key is mapped to *close* by the CwDialogShell class (for details, see AbtShellView class>>cwModalWidgetClass and AbtShellView>>primCreateWidget). If you do not set an owner window, the CwTopLevelShell is used as the shell class (see AbtShellView class>>cwWidgetClass).

You could map the Esc key (or any key you prefer) to close. Here′s an example. First, connect the Window part′s *openedWidget* event to a script:

Smalltalk

```
hookKeyReleaseEvent
  (self subpartNamed: ′Window′) components do: {:view |
      view primaryWidget
        addEventHandler: KeyReleaseMask
          receiver: self
          selector: #keyReleased:clientData:callData:
          clientData: nil}
```

This puts a key release hook on every widget under Window. Next, add a method to handle the key release:

Smalltalk

```
keyReleased: w clientData: d callData: callData
  (callData keysym == XKCancel) ifTrue: {self closeWidgetCommand}.
```

Note: XKCancel is the Motif name for the Esc key.

Transparent Background

Question How can I make the background of some text placed on top of a bitmap be transparent?

Answer CwLabels cannot be made transparent. I suggest you hook the expose callback of the form and draw the text yourself.

Getting the Anchor Block Handle

Question How do I get the anchor block handle of a primary visual part?

Answer Here is an example of how you can get the anchor block handle of your primary visual part:

Smalltalk

```
 (self subpartNamed:  Window’) yourself primaryWidget
     osWidget handle winQueryAnchorBlock
```

Turning Off Error Message

Question How do I turn off the error message *** error *** that is shown by data converters such as AbtStringConverter on text parts?

Answer Turn off the setting *reformatOnFocusChange* for each of the text fields you wish to affect. You can do this in the settings notebook.

Locating the Focus

Question How can I determine which form contains the widget that has the focus?

Answer The following code shows how you can determine which widget has the focus:

Smalltalk

```
 queryFocusForm

 | focusWidget focusView |

  focusWidget := self shell focusWidget.

  (focusWidget notNil and:
      [ (focusView := focusWidget userData) isKindOf: AbtBasicView])
  ifTrue: [
     [ focusView notNil and: [ focusView isKindOf: AbtFormView ] ]
      whileFalse: [ focusView := focusView parentView ] ].
^focusView
```

Note that *userData* of the widget of AbtBasicView subclasses refer back to their view. You could accomplish the same thing by iterating the target form components and comparing their widget against the focus widget, but the above is quicker.

Bitmaps on Buttons

Question Is there a way to drop a bitmap (BMP) on a button visually?

Answer Yes. Use the following code:

Smalltalk

```
myPushButtonView graphicsDescriptor:
   (AbtImageDescriptor new moduleName: ’bitmap.bmp’)
```

Progress Indicator Window

Question I would like to use a progress-indicator window in my application. I can′t seem to find a ready part for this. CwWorkingDialog seems to be the base part, but it does not provide the *percentage completed* support. The EtWindow class seems to offer all support, but I am confused as to how to get it started. What would be the recommended way for implementing a progress indicator window?

Answer The following example shows how you can use the CwProgressDialog to create a progress indicator:

Smalltalk

```
| shell main form dialog fraction |

shell := CwTopLevelShell createApplicationShell: ′shell′
  argBlock: [ :w | w title: ′Main′ ].
main := shell createMainWindow: ′main′
  argBlock: [ :w | w width: 400; height: 300].
form := main createForm: ′mainForm′ argBlock: nil.

form manageChild.
main manageChild.
shell realizeWidget.

dialog := CwProgressDialog for: main.
dialog mwmInputMode: MWMINPUTFULLAPPLICATIONMODAL.
dialog open.

[[ (fraction := dialog fractionComplete) < 1 ] whileTrue:
   [ (Delay forSeconds: 1) wait.
   fraction := fraction + (1 / 10).
   dialog fractionComplete: fraction ].
dialog close ] fork.
```

Hot Spots

Question If I try to set the *borderWidth* attribute for a hot spot, either from the *General* page for the part or from code, I see these settings when I query the value. However, what is the reason that I won′t be able to actually *see* the border?

Answer You can′t see the hot spot because it is never mapped. It is generally used as a clickable spot for images, hence it does not draw any border.

Question How can I create hot spots that can be reshaped?

Answer Create a subclass of AbtHotSpot, overriding the *containsPoint:* method. Create a subclass of AbtHotSpotView and override *postCreationInitialization* to use your AbtHotSpot subclass. Hot spots can only be rectangular. However, if you need a nonrectangular shape, you can try using multiple smaller hot spots that are next to each other to approximately cover the area you need.

Question How do I find the coordinates of a hot spot?

Answer To get the x and y coordinates of the hot spot, send it the message *x* or *y.* To get its width and height, send *width* or *height.* This is true for all parts.

Links

Question What is a link in the world of drag-and-drop?

Answer Link means that the instances dropped get added to the target and remain in the source. *Copy* duplicated the instances and *move* deletes them in the source.

Wallpaper

Question How do I display an image as a wallpaper with more than 16 colors?

Answer Here′s a complete script you can use and connect to the *openedWidget* event of the window:

Smalltalk

```
showWallpaper
| anImageDescriptor |

anImageDescriptor := AbtImageDescriptor new
  moduleName: ′ < d:\myPath\imageFileName>′;
  yourself.
(self subpartNamed: ′Window′) primaryWidget shell window setPalette:
  anImageDescriptor image palette.
(self subpartNamed: ′Window′) backgroundGraphicsDescriptor:
  anImageDescriptor.
```

In the *(self subpartNamed: ′Window′*) above, *′Window′* should be replaced with the name of the part where you want to specify the wallpaper.

Question Is it true that the wallpaper of a window part under Windows 3.1 can represent only 16 colors?

Answer Under Windows, by default, the color palette of the image is mapped to the closest values of the system palette. Typically, the system palette does not have many of the colors found in most bitmaps, so the mapping is poor. To get the image to appear the way you want, add the following code snippet just before you display your image.

Smalltalk

```
<myWindow> primaryWidget shell window setPalette:
  <myImageDescriptor> image palette
```

This code snippet must be executed after <myWindow> has been opened.

Labels on Message Prompter

Question Can I change the labels on an AbtMessagePrompter?

Answer No. AbtMessagePrompter uses CwMessagePrompter, which uses the native operating system API, such as WinMessageBox under OS/2, and that does not support button string substitution. I suggest creating your own view with your message, then displaying it as a dialog, like this:

Smalltalk

```
myView openApplicationModalWidget suspendExecutionUntilRemoved.
  (myView result == ...) ifTrue: ...
```

The *result* method is implemented by your view and is the result of the user′s selection, probably saved in an instance variable.

Visual Part Reuse

Question Can I reuse my visual parts?

Answer Yes, by creating reusable forms. Create a new visual part, then select and delete the default window. Add a Form and make it the primary part. Lay out your GUI design and add in any functionality you like. Save the part and add it to the palette for everyone to reuse! You can also reuse menus.

Supported Image Formats

Question What are the image formats that VisualAge for Smalltalk supports?

Answer Individual bitmap files and icon resources in DLLs work for push button images and for wallpaper. You can also have individual graphic files. Open the settings on the button and choose label type Graphic, graphic type Image and enter the filename (should be path relative to your VisualAge directory). Enter a text label first if you are using hover help, since the text for the button is what appears.

Click-Sensitive Bitmap

Question How do I display a bitmap and then sense mouse clicks over it?

Answer Look at the Graphic Link example (AbtGraphicLinkView) in the VisualAge Help Example feature. It uses Smalltalk code to track mouse clicks and movements over a bitmap. For your application, you might put hot spot parts on the bitmap.

Setting Help

Question How do I programatically set the help file and help topic IDs for the menu items?

Answer I assume you want to use a symbolic name that will be mapped to an actual name at execution time. This is handy for dealing with files whose names include version information. For example, ABTVAE30.OS2 contains such a mapping entry. The entry *abtvahelp* is the symbolic entry we use in the settings view as the help file name. It is mapped to an actual file name of ABTVAE30 at execution time. When we updated the file for Version 4 to ABTVAE40, we simply updated the one entry in the mapping file and did not have to touch any code. The entries in a given mapping file do not support automatic substitution. The design is such that the name of the mapping file can vary by platform (for example ABTVAE30.OS2 is the English version while ABTVAJ30.OS2 would be Japanese) and these mapping files would map the symbolic name to the appropriate locale-specific name. Your application would then register the mapping file with the locale character as a wild card. You can see an example of this in the method AbtEditBaseApp class>>#loaded. Create locale-specific versions of the mapping file where, within each mapping file, a consistent symbolic name maps to the locale-specific actual (absolute) file name and then register the mapping file name with the locale character as a wild card.

Modifying Data Entry Parts

Question Is there a *userModified* event for data entry parts?

Answer The available events are *losing focus*, which occurs whether the field value has changed or not, and *object*, which is triggered for any changes including user and programmatic changes.

Reacting to Keyboard Input

Question How do I receive keyboard input and react to it?

Answer You can set up an event handler to handle key press events and check for the F4 there. See page 188 of the *IBM Smalltalk Programmer′s Reference* for a description and example of this. Add an event handler to your window:

Smalltalk

```
(self subpartNamed: ′Window′) primaryWidget
addEventHandler: KeyPressMask
receiver: self
selector: #keyEventHandler:clientData:event:
clientData: nil.
```

Then you handle the event like this:

Smalltalk

```
keyEventHandler: widget clientData: clientData event: event
Transcript show: event keysym printString
```

Using Esc to Close Nonmodal Dialog

Question Can I use Esc to close a nonmodal dialog?

Answer Since you are bringing up a dialog, the Esc key is mapped to close by the CwDialogShell class (see AbtShellView class>>cwModalWidgetClass and AbtShellView>>primCreateWidget for details). If you do not set an owner window, the CwTopLevelShell is used as the shell class (see AbtShellView class>>cwWidgetClass).

You could map the Esc key (or any key you prefer) yourself to close. Here′s an example. First, connect the Window part′s *openedWidget* event to a script:

Smalltalk

```
hookKeyReleaseEvent

  (self subpartNamed: ′Window′) components do:
      [ :view | view primaryWidget
        addEventHandler: KeyReleaseMask
          receiver: self
          selector: #keyReleased:clientData:callData:
          clientData: nil ]
```

This puts a key release hook on every widget under *Window*. Next, add a method to handle the key release:

Smalltalk

```
keyReleased: w clientData: d callData: callData

  (callData keysym == XKCancel) ifTrue: [self closeWidgetCommand].
```

Note: *XKCancel* is the Motif name for the Esc key.

Multiline Edit Part and Tab Order

Question If I drop a multiline edit part on a window, I can′t use the tab key to navigate through the different objects on the window. The multiline edit part uses the tab key for spacing. Is there another way to navigate through the window?

Answer You can use Ctrl+Tab instead to navigate through the different objects on a window.

Using Predefined Function Keys

Question Why does the following code not work to catch the event when pressing a function key? The code adds an *eventHandler* to the widget of the part and should cause a debugger window to come up as soon as an function key is pressed.

Smalltalk

```
| widget |
widget := (self subpartNamed: ′myForm′) primaryWidget.
widget addEventHandler: KeyPressMask
      receiver: self
      selector: #keyPressed:clientData:event:
      clientData: widget.
```

The event handler looks like this:

Smalltalk

```
keyPressed: widget clientData: clientData event: event
| key |
key := event keysym.

( key = XF10 | key = XF22 ) ifTrue: self halt.
( key = XF9 | key = XF21 ) ifTrue: self halt.
```

Answer The above code is correct, but you should take into account that some of the function keys have been reserved for the system. For example, PF10 on a window (with a menu bar) will move the cursor to the first menu bar item. Hence the debugger window will not come up in the above example.

Grabbing Typed Words

Question When using a multiline edit (MLE) window, I would like to grab the words as they are typed. The only event I saw that looked like it might be of use was the notify on each key stroke. If I use that, I′d have to look for a blank and then post a signal that a word has been completed. Am I missing something? Is there a mechanism already there?

Answer No you are not missing something. The MLE does not parse its own text. If you are interested in dropping to the widget layer, you can also check out the *XmNvalueChangedCallback* and *XmNmodifyVerifyCallback*. The *XmNmodifyVerifyCallback* is fired before the text shows up when the user has typed a character. The *XmNvalueChangedCallback* is fired after the text has been displayed in the MLE.

Dropping an Item on a Push Button

Question How can I use the drag-and-drop operation between one item of a list box and a graphical push button representing a trash can? It seems that when I drop an item of the list box on the graphical button, my icon is replaced by the label of the item.

Answer What is described above is the default behavior. However, all of the drag-and-drop events are public, so you can do whatever you want. The way to get the graphical push button to remain graphical is to connect the *dropped* event of the push button to a script that accepts a parameter. The parameter will be an instance of EwDropCallbackData. It contains information on what you just dropped on the push button as well as other things. Be aware, though, if you connect from any of these drag-and-drop events, you become responsible for what happens in these cases.

Changing a Part′s Settings View

Question I created a new TextView part inherited from AbtTextView and extend it with a new attribute *dataRequired* of type Boolean to see if this field must include data or not.

Now I want to add a toggle button for this attribute to the attributes page in the part′s settings notebook. I found a class AbtTextViewAttributesPage in the subapplication AbtEditViewsSubApp which probably defines this attributes page. In opening this view part, I do not see anything except some connections that point to nowhere. In testing the view part I get the attributes page. Why do I get this behavior?

Answer It is a little strange, but you are not seeing anything because the form has no size when you edit the settings page. It has no attachments so it tight-wraps around its children. Here′s what you can do:

1. Select **View Parts List...** to bring up its settings and change its size to nonzero.
2. Modify the view as you want.
3. Generate archival code and remove the code that sets the form size.
4. Regenerate the run-time code (see the *test* method in the archival code for details).

Steps (3) and (4) are necessary because the Composition Editor does not provide a way to specify that the form is an unattached form (what is called a *shrink-wrap* form).

Hiding Table Columns Dynamically

Tip If you want to hide one or more columns of a table, here is what you have to do. Unfortunately, no action is available that lets you hide a column. You can, however, hide the entire table. As a work-around, you can simulate the wanted behavior by manipulating the *ratioWidth* of the column: Setting the value to zero will make it look as though the column has been hidden.

Initializing a View Wrapper

Question When I use a view wrapper, the widget does not get created until I send it the *openWidget* request. Therefore, I can′t do anything (for example, set some of its features) until after the first time I′ve issued *openWidget* which in many cases is not desirable. I got around this by the following code:

Smalltalk

```
((self subpartNamed: ′MyWindow′) value) isNil
ifTrue: [
   (self subpartNamed: ′MyWindow′) createValue].
```

However, both *value* and *createValue* selectors are private. How can I solve this problem without resorting to the use of private methods?

Answer You should be able to set feature values via attribute-to-attribute connections. Although the connections will not actually do anything until the view is created, once it is, all the connections are aligned.

Adding Columns to Tables

Question How can I dynamically add columns to a table?

Answer Here is some code that will give you the ability to add columns dynamically.

Smalltalk

```
AbtTableView>>addLastColumnNamed: aColName
   columnHeader: aColumnHeader
   attributeName: anAttributeName
   readOnly: aBoolReadOnly
   converter: aConverter

   ″Public - This is a helper function which will create a new column
     view and add it to the table.  The column will be added to the end
     of the table.  Its width will be equal to the width of the most
     narrow column currently in the table.″

   | cv |
   (cv := AbtTableColumnView new)
      attributeName: anAttributeName;
      columnHeader: aColumnHeader;
      readOnly: aBoolReadOnly;
      ratioWidth: self suggestedNewColumnWidth;
      converter: aConverter.
   self subpartNamed: aColName put: cv.
   cv openWidget.
   ^true
```

Smalltalk

```
AbtTableView>>subpartNamed: aColName put: aColumnView

   | colNdx |
   (self components includesKey: aColName)
       ifTrue: [ ^false ].
   colNdx := self components size + 1.
   self subpartNamed: aColName  put: aColumnView partIndex: colNdx.
```

Smalltalk

```
AbtTableView>>suggestedNewColumnWidth

    ″Use this method when adding new columns dynamically at runtime
     to prevent the new column from being too small.″

   | ans |
   self components size = 0
      ifTrue: [ ^1 ].
   ans := 9999999999.
   self components
      do: [ :col | ans := ans min: col ratioWidth ].
   ^ans
```

Smalltalk

```
AbtTableColumnView>>openWidget

″Public.  Category: AbtRun-Internal″

  | pv |
  (pv := self parentView) == nil ifTrue: [ ^self ].
  pv setUpWidget.
```

Synchronize Table Scrolling

Question Is there any way of synchronizing the scrolling of two tables in the same window? I would like to move the vertical scroll bar of one table and have the scroll bar of the other table move in the same way.

Answer To synchronize the scrolling of two tables, you will have to use callbacks. The slider bar is a CwScrollBar widget. Callbacks can be specified for the various slider requests (for example, moving the slider, clicking on the arrows). Each callback method will have to scroll the second table. For example, suppose you set up an increment callback for table A. The method specified for this callback gets called every time the user clicks on the slider bar down arrow. The callback method will have to scroll the second table down one row.

To synchronizing scrolling between two tables, you would need to add callbacks for each type of scroll request. In addition, you would need to add them for both CwScrollBar widgets.

Changing Table Size

Question How do I create a table that can grow and shrink in size?

Answer Create an example of what you want, save that part, then look at the *abtBuildInternals* method. It shows the code that constructed the part. You can copy or modify it to dynamically add parts. A common mistake is to forget to send *openWidget* to the newly added part (including container detail columns)—it is necessary if the view is already open.

Table List Part

Question Is there a *table list* part that has capabilities like row selection and programmatic column setting?

Answer EwTableListExample fullfills these requirements. EwTableList is the CommonWidget class used by the Container Details View available on the parts palette.

Reusable Table Part

Question How do I create a reusable table part?

Answer You should promote the width attribute of each column so it becomes an attribute of the reusable part. They should be able to change the width through the settings of the reusable part or via a script. The width cannot be changed with the mouse because embedded parts are treated as singular primitive objects.

Changing a Table Cell

Question How do I keep track of changes made to a cell in a table?

Answer It may be best to track this at the model level, not the view. That is, register interest in your model for the events that are important to you. For example, *self* for an ordered collection dropped on the free form surface will gives you notification everytime the collection changes. Presumably your model is more interesting than an ordered collection (person, employee, company, whatever), so you should register interest in whatever event is important to you (based on the public interface you defined).

Selecting Multiple Rows

Question How do I select multiple rows in a table part?

Answer You can do this using the Container Details View. You just go into the settings and specify *Multiple* as the selection type.

Changing Push Button Activation

Question How can I change push-button enablement when text is entered in a textbox?

Answer There are several ways to go about doing this depending on what you′d like to happen. Here are two examples:

The first example starts off with a text box and a push button that′s initialized as disabled. You can disable a push button initially by going into the settings for the push button and removing the check mark next to the enabled box. Disable the push button before making the following connection. Connect the object event of the text box to the enabled action of the push button. Then go into the settings of the connection, select the set parameters button, and set the value to *true* (check the value box). This causes the push button to be disabled until any text is entered in the text box. The push button then remains enabled. If you want the push button to again become disabled for whatever reason, you would have to write script for that. Which leads to the next example...

The second example starts off exactly like the first example. The view contains a textbox and a disabled push button. This push button will remain disabled until text is entered into the text box and the text box loses focus (as when you tab out). As soon as the text box loses focus with text entered into it, the push button becomes enabled. If you then go back to the text box, delete the text, and tab out of it, then the push button becomes disabled again and will remain disabled until text is entered into the text box. This is accomplished by writing a small script that looks something like this:

Smalltalk

```
enableButton

  | info |

  info := ((self subpartNamed: ′Text′) string).

  (info isEmpty)
  ifTrue: [(self subpartNamed: ′Push Button′) enabled: false]
  ifFalse: [(self subpartNamed: ′Push Button′) enabled: true].
```

You would connect the losingFocus event of the text box to this script.

Event Handling in Windows, AIX, and OS/2

Tip

OS/2 does not work the same as Windows and AIX when it comes to event handlers. OS/2 propagates messages up the parent tree. For example, if I have a main window with three text fields, a drop- down list, and a spin button, and I add an event handler to detect key presses to the main window, the child widgets also receive those messages and handle those key press events (on OS/2). However, on Windows and AIX this is not the case; the messages are not propagated to all children. To get around this, you can write Windows- or AIX-specific code to send the message *recursiveDo:* to your main window and set up the same event handler or callback for all the children, the children′s children, and so forth. You probably create a resource hog to do this, but its really the only way to achieve this effect on Windows or AIX.

Note: Using *recursiveDo:* will not help you with widgets (controls) that are composites at the operating system level. For example, this will not solve the problem on Windows where you can′t get pointer motion notification over certain parts of the combo box control. From Smalltalk′s perspective, there is only one view (and hence nothing to recurse) while there are actually two to three window handles at the operating system level.

Part Validation

Question How do I execute an error script every time a user leaves a part to be validated?

Answer You must modify the method *primCommitUserInput:* in AbtTextConverterManager to handle this or run this script (using aDateString as an example) when losing focus:

Smalltalk

```
(AbtDateConverter new) primDisplayToObject: aDateString
ifError: [ :error ]
  CwAppContext default asyncExecInUI: aView setFocus.
```

Moving Icons

Question How do I move icons around in a window?

Answer If the icons were drawn as a label widget, you would use *CwWidget>>#makeGeometryRequest:replyReturn:* if the label is an attached widget, otherwise *CwWidget>>#moveWidget:y:* (see the comments of these methods for more details). Moving a widget to simulate a spinning effect, for example, probably wouldn′t look very smooth. I′d probably go directly to the Cg layer, using a variation of the code in the CgClock examples (for drawing the wheel points) and perhaps CgStix for animating the spinning of the arrow.

Printing a Form with a Visual Part

Question How do I print a form I created that contains a visual part?

Answer You create a CwPrinterShell, initialize the shell on the map callback, and draw your text during the expose callback. The printer shell has additional APIs to support job and page start and end. You can set fonts and colors by modifying the attributes of the graphics context (GC) for the printer shell. CwTextPrintingManager uses CommonPrinting to print workspaces; you can see the basics there.

Text Box Problem

Question I have a problem using the *defaultActionRequested* event of a text box. How do I fix it?

Answer This is a known problem. You′ll need to change the method *postCreationInitialization* in AbtTextView. Locate the following piece of code in the method and remove it:

Smalltalk

```
widget addCallback: XmNactivateCallback
receiver: self
selector: #defaultAction:clientData:callData:
clientData: nil.
```

Reusable Table Part

Question How do I create a reusable table part with different widths?

Answer Promote the width attribute of each column so it becomes an attribute of the reusable part. You should be able to change the width through the settings of the reusable part or via a script. The width cannot be changed with the mouse because embedded parts are treated as singular primitive objects.

Trap the Enter Key for Tabbing

Question I want to add a callback (XmNactivateCallback) to a toggle button so that I can trap the Enter key to do the tabbing. Would this be the right way to do this?

Answer To do this, use an event handler rather than trying to add a callback not supported by the widget. The following example sets up four toggle buttons in a view. Run the following script when the *openedWidget:* event fires:

Smalltalk

```
setUpEventHandlers
  | tog1 tog2 tog3 tog4 |

  tog1 := (self subpartNamed: ′Toggle Button1′) yourself.
  (tog1 primaryWidget)
     addEventHandler: KeyPressMask
     receiver: self
     selector: #tabTo2:clientData:event:
     clientData: nil.

  tog2 := (self subpartNamed: ′Toggle Button2′) yourself.
  (tog2 primaryWidget)
     addEventHandler: KeyPressMask
     receiver: self
     selector: #tabTo3:clientData:event:
     clientData: nil.

  tog3 := (self subpartNamed: ′Toggle Button3′) yourself.
  (tog3 primaryWidget)
     addEventHandler: KeyPressMask
     receiver: self
     selector: #tabTo4:clientData:event:
     clientData: nil.

  tog4 := (self subpartNamed: ′Toggle Button4′) yourself.
  (tog4 primaryWidget)
     addEventHandler: KeyPressMask
     receiver: self
     selector: #tabTo1:clientData:event:
     clientData: nil.

  tabTo1: widget clientData: clientData event: event

  event type = KeyPress
  ifTrue: [ (event keysym == XKReturn)
      ifTrue: [ (self subpartNamed: ′Toggle Button1′) setFocus] ].

  tabTo2: widget clientData: clientData event: event

  ... and so on ...
```

Creating an Operation-in-Progress Window

Question I would like to create an In-Progress window that gets displayed while the application is off querying a database. I want to block the user interface so the users won′t be able to start queuing up a lot of database queries, which leads to a cache resource error. I have tried opening an in-progress window as a full application modal but I am not satisfied with the ability to hit the Esc key and exit out of the in-progress window. Is there another way of doing this?

Answer The VisualAge for Smalltalk processing window is achieved by:

Smalltalk

```
execLongOperation: [ aBlock with your long running stuff ]
   message: ′I am thinking, please wait...′.
```

But to be honest, this is only half the story, because the message is implemented by AbtShellView. That means you have to send it to the current window. That in turn leaves the problem how to find the *actual* window (for example, the one having the focus). You could use something like this:

Smalltalk

```
CwTopLevelShell allShells reverseDo: [ :sh |
   sh hasFocus ifTrue: [ ^ sh userData ].
].
```

This method returns the window having the focus and accepts the *execLongOperation:message:* message. If you don′t pick the window having the focus, the focus in your application changes to the window you reference. This is quite confusing for users— at least, so they tell me. The above examples work in Version 3, and have not been tested in any other version.

Progress Messages

Question How are the progress messages in the browsers implemented?

Answer The CwProgressDialog implements the *in-progress* indicator (no buttons, just a message and a bar).

Limiting the Number of Lines in Multiple-Line Editor

Question We need to limit user input to 30 lines with 60 characters per line. Is there a way to do this using a multiline editor part? I see only the ability to specify 1800 (30 * 60) characters as the maximum number for the multiline editor part. This won′t do because the user could type 1800 new-line characters, much more than the 30 lines maximum we want. If a multiline editor part is not usable, any suggestions?

Answer You are correct, there is no way to set the number of lines a user can enter into a multiline editor part, only the total amount of text. You could, however, write a script that does this type of checking and formatting. You would need to invoke the script every time the user presses a key. The script could determine the number of lines in the multi-line editor part by keeping a count of the number of line delimiters entered in the multiline editor part′s text. The script could also force a new line if the user has typed in 60 characters but failed to hit the new-line key.

Disabling a Button

Question Suppose you have a text entry part and an add button. How can you disable the add button whenever there is no text in the text entry part?

Answer If the text entry part is connected to the selected item in the list part, you can connect the button′s *enabled* attribute to the *selectionIsValid* attribute of the list.

If the button is independent of the list, you have to write a small script and connect a *string* event—string in the text entry part has been changed—to the script. You should probably set the button to be disabled initially (from the settings notebook for the button). The sort of script you need is:

Smalltalk

```
setButtonForString: aString
  ″Disable the button if the string is empty, otherwise enable it.″

   (self subpartNamed:  Button′) enabled: (aString notEmpty)
```

Accelerator Keys

Question I am trying to use accelerator keys for some menus that I would like to have in my application. Is it possible to use accelerator keys for menu items?

Answer Defining accelerator keys is not supported for menu items in pop-up menus. They are supported only for menu items in the menu bar. The underlying reason is because pop-up menus are not created until the user clicks the right mouse button, and this would be when the accelerator table gets created. The accelerator table then gets destroyed when the pop-up menu goes away. In effect, if you have defined accelerator keys for a menu item in a pop-up menu, and hit for example Ctrl-V, the menu has not been created so the event does not get signaled. If it would be appropriate in your application, you could design duplicate menus in your menu bar with (accelerator keys defined) for each pop-up menu you have. That is how we have dealt with this in the development environment.

Hover-Help on Widgets

Question Is it possible to get hover-help for push buttons, entry fields, and table cells as well as simple icons? The documentation implies hover-help works for icons only.

Answer The underlying support for hover-help is implemented as an extended widget and can be used for all widgets that support mouse move callbacks. Thus, hover-help does work for other than icon push buttons. Working with icon push buttons is the only case where, by default, a hover-help tag is displayed. Entry fields, for example, have no string resource to use as a default hover-help tag. In these cases, you need to hook the pop-up callback and provide the hover-help string to be displayed in the callback data.

The widgets not supported for hover-help are those that do not signal mouse moves; such widgets are platform dependent. For example, on some platforms drop-down list boxes *do* signal mouse moves when the pointer is over the drop-down button, but *do not* signal mouse moves when the pointer is over the text portion of the drop-down list. On those platforms, hover-help is not supported for drop-down lists. There currently is no list of unsupported widgets, since whether the widgets signal mouse moves depends entirely on the operating system′s implementation of the control.

Improving Performance

Tip

In order to improve performance with our application (which uses many complex notebook objects) and to help with memory utilization, we implemented an approach that hides existing visual objects rather than the *closeWidget* approach. When the application needs to open a notebook, we iterate through existing instances of the visual class and *show* them rather than issuing an *openWidget* event.

This has had a dramatic performance improvement in our application, making it perform as well as or even better than some C++ applications which may display many complex notebook objects. The notebook snaps open and clicking on the notebook pages is immediate.

Tabbing Order

Question

Is there an explanation somewhere of how to set tabbing order in a variety of situations? The simple case is straightforward and one can happily drag the yellow tags around to change the order. But what do the blue tags mean and how are they manipulated? Also, how can you create different traversal groups for separate groups of buttons?

Suppose you have a form which contains a set of buttons plus another group of buttons directly placed on the window. The yellow tags show ′3:1′ and ′3:2′ on the directly placed buttons but there is a blue tag showing ′3′ on the buttons form. The form gets tabbed to first when you want the other buttons to be tabbed to first. Is there a way to change this?

Answer

When you see 1, 2, 3 ... (whether it′s blue or yellow), that means it is a tab group and you use the tab key to navigate to or from it. In this case, yellow indicates that it is a tab group consisting of one part only. An example would be an entry field. Blue indicates that it is also a tab group, but that it has multiple parts within it. An example would be the form with multiple push buttons in it.

You typically want to navigate between push buttons using the arrow keys, so when you place push buttons directly on the window, it defaults to traversal only. Consequently, you see numbers like 1.1, 1.2, 1.3, ... It′s acting as though it′s within a tab group, and will always be added as 1.x.

Since you can change the tab order of parts at the same level only, you have two options. Do not put the push buttons directly onto the window, put them in a form as with the other push buttons. Or, check the tab group option on the individual push buttons. You will have to use the tab key to navigate to and from these push buttons if you choose the latter option.

Ctrl-Click Events

Question I can set individual button press (ButtonPressMask) and key press (KeyPressMask) event handlers. But how can I detect if a key was pressed at the same time a mouse button was pressed? I am trying to differentiate between a mouse button-Ctrl key event and just a mouse button pressed event.

Answer You will have to process the keyboard key and mouse button events individually. A suggestion is to set a flag when the mouse button has been pressed (but not released). When you process the keyboard key event, check to see if the mouse button flag has been set. If it has and the Ctrl key was pressed, execute your mouse button+Ctrl code. You will need to clear the mouse button flag whenever the mouse button is released.

Here′s another, probably easier way, of doing the same. The method that processes the keyboard key event can check the event state. If the state is Button1Mask and the key pressed is Ctrl, execute your mouse button+Ctrl code. Here is what the code would look like:

Smalltalk

```
(event state = Button1Mask) & (event keysym = XKControlL)

  ... mouse button+Ctrl logic
```

Creating Push Buttons

Question How do I create push buttons on the fly?

Answer You have two different approaches:

1. You do it in pure Smalltalk: starting from the parent part, such as an AbtFormView, you do:

   Smalltalk

   ```
   (<part> widget createButton: ′myButton′ argBlock: nil)
                                manageChild.
   ```

   In the *argBlock* you could specify the widget attachments The disadvantage is that this will leave you without an AbtButtonView part, which may be inconvenient.

2. The other method is adapted from the *abtBuildInternals* method:

   Smalltalk

   ```
   <part> subpartNamed: ′button′ put:
      (tVar:= AbtButtonView newPart).
   tVar openWidget.
   ```

   The *openWidget* will actually draw the button. To position it properly you might add a framingSpec (look into *abtBuildInternals* of a visual part for an idea of how to do that).

   If that does not satisfy you, add the maximum amount of buttons in the Composition Editor and use the *hide/show* protocol to dynamically hide and show your buttons. For buttons you need initially invisible, you send a *hide* attached to the *aboutToOpenWidget* event of your window, so they won′t show.

Changing Part Labels

Question How do I change the labels on the AbtMessagePrompter part?

Answer You can′t because AbtMessagePrompter uses CwMessagePrompter which uses the native operating system API (for example, WinMessageBox under OS/2) and that doesn′t support button string substitution. I suggest creating your own view with your message, then displaying it as a dialog, such as

Smalltalk

```
myView openApplicationModalWidget suspendExecutionUntilRemoved.
  (myView result == ...) ifTrue: ...
```

The *result* method is implemented by your view and is the result of the user′s selection, probably saved in an instance variable.

Adding BMP Files to Buttons

Question Does VisualAge support putting BMP files directly onto buttons?

Answer Yes, since VisualAge for Smalltalk Version 3. Good-bye one DLL; hello dozens of .BMP files. Use the following code:

Smalltalk

```
myPushButtonView graphicsDescriptor:
    (AbtImageDescriptor new moduleName: ′bitmap.bmp′)
```

Creating a Prompter

Question How do I create my own prompter for a GUI?

Answer Add an instance variable *result* to your view, and implement a *prompt* method:

Smalltalk

```
prompt

  result := nil.
  self suspendExecutionUntilRemoved.
  ^result
```

You would open the view like this:

Smalltalk

```
(view := MyView new) openApplicationModalWidget
myResult := view prompt.
```

Setting Focus to Next Entry Field

Question How can I automatically move the tab to the next entry field in VisualAge?

Answer You can implement automatic tabbing in VisualAge by writing some script and making a connection to that script. Suppose your have two text entry fields. Go into the settings of TextField1 and set its text limit to five. If you then connect the object event of the text entry field to the following script, the tabbing will be done automatically.

Smalltalk

```
autoTab
| position limit |

position := ((self subpartNamed:  TextField1′) queryCursorPosition).
limit := (self subpartNamed:  TextField1′) textLimit.
(position = limit)
ifTrue: [ (self subpartNamed:  TextField2′) setFocus ].
```

If the user types five characters in the first text field, focus is automatically set onto the second text field.

Missing Carriage Return in Multiple-Line Editor

Question I am trying to write a string, which I got from a multiple-line editor, to a stream. It seems that I lose all the carriage returns by executing the following code. Any help would be highly appreciated.

Smalltalk

```
writeFile
  ″Write the contents of a multiple line editor to a stream″

  | fileStream, mleData |
  fileStream := CfsWriteFileStream openEmpty: temp.txt′.
  mleData := (self subpartNamed: Multi-line Edit′) string.
  fileStream nextPutAll: mleData.
  fileStream close.
```

Answer To solve this problem, the only thing you have to do is change *string* to *object*. This will include a return character at the end of each string from the multiline editor.

Action on PFkey Pressed

Question I would like to close a window when the PF12 key is pressed. The problem I have is how to see if PF12 gets pressed?

Answer This is not quite as trivial as it seems, but it can be done:

1. Every visual part you attach to your window has a corresponding CwWidget attached to it. It can be accessed via *(self subpartNamed: ′myText′) widget*. It is accessible once the window is created, for example, in scripts hooked to *aboutToOpenWidget*.
2. Attach a keyboard message handler to each and every such widget that is capable of getting input focus, something like this:

   Smalltalk

   ```
   (self subpartNamed: ′myText′) widget
      addEventhandler: KeyReleaseMask
      receiver: self
      selector: #keyRelease:clientData:event:
      clientData: nil.
   ```

3. Code the actual event handler corresponding to the selector. Create a method with signature *keyRelease:clientData:event:*.
4. Every time a key is released this method will be called. The event is the place to look for specific key information (just *printString* it to the Transcript, it will tell you most everything about the key released). Whenever you think it is appropriate to close the window, just do it.

Chapter 4. IBM Smalltalk Programming Language

This chapter covers programming tips for the IBM Smalltalk language.

Smalltalk Books

Question What mass-market Smalltalk books are available?

Answer

- *Smalltalk 80, The Language*, Adele Goldberg & David Robson, Addison-Wesley 1989, ISBN 0-201-13688-0
- *Smalltalk 80, The Interactive Programming Environment,* Adele Goldberg, Addison Wesley 1984, ISBN 0-201-11372-4
- Smalltalk 80, Bits of History, Words of Advice, Glenn Krasner, Addison Wesley 1984, ISBN 0-201-11669-3
- Inside Smalltalk, Volume I, Wilf Lalonde & John Pugh, Prentice Hall 1990, ISBN 0-13-630070-7
- Inside Smalltalk, Volume II, Wilf Lalonde & John Pugh, Prentice Hall 1991, ISBN 0-13-465964-3
- Object-Oriented Graphics, P. Wisskirchen, Springer-Verlag 1990
- Practical Smalltalk: Using Smalltalk/V, Dan Shafer and Dean A. Ritz, Springer-Verlag, ISBN 0-387-97394-X
- Rapid Prototyping for Object Oriented Systems, Mark Mullen, Addison Wesley 1990, ISBN 0-201-55024-5
- Object-Oriented Design, Peter Coad and Ed Yourdon, Yourdon Press 1991
- Object Oriented Programming for Artificial Intelligence, Ernest Tello, Addison Wesley 1989, ISBN 0-201-09228-x
- The Well Tempered Object, Stephen Travis Pope, MIT Press 1991, ISBN 0-262-16126-5
- Human-Computer Interface Design Guidelines, C. Marlin Brown, Ablex Publishing 1989, ISBN 0-89391-332-4
- Designing Object-Oriented Software, Rebecca Wirfs-Brock, Brian Wilkerson, and Lauren Wiener, Prentice-Hall 1990, ISBN 0-13-629825-7
- An Introduction to Object Oriented Programming & Smalltalk, Lewis Pinson & Richard Wiener, Addison Wesley 1988, ISBN 0-201-19127-x

- Object Oriented Design with Applications, Grady Booch, Benjamin/Cummings 1991, ISBN 0-8053-0091-0
- A Quick Trip to ObjectLand: Object-Oriented Programming with Smalltalk, PTR Prentice-Hall 1993, ISBN 0-13-012550-4
- IBM Smalltalk: The Language, David N. Smith, Benjamin/Cummings 1995, ISBN 0-8053-0908-X
- IBM Smalltalk Programming for Windows and OS/2, Dan Shafer and Dan Herndon, Prima 1995, ISBN 1-55958-749-0
- Smalltalk with Style, Suzanne Skublics, Edward J. Klimas, and David A. Thomas, Prentice Hall 1996, ISBN 0-13-165549-3

Question What books or reports about Smalltalk are available from IBM, besides product documentation?

Answer

- *Object-Oriented Design - A Preliminary Approach*, IBM 1990, GG24-3647
- *Developing a CUA Workplace Application*, IBM 1991, GG24-3580
- *Object Oriented Analysis of the ITSO Common Scenario*, IBM 1990, GG24-3566
- *Cooperative Processing in an Object-Oriented Environment*, IBM 1991, GG24-3801
- *A Practical Introduction to Object-Oriented Programming*, IBM 1992, GG24-3641-01
- *Smalltalk Portability: A Common Base*, IBM 1992, GG24-3903
- *Object-Centered Environments for Smalltalk*, IBM 1990, RC 15548
- *The Geometry Browser - A Feature Modeler in Smalltalk*, IBM 1990, RC 15394

Smalltalk Standardization

Question What is going on with the effort to standardize Smalltalk?

Answer ANSI officially approved the IBM project proposal for Smalltalk. The project name is X3 Project 986-D and the technical committee is X3J20.

Smalltalk Portability: A Common Base, document number GG24-3903, contains the proposed ANSI standard.

##() Construct

Question What does ##() do?

Answer The ##() construct in IBM Smalltalk is not unlike #define from the C world. It allows you to have objects created at compile time, rather than at run-time. We all use #(1 2 3) to create an array at compile time. Well, ##() is no different. It just lets you create objects other than arrays. Writing

Smalltalk

```
##( expression )
```

evaluates the expression at compile time, as if it were text selected in a workspace, but takes the resulting object and places a reference to it in place of the ##() construct. Thus,

Smalltalk

```
x := ##( 60 * 60 * 24 )
```

would return the number of seconds in a day. The expression is evaluated at compile time. At run time, it acts as if this had been the source code:

Smalltalk

```
x := 86400
```

Now, it is possible that the compiler could make this optimization. But how about this?

Smalltalk

```
##( | birthdays |
    birthdays := Dictionary new.
    birthdays at: ′Boris′  put: ′Jan 05′.
    birthdays at: ′Dennis′  put: ′Jul 05′.
    birthdays at: ′Leo′   put: ′Jan 05′.
    birthdays )
```

The dictionary is created at compile time, and not every time one runs the method. (Of course, that means there is just one copy around and that may not be what you want at all!)

Note!

The ##() construct is different from the ## construct. The ##() will evaluate whatever is in the parentheses, making it possible for a method to return a constant, such as ##(Date today), which is the date the method was compiled. Note that the answer is an actual Date instance.

The ## construct is an atom (EsAtom). (You can also use ##′′, for example, the expression *##hello == ##′hello′* is true). There is only one instance of an atom with some name. Once you create an atom, you cannot access or change the data in it. So atoms are really used for state purposes (to avoid using symbols, to make packaging easier in IBM Smalltalk).

Watch this!

The ##() construct is not documented in the IBM Smalltalk manuals. Use it at your own risk!

Question But IBM Smalltalk Version 3 does many optimizations automatically, so I don′t need ##() for this. Is there a nontrivial example of usage of ##()?

Answer Yes, for example you can use it to take some specific action each time you recompile the class. See the next section, "Taking Actions Automatically at Class Recompilation," for details.

Taking Actions Automatically at Class Recompilation

Question Typically, the *initialize* method is run once when a class is loaded (which most systems do automatically when it is filed in). When making changes to the class, it is easy to forget to run the method. Most people put a comment at the top:

Smalltalk

```
″ Don′t forget to reinitialize me when you make changes:

  ThisClassName initialize  ″
```

and then select the expression and evaluate it after making a change.

Is there a way to take actions automatically each time when the class is recompiled?

Answer Yes. In the initialization method, add this line:

Smalltalk

```
##( System message: self class name asString,
    ′ needs to be initialized.′ )
```

This puts up a dialog at compile time to remind you to initialize the class.

The compiler, being so helpful sometimes, complains that this expression doesn′t do anything. So, fake it out:

Smalltalk

```
| junk |
...
junk := ##( System message: self class name asString,
                    ′ needs to be initialized.′ )
```

But you say, why not initialize the class directly, by putting this at the end of the *initialize* method?

Smalltalk

```
##( ThisClassName initialize )
```

If you try it, you discover that it indeed does run the method. And you discover after a while that it doesn′t contain the latest change, but always the one before. Make a change, and it doesn′t show up; make another and the first shows up! It′s clear once you realize that:

Smalltalk

```
##( ThisClassName initialize )
```

runs the method that is in the system. The new method being compiled, which you′d like to run, has not yet been inserted into the system. Thus you run the last one that was compiled. And the next time, you run this one, and so on.

Warning!

I′ve discovered that writing to the Transcript from within ##() has some problems.

Weak Pointers

Question What does the term ″Weak Pointers″ mean? Can I use weak pointers in IBM Smalltalk?

Answer Weak pointers are pointers you can have to objects that do not prevent the garbage collector from destroying the objects. In other words, if there are only weak pointers to an object and no other pointers, the garbage collector will destroy the object. You can use weak pointers in IBM Smalltalk, for example:

Smalltalk

```
| string array |

string := String new: 100.
array := (Array new: 10) makeWeak.
array at: 1 put: string.
string onFinalizeDo: #halt.
```

You′ll get a walkback the next time the finalize cycle runs (when the system goes idle).

Utility of Weak Pointers

Question Do weak pointers have practical importance? Can you give a couple of examples of when I might want to use them, and what they buy me?

Answer In VisualAge, a large number of collection classes have the word “weak” in their name. I′ll just mention briefly some usages of weak objects here. Below, I will talk only about a simple collection (like Bag, Set, OrderedCollection) and not Dictionary or LookUpTable. Various *WeakKey* collections or *WeakValue* collections have slightly different behaviors that the class names are trying to describe.

A weak collection can be used as a cache to hold cached objects. The cache itself can be assigned to a global variable, a class variable, and so on. Since the cache is a weak collection, references to the cache objects are not considered permanent references.

Normally, if I want to use the same object A in several windows, I would save that value in a global variable, or put that object A in some global collection. It is especially true if those several windows don′t really have any logical connection to each other. Since the object A is referenced by a global variable, it is not a candidate for garbage collection. However, if the global cache collection is a weak collection, every element of that collection is a candidate for garbage collection if that particular element is no longer referenced by any object that is live in terms of garbage collection.

Object Finalization

Question I heard that now *finalization* is a hot topic. What is it, actually? Does IBM Smalltalk support it? If yes, in what way?

Answer Finalization is roughly like destructors in C++. Normally, you don′t need to take any special actions to free objects. When an object is no longer needed (there are no more references to it), the Smalltalk garbage collector removes it from the object memory.

There are, however, cases when automatic garbage collection (GC) is not enough, as when you deal with operating system resources. For example, you can open a file and then leave the stream to GC′s mercy:

Smalltalk

```
myMethod
  | f |

  f := CfsReadFileStream open: ′MYFILE.TXT′.
  ^self

  ″In some moment, the GC will destroy the object
  and you are probably going to suffer″
```

In situations like this,you want to be notified when the garbage collector is going to destroy an object. You can do this, for example, by using weak pointers. Point two variables to the same object, one in the weak and one in the hard way, then examine whether the weak pointer is nil. If it is, you know the garbage collector was trying to get rid of your object and you are going to do an appropriate action such as closing a file.

In IBM Smalltalk, you have a much easier and stylish way of finalization. For example, you can say

Smalltalk

```
myMethod

 | f |
 f := CfsReadFileStream open: ′MYFILE.TXT′.
 f onFinalizeDo: #close.
 .....
```

and don′t worry any more about closing the file.

Standard Methods not in API Category

Question While wandering through the class hierarchy in IBM Smalltalk, I noticed that a number of what I considered standard Smalltalk APIs were in fact in non-API categories. For example the *value* messages sent to blocks passed as parameters are in the category ES-Internal. The method *deepCopy* is another example. Is there some reason why these messages are not in an API category?

Answer The *Smalltalk Programmers Reference* says about *deepCopy*:

″Unlike other Smalltalk implementations, CLDT does not provide a public default version of either deepCopy or shallowCopy in Object protocol. The semantics of both these messages are normally specific to each class, and often to an application.″

The categorization for #value has been taken care of for the next release. Note, also, that Block is an abstract superclass.

Using the Correct Variable Type

Question If I want many different methods to use a dictionary, should I have the following code:

Smalltalk

```
SomeName:=Dictionary new.
```

and assign the SomeName to be a global variable? Is this the correct way of doing it? I guess the very first time it will ask me if I would like to make SomeName a global variable.

Answer If those methods are all instance methods on the same class, then you would store the dictionary in an instance variable. Otherwise, you might want to make it a class variable with a public *get* method. That way, your global name space does not get overloaded.

Smalltalk printf()-like Formatter

Question Is there any code available that does string formatting like printf() in C?

Answer You might want to check out EsString>>#bindWith: and all its variants in IBM Smalltalk. It′s good for the case where you need to substitute strings at set positions within a source string.

The *bindWith:* family of messages answers with the string formatted under the control of the receiver. The receiver is a character string patterned after the formatting conventions of printf() and contains field descriptors used to insert arguments into the format string.

If you use the *bindWith:* family of messages and you have VisualAge for Smalltalk, then you can send each argument the message #convertToDisplay, and the conversion will be performed along with the substitution. If you just have IBM Smalltalk, you may want to extend the Object class to have an *asString* method that returns *^self printString*,, and you would then be able to use *asString* polymorphically with your arguments.

Testing Instances and Classes

Question It seems strange that I have to create special classes to test my classes and methods. Doesn′t it seem that Smalltalk should have a *test method* besides instance and class methods. The idea is that a builder like VisualAge would strip out test methods when the class is packaged. With test methods for both class and instances, I′d be able to keep the tests with the class.

Answer Why don′t you code “Example” comments in each method which are the test cases. (I reckon about 30% of my code is such test cases.) You can highlight the example code and execute it to verify the test, but as it′s a comment, it′s not stored in the image. One day when I have some spare time I′ll write a program to automate this testing, though my experience is that there are frequently insignificant changes to the output that would frustrate the automation (I can see the difference is minor, the test program would not).

As an aside, a crucial requirements for such testing to work is to code a decent *printOn:* method in every new class, which prints enough details to identify the result. Merely relying on Object′s default ″a NewClass″ makes testing very hard.

You can always use a naming convention. One that′s possible in Digitalk Smalltalk and perhaps elsewhere is to use a capital letter in the method name (Digitalk creates *Doit* methods all over the place), for example: Test or TestInteraction. It′s not to hard to strip such methods from the image.

With Visualage for Smalltalk, you should be able to extend your class in a separate test application to hold all the (instance/class) test methods for your class.

Garbage Collection Rate

Question Can a Smalltalk application accessing many data objects in a short time from a large relational database (via a persistence layer) cause the workstation to page or swap excessively because the rate of object creation exceeds the rate of garbage collection?

Answer The only way to determine this is to try. When (and if) garbage collection runs differs from product to product, release to release.

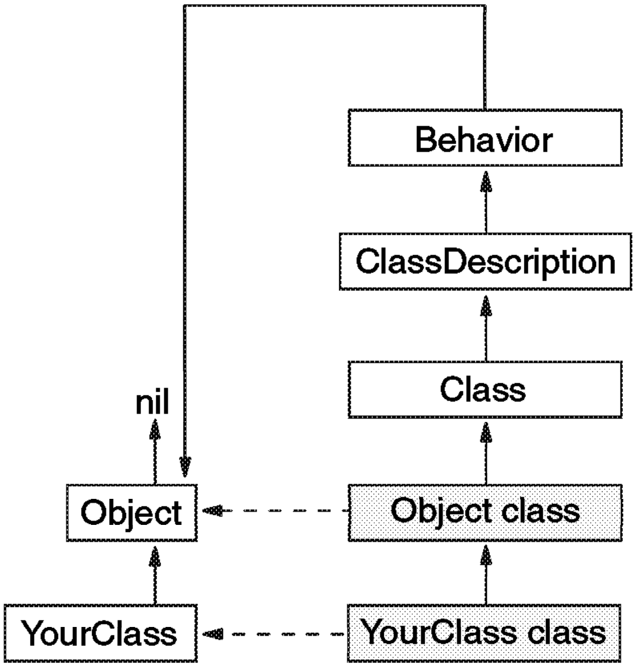
If you are not happy with the rate of garbage collection, you should be able to override the default interval of garbage collection and force it whenever you want. We noticed that IBM Smalltalk will appear hung sometimes since the garbage collector is invoked only when the system is idle, but if you are running a long-running task that creates a lot of objects, you can run into a situation where you blow the available memory. The way we worked around it is by creating a high-priority background task that wakes up every few seconds and checks if the available memory is running low. If it is, we force the garbage collect ourselves. It seems a patch that should not be necessary, but it made our hang-ups go away.

Inherited Messages

Question How can I see all inherited messages?

Answer The best way to see all messages that instances of your class can understand is to change the value for method visibility on the Class menu of most browsers; this will show inherited methods as well as methods defined on your class itself.

With the class methods, things are a bit more tricky. There are class methods that you don′t see in any of your object′s superclasses up to Object, but which your class still does understand. For example, suppose YourClass is a subclass of Object; YourClass class does understand the message *new,* but it′s not a class method of either YourClass nor Object. How this can be? A look at the following picture makes this clear.



For example, the method *new* is not a class method of Object, but all classes understand it. That′s because the class Object is itself an instance of the metaclass Object (which in turn is an instance of the metaclass). When you execute *YourClass new*, method lookup starts in YourClass class and continues to Object class; but Object class itself is inherited from Class, and Class is (indirectly) inherited from Behavior, and Behavior has an instance method *new.* If you look at all public instance methods of Behavior, then you will see *new*, which basically calls a primitive, VMprBehaviorBasicNew. I don′t know why it doesn′t appear in the catalog of classes, as it is contained in the CLIM-API methods category. You′ll see what I′m talking about by selecting **Browse Class** in your System Transcript and entering *Behavior.*

File System

Question How do I get the contents of a directory?

Answer The contents of a directory can be determined by using an instance of CfsDirectoryDescriptor. For simplicity, a reusable class named FileManager can be created to provide higher-level abstractions of the Common File System API.

The following method evaluates a block with the names of the files in a directory that match a given pattern:

Smalltalk

```
filesMatching: pattern on: pathName do: block

   ″Evalutes block with all files on pathname
    which match the provided pattern. Note that only
    the names of files are provided (no full paths).″

   | directory entry |
   directory := CfsDirectoryDescriptor
      opendir: pathName
      pattern: pattern
      mode: FREG.
   directory isCfsError ifFalse: [
      [(entry := directory readdirName) notNil]
         whileTrue: [block value: entry].
      directory closedir]
```

FREG is an entry in the pool dictionary CfsConstants. The *mode* keyword in the message *opendir:pattern:mode:* uses flags to determine which files to find. FREG provides the names of all regular files. FDIR provides the names of subdirectories. To get both the names of files and directories, pass *FREG : FDIR* as the parameter to this keyword.

Use this method to print the names of all text files in the root directory (the following code assumes the above method has been implemented in a class called FileManager).

Smalltalk

```
FileManager
   filesMatching: ′*.txt′
   on: ′/′
   do: [:fileName | Transcript cr; show: fileName]
```

The root directory can be specified as a forward slash, which is very UNIX. This works in OS/2 and in Windows as well as in AIX. It may be handy to get a collection containing the file names:

Smalltalk

```
filesMatching: pattern on: pathName

  ″Answers a collection with all files on pathname
   which match the provided pattern. Note that only
   the names of files are provided (no full paths).″

  | files |
  files := OrderedCollection new.
  self
     filesMatching: pattern
     on: pathName
     do: [:fileName | files add: fileName].
  ^files
```

File Existence

Question How do I check in Smalltalk if a file exists?

Answer Use the methods provided above to determine if a file exists:

Smalltalk

```
fileExists: fileName on: pathName

    ″Answers whether or not the file name file
     name exists on the path named pathName.″

    ^(self filesMatching: fileName on: pathName) notEmpty
```

Accessing COM Ports

Question What is the correct way to access a COM port from Smalltalk? Are there special classes or do I have to use C for that?

Answer There are no special classes for that in Smalltalk, but you can directly use IOCtl from Smalltalk without any C code. For example, if you want to know how many bytes are on the input queue, use a script like this:

Smalltalk

```
 inputQueueCount
   | rc fileDescriptor queueLength |

  queueLength := ByteArray new: 2.
  fileDescriptor :=  deviceFile fileDescriptor.
  rc :=fileDescriptor dosDevIOCtl: IoctlAsync
        function: AsyncGetinquecount
        pParams: nil
        cbParmLenMax: 0
        pcbParmLen: nil
        pData: queueLength
        cbDataLenMax: queueLength size
        pcbDataLen: nil.

  (rc = 0) ifFalse: ^AbtError new.
  ^(OSType fromBytes: queueLength) asInteger
```

where you get the *deviceFile* with:

Smalltalk

```
open: portName
 ″Perform the open action.″

  deviceFile := CfsFileDescriptor
    open: ′COM1′
    oflag: ORDWR
    share: ODENYNONE.

  deviceFile isNil ifTrue:
    (CwMessagePrompter errorMessage: ′Error opening COM1′).
```

You can read from COM1 with something like:

Smalltalk

```
 ((queueLength := self inputQueueCount) > 0)   ifTrue:
     [ deviceFile  read: (buf := String new: queueLength)
         startingAt: 1
         nbyte: queueLength.
     self incomingData: buf. ]
```

Catching Errors

Question How do I catch errors in Smalltalk?

Answer To catch errors that are difficult or impossible to anticipate, use the exception handling facilities provided by IBM Smalltalk. The simplest form of exception handling involves catching errors which occur while attempting to evaluate a block. Use this:

Smalltalk

```
 [ <do something> ]
     when: <error> do: [:signal | <handle the error> ]
```

The message *when:do:*, when sent to a block, evaluates the block. If an error matching the first parameter occurs, the exception handler will be invoked.

The first parameter describes the kind of error that can be trapped. Potential values for this parameter can be found in the pool dictionary called SystemExceptions. They are:

- ExError – Any error
- ExHalt – A halt has been encountered
- ExIndexOutOfRange – An attempt to access a nonexistent array index
- ExCFSError – File system error

The second parameter is a block that takes a single argument. That argument is an instance of class Signal. Tell the signal what to do next:.

- *exitWith:* – Exit the block with some value
- *retry* – Reattempt evaluation of the block
- *signal* – Pass the responsibility for handling the error to the outer context

File System Errors

Question How do I catch file system errors?

Answer The following method opens the file named by its first parameter. The second parameter is a block to evaluate in the event of an error.

Smalltalk

```
contentsOfFileNamed: fileName ifError: errorBlock

  ″Answers the contents of the text file named aString.
  If an error occurs, then the result of evaluating
  errorBlock is answered.″

  | file contents |
  file := CfsReadFileStream open: fileName.
  file isCfsError ifTrue: [^errorBlock value].
  contents := [file contents]
     when: ExCFSError do: [:signal | signal exitWith: nil].
  file close.
  contents notNil ifTrue: [^contents].
  ^errorBlock value
```

First, the file is opened. If an error occurs during the opening of a file, the result of the *open:* message will be an object that answers *true* to *isCfsError.*

Second, the attempt to read the contents of the file is wrapped in a block. That block is sent the message *#when:do:*. The first parameter to this message is the kind of error we are interested in (ExCFSError in this case), and the second parameter is a block containing the code to evaluate in the event that this kind of error occurs. The block takes a single parameter, an instance of class Signal.

In this case, an error when trying to read a file is considered final. The message *exitWith:*, sent to the Signal, will exit the block with the provided value (nil in this case).

The variable *contents* should either be a string or nil. The file is closed and if successful, the contents is answered. Otherwise, the result of evaluating errorBlock is answered.

Catching All Errors

Question How do I catch all errors?

Answer Use ExError for the kind of error to catch:

Smalltalk

```
| succeeded |

succeeded := [ <do something>. true ]
   when: ExError do: [:signal | signal exitWith: false]
```

In this example, the variable *succeeded* is set to *true* if the <do something> works without signaling any errors; the variable is set to *false* otherwise.

You should not generally try to catch all errors, because you will also catch programming errors, including messages not understood. Flaws in your code may go unnoticed with unexpected results.

Exception-Handling Code

Question When should I include exception handling code?

Answer In general, exception handling should be added after you are convinced your code works. Exception handling should be used to catch unexpected problems— for example, your file server shutting down.

Fixed Decimal Class

Question Is there a fixed decimal class available or any other support that helps with the rounding issues encountered with floating decimals?

Answer If you are stuck with a Smalltalk without fixed decimal, don′t forget that Smalltalk fractions are true fractions, so 10075/100 is a true representation of 100.75, although it will print as 403/4. Also, Decimal class is part of the base product in Version 3.

Object Class Identification

Question How can I programmatically see if *anObject* belongs to *aClass*?

Answer This is actually two separate questions.

Object>>#isKindOf: aClass answers a Boolean which is *true* if *aClass* is the class or a superclass of the receiver, and *false* otherwise.

Object>>#isMemberOf: aClass answers a Boolean which is *true* if *aClass* is the class of the receiver, and *false* otherwise.

Floating Point Numbers

Question I believe there is much oddity with the behavior of floating point numbers:

- (534.92 * 100) is displayed as 53492.0
- 53492.0 truncated is displayed as 53492
- (534.92 * 100) truncated is displayed as 53491

Is this as bug?

Answer

This is not a bug. No matter how odd they look, floats are precisely what scientific and engineering problems need. Those oddities were carefully and specifically designed.

- Floating point numbers are designed for use in the engineering and scientific communities where calculations involve wide ranges of number sizes.
- One basic rule is: **Never compare Floats for equality**.
- It is impossible, in general, to precisely convert decimal numbers to binary, and floats are binary. Thus most numbers are *wrong* as soon as they are converted. Multiplying them by 100.0 may make a particular calculation look *right* but just shift the problem to another calculation.
- Certain operations can cause the loss of precision. One famous *gotcha* involves subtracting two floats that are nearly equal. The precision of the result may be only a digit or two, or even none at all.

To see what is actually going on I used some code that prints floating point numbers to any length. I also coded a method which prints IEEE double-precision floats in hexadecimal. (The method is near the end.) The exponents, in hexadecimal, are the number of bits to shift the fraction.

Lets look at some examples. First, just some plain numbers:

Smalltalk

```
0.00 printHex      ′0e00′
1.00 printHex      ′1.0e0′
10.0 printHex      ′1.4000000000000e3′
16.0 printHex      ′1.0e4′
```

Now, the numbers from the thread, and one number that′s 0.01 larger:

Smalltalk

```
1502.03 printHex   ′1.7781EB851EB85eA
1502.04 printHex   ′1.77828F5C28F5CeA′     (Note difference in fifth digit.)
1051.42 printHex   ′1.6DAE147AE148eA′
```

The *printStringWidth* method prints numbers to the given character width. The width used here is 30, which means 28 digits (less period and sign). As a result, it may show junk past the end of the number. In the cases below, it shows extra zeros.

Smalltalk

```
1502.03 printStringWidth: 30
       ′1502.0299999999999700000000000′

1051.42 printStringWidth: 30
       ′1051.4200000000000700000000000′
```

Now, calculating the difference we see that the expected answer 450.61 is not quite what we get:

Smalltalk

```
(1502.03 - 1051.42) printStringWidth: 30
       ′450.609999999999990000000000000′

450.61 printStringWidth: 30
       ′450.610000000000001000000000000′

(1502.03 - 1051.42) printHex              ′1.C29C28F5C28F4e8′
450.61 printHex                           ′1.C29C28F5C28F6e8′
```

In fact, the difference is just in the rightmost two bits, but that′s enough to make it not equal. Now, what do *roundTo:* and *truncateTo:* do?

Smalltalk

```
((1502.03 - 1051.42) roundTo: 0.01) printStringWidth: 30
      ′450.610000000000001000000000000′

((1502.03 - 1051.42) truncateTo: 0.01) printStringWidth: 30
      ′450.600000000000002000000000000′
```

Clearly, equality testing is not going to work. But, consider the following:

Smalltalk

```
 (1502.03-1051.42) between: 450.605 and: 450.615
```

which *does* answer *true* and will answer *true* unless and until the real, internal precision gets very much smaller (or the values get much, much larger). The former is not likely when doing currency calculations.

A method which does the between:and: test for floats is:

Smalltalk

```
| Float publicMethods |
equals: aFloat to: delta
″ Compare self (a float) with aFloat to a ′precision′ of delta.
  Example: 1502.03 equals: 1051.42 to: 0.01 ″

| deltah |
deltah := delta / 2.
^ self between: aFloat-deltah
         and: aFloat+deltah
```

Examples:

Smalltalk

```
(1502.03 - 1051.42) equals: 450.61 to: 0.01           true
(1502.03 - 1051.42) equals: 450.61 to: 0.001          true
(1502.03 - 1051.42) equals: 450.61 to: 0.0000001      true
(1502.03 - 1051.42) equals: 450.61 to: 100.0          true

(1502.03 - 1051.42) equals: 450.62 to: 0.01           false
(1502.03 - 1051.42) equals: 450.60 to: 0.01           false
```

Smalltalk

```
!Float publicMethods !

printHex
 ″ Answer a string which is the hexadecimal representation of a float.
   Assumes IEEE format  ″
 | shifty sign exp fraction str |

 shifty := 0.
 8 to: 1 by: -1 do: [ :n |
   shifty := shifty<<8 + (self basicAt: n) ].

 sign := (shifty bitAt: 64).
 shifty clearBit: 64.
 exp := (shifty >> 52).
 fraction := shifty bitAnd: 16r000FFFFFFFFFFFFF.
 (exp = 0) & (fraction = 0) ifTrue: [ ^′0e00′ ].
 str := sign = 0 ifTrue: [ ′′ ] ifFalse: [ ′-′ ].
 exp := exp - 16r3FF.
 str := str, ′1.′, (fraction printStringRadix: 16 showRadix: false).
 str := str,′e′, (exp printStringRadix: 16 showRadix: false).
 ^ str
```

Reversing Collections

Question What collections can be reversed?

Answer The answer is all sequenceable collections, that is, SequenceableCollection and all its subclasses implement the *reverse* method.

Substrings

Question Something that looks and sounds simple is not turning out to be simple at all. I am dealing with a string in the following format: *lastName*, *firstName*, *midInitial*. From this string I want to create a new string that contains only the *lastName* but I can′t figure out how to do this.

Answer There are many ways to unpack a string. One way to get the *lastName* is the following:

Smalltalk

```
| str array lastName initial |

str := ′Doe, John A′.
array := str subStrings.
lastName := array at: 1.
initial := array at: 3.
```

Implementation of #become:

Question How efficient is *become:* in IBM Smalltalk Version 3? I had thought that it was all done internally by placing a proxy for the new object in the same memory address as the old object, so that the garbage collector then removed these proxies (the added level of indirection is temporary). However, someone has suggested that this may not be true. Apparently, Parkplace does it this way, but Digitalk doesn′t, and their implementation is not very efficient. Anyone know anything?

Answer I can′t answer your question for sure, but here is some information on *become:*.

Once, Smalltalk implementations had an object table. Object pointers were indexes into the table. A *become:* was simply a swap of two pointers in the object table. It was fast, and symmetrical.

But, object tables were of fixed size and one had to know ahead of time how big a table to allocate. Further, each object reference was indirect through the table: calculate the table address given the index and then fetch the real pointer to the object.

So what to do? Without an object table, doing a full-fledged *become:* requires scanning all objects in the world to find pointers to other objects and changing the affected pointers each place they are found. What was a few microseconds suddenly became several seconds. Not good!

So, two schools of wizards developed. One school forbade symmetric *become:* entirely. All *become:* had to be asymmetric, with the second object simply replacing the first. While this still required scanning the world, it was faster than the work required to to a full symmetric *become:*.

The second school felt that *become:* was too important to go away. They made all object pointers indirect— not through an object table, but through a hidden pointer. While this took more memory, and made object references slower than the other wizard′s solution, it was better than using an object table.

But which wizard is which? One can often tell a wizard by the spells they use. In Smalltalk, running some timing code often lets one deduce which spell was used. Below, are tests of the three implementations I have handy: IBM Smalltalk V3 on a 486 under Warp, VisualWorks 2.0 on a Mac Quadra 950 (68040), and Digitalk Smalltalk/V-Mac on the same Quadra. I ran some tests. First is the *become:* test:

Smalltalk

```
System globalGarbageCollect.
Time millisecondsToRun: [
  1000 timesRepeat: [
     | a b |
     a := ′asdf′ copy.
     b := a.
     a become: ′qwer′ copy.
     a = b ifFalse: [ self error: ′Whoops!′].
     b   ] ]
```

It creates an object, assigns it to two variables (so to speak), and then does a *become:* with a similar object. When done, it tests to see if the two objects have the same value. If not, something went astray. It answers b, which has the same value as a. (Select the innermost block contents and do evaluate it.) A similar test uses assignment instead:

Smalltalk

```
Time millisecondsToRun: [
  1000 timesRepeat: [
     | a b |
     a := ′asdf′ copy.
     b := a.
     a := ′qwer′ copy.
     a = b ifTrue: [ self error: ′Whoops!′].
     b  ]  ]
```

Since it uses assignment, b is unchanged and the test is reversed. If an implementation has an object table, or uses pointer indirection, one would expect the two methods to run with very similar timings. If it has no object table, but has to search the world, the two should be very different. When timing the tests on IBM Smalltalk Version 3, I had the following results: 6230, 6232, 6220, 6264. The results without *globalGarbageCollect* were 13417, 12105, 10348, 14050, 3355. Then I used the second method with the results being 66, 69, 66. The ratio is roughly 6200/66 or roughly 100:1. Now I was timing the code on VisualWorks 2.0. The first method with *become:* resulted in 396, 464, 460, the second method came back with 56, 52. The ratio here is roughly 400/50 or 8:1. Finally, I timed the test on Smalltalk/V-MAC. The result of the first version were 9617, 6917, 8100, 8417. The second method resulted in 183, 183, 183. The ratio is roughly 7000/180 or roughly 40:1. In conclusion, IBM and Digitalk got 100:1 and 40:1 which indicates considerable slowness is involved. VisualWorks gave only 9:1, which is a lot faster. One would suspect the wizards are using different spells, but which ones? Isn′t 100:1 too fast for a search of the whole world? Isn′t 9:1 too slow for a simple indirection? I don′t know...

#to: Method

Question What does the Number class>>to: do? I get *to:do:*, and I understand what this does, but what is this plain *to:*?

Answer The *to:* message answers an instance of a collection named Interval.

Smalltalk

```
1 to: 10
```

is equivalent to:

Smalltalk

```
Interval from: 1 to: 10
```

and is equivalent to:

Smalltalk

```
Interval from: 1 to: 10 by: 1
```

Intervals act like collections if you read from them, but they cannot be modified. Further, they are not stored as collections so that (1 to: 1000000) doesn′t take any more space than (1 to: 10). For example, you can write loops just as with any other kind of collection:

Smalltalk

```
(Interval from: 1 to: 10) do: [  ...block... ]
```

For convenience, the *to:* (and *to:by:*) messages of Number answer intervals so this can be written as:

Smalltalk

```
(1 to: 10) do: [  ...block... ]
```

And for convenience, the *to:do:* (and the *to:by:do:*) messages of Number make things even easier:

Smalltalk

```
1 to: 10 do: [  ...block... ]
```

Interval>>reverse Method

Question The comment explaining what *reverse* does, says: *″Answer a object conforming to the same protocols as the receiver, but with its elements arranged in reverse order.″* I can′t make head or tail of this. Why not say simply ″Answer an Interval in reverse order?″

Answer Inspect the following code:

Smalltalk

```
(0 to: 5) reverse
```

The elements of the result are just as you expected. However, the class of the result may not be what you expected (notice it′s an Array, not an Interval).

Updating Widgets

Question We have some code that sends messages to a visual part, which is then supposed to display some status information in a text window. The code works fine except that the status messages are showing up asynchonously (that is, none show up at all until they all show up at once in a big flood). I am guessing that what is happening is that the VisualAge part is getting some events queued up, but that these events aren′t handled until the user interface process gets control again. Is there some way of forcing these events to be handled right away so that the updates to the status text window will be synchronous?

Answer To do so, try the *updateWidget* message. For example, here is a sample script for a window with a button, a text part, and a label. This script should be invoked from the button click:

Smalltalk

```
buttonClicked
″Demonstrate how to update text on a widget″

1 to: 100 do: [:i |
  (self subpartNamed: ′Label1′) object: i printString.
  (self subpartNamed: ′Label1′) primaryWidget updateWidget.
  (self subpartNamed: ′Text1′) object:  i printString.
  (self subpartNamed: ′Text1′) primaryWidget updateWidget.
  (self subpartNamed: ′Push Button1′) object: i printString
].
```

All of these controls will reflect the changes instantly. The push button does not need to have an *updateWidget*.

Incremental Compiler

Question Does IBM Smalltalk come with an incremental compiler?

Answer Yes, IBM Smalltalk does use incremental compilation. Incremememental compilation means that when you save a changed Smalltalk program (typically quite short), it is compiled as part of the Save process; you don′t have to recompile everything in the Smalltalk application, just what you′ve changed. You rarely notice the compiler working, it′s so fast.

IBM Smalltalk comes with a lot of other features for programmer productivity: the TrailBlazer code browser (unique to IBM), application change management in the IBM Smalltalk Professional version, and and Motif-based GUI widgets that work on all platforms.

Caching Compiler

Question Is the IBM Smalltalk compiler a caching compiler?

Answer Absolutely. IBM Smalltalk moved from being one of the slowest to being the fastest with Version 3.

Creation of Blocks at Run-time

Question Is it possible to create and execute blocks of Smalltalk code at run time? For example, I want to read a block from a file. The string I read should then be converted to a Smalltalk block and this block should be executed.

Answer The basic answer is no, because you need to ship the Smalltalk compiler as part of your application, and that′s not allowed in the Smalltalk license agreements.

The fuller answer is to ask why you want to do this. Are you dynamically creating blocks in response to (say) user input, or just selecting one block from a collection (stored on disk)?

If the former, you will need some sort of compiler or interpreter to translate the user′s input to the blocks in the first place, so you could instead produce an interpreter that works on the user′s input, and store that input on disk instead of the block you would have produced

If you are just selecting a block from a collection, perhaps you should be creating classes that all implement a behavior according to the user′s input; the methods that implement these behaviours contain the code that would have been in your blocks.

Pool Dictionaries

Question What is the exact mechanism used by Smalltalk to resolve Pool Dictionaries constant references? Apparently it is not a static substitution of the value at compile time (because pool dictionary entries can be dynamically changed), nor, I believe, is there a complete dictionary lookup.

Answer The association (key and value pair), not the value itself, is compiled into the method. Therefore, run-time dictionary lookups are not needed and changes to the value done via the pool dictionary don′t require any recompiling of the method.

Text Color Changes

Question While the application is running, I send the message

Smalltalk

```
(self subpartNamed: ′Text2′)
  backgroundColor:′yellow′;
  foregroundColor: ′blue′ ;
  show.
```

to a text subpart. When I put a halt anywhere in the running code, then Text2 shows the changed color. When I don′t stop in the code, then the changed color is not displayed. What can I do to have the color change visible with a running application? I need this indication to know what is happening with the application.

Answer Sounds like you are not giving the user interface process a chance to run. Either use *updateWidget* to force it to update immediately, or fork your work process at a lower priority (*Processor userBackgroundPriority*) and use *CwAppContext default syncExecInUI:* to update the user interface

Sorting SortedCollections

Question In Digitalk Smalltalk, SortedCollection includes the method *reSort.* I′m trying to implement my own *reSort* method:

Smalltalk

```
reSort
| newColl |

size==0 ifTrue: [ ^self ].
newColl := self species new: size.
newColl sortBlock: self sortBlock.
newColl addAll: self.
elements := newColl elements.
```

However, this method produces random results (the contents of the SortedCollection don′t get sorted). Am I missing something?

Answer Yes, you get the same result if you evaluate the following code in a Workspace:

Smalltalk

```
| newColl coll |

coll := #( andi boris vincent ) asSortedCollection.

newColl := coll species new: coll size.
newColl sortBlock: coll sortBlock.
newColl addAll: coll.

coll elements: newColl elements.

^coll.
```

But now, try this:

Smalltalk

```
| newColl coll |

coll := #( andi boris vincent ) asSortedCollection.

newColl := coll species new: coll size.
newColl sortBlock: coll sortBlock.
newColl sorted: true.
newColl addAll: coll.

coll elements: newColl elements.

^coll.
```

Now, the resulting collection gets sorted.

Compiler Severity Level

Question How do I control the compiler warning severity level?

Answer You must have the TrailBlazer loaded and enabled. From the TrailBlazer pulldown menu, select **Options** → **Compiler Warning Level**.

Finding the Sender Signature

Question How do I get a hold of the signature (Class>>#method) for the sender of a method? In other words, during the execution of a method, how can I find out who sent me?

Answer To get the sending method, just send:

Smalltalk

```
Processor activeProcess methodAtFrame: 1.
```

This will answer the CompiledMethod who sent the method currently executing.

It can be extremely instructive to have a look at EsImageStarUp class >> outputWalkback:on:process:

The debugger itself provides a good clue to information like this, but note that much of this code is development-time only and private. That said, take a look at:

```
Process>>#receiverAtFrame:
Process>>#stackAtFrame:offset:      (good comments in this method)
```

View the senders of *stackAtFrame:offset:* for lots of examples of how to use it.

Sender Context

Question How do I find the sender context of a method?

Answer You can use

Smalltalk

```
Process>>contextAtFrame: frameDepth
```

The following script prints out the top ten stack frames:

Smalltalk

```
| level nFrames |

level := 10.
nFrames := Processor activeProcess numberOfFrames - 1.
1 to: (level min: nFrames) do:
   [ :f |
     Transcript cr;
        show: (Processor activeProcess methodAtFrame: f) printString;
        show: ′  to ′;
        show: (Processor activeProcess receiverAtFrame: f) printString.
   ]
```

Pool Dictionary Tips

Question Are there tips to follow when working with pool dictionaries?

Answer Yes. In general, replace pool dictionary values using the *at:* method. If you replace the pool dictionary associations (or the entire pool dictionary) then you need to recompile all the methods that reference its values. When a method uses a pool constant, the association from the pool is stored directly in the compiled method. At execution time, the value is taken from the association and used. If you update the association value (using #atto the *new* pool will not be reflected in the execution of the code until it gets recompiled. It is always best to establish pool dictionaries at load time by using *toBeLoadedCode* for your application. Even if you cannot assign values immediately as they must be computed through the application′s logic, you should still build the dictionary with keys and use some form of lazy initialization to enter the values at an appropriate time. This helps not only with import and export, but also for loading your application into fresh images.

Text Display on Update

Question Why does my text not show up when I update the database?

Answer You can pause a process by using the Delay class, but it seems like something you shouldn′t have to do. The typical way of coding this is as follows:

1. Put up progress indicator
2. Fork background process: *forkAt: Processor userBackgroundPriority*
3. Background process calls *CwAppContext default asyncExecInUI: <update progress indicator>* occasionally.
4. Background process finishes, calls *CwAppContext default syncExecInUI: <close process indicator>*

Sometimes developers forget step (2) and run off the user interface process. It doesn′t halt VisualAge because the database component forks work processes, but they don′t get user interface updates as they expect. Alternative approaches to background processing are covered in the *IBM Smalltalk Programmer′s Reference*.

Pool Dictionary

Question How do I define a pool dictionary?

Answer A pool dictionary is simply a dictionary that is used to declare constants. You can define it with:

Smalltalk

```
Smalltalk at: #YourPoolDictionaryName put: (Dictionary new).
```

or:

Smalltalk

```
Smalltalk declarePoolDictionary: #YourPoolDictionaryName.
```

Swapper Error

Question Why do I get an error message when loading variable references? It says “swapper error.”

Answer Sounds like what you′ve dumped has a reference (possibly a subclass of, extension to, or contains a class variable instance of) a class that was reduced away in the run-time image you are binding to. Was the run-time image you are binding to created by the packager as a reduced image? If so, you must pay careful attention to both what actually exists in that image and what the application files you create require to ensure that the application file does not refer to nonexistent classes, methods, or both.

Hover-Help Behavior

Question Why do I see unpredictable behavior using hover-help?

Answer By default, hover-help displays the *labelString* attribute of a graphical label or push button. All other parts, including textual labels and push buttons do not display hover-help. Also if no label string is specified for a graphic label or push button, no hover-help will be displayed. To specify the label string for a graphic push button, open the settings for the part, click on **Text** under **Label type**, specify the label string, click on **Graphic** again under **Label type**, and click **OK.**

Aborting a Process Stack

Question How can I abort the current process stack?

Answer Either use exception blocks, such as *at:ifAbsent:*, or exceptions (see the *IBM Smalltalk Programmer′s Reference*). If you are just looking to stop the exceptions from propagating down the stack, you can add a *signal exitWith: nil* to the end of your exception block. If you instead want your entire application to stop, then you can just *System exit*.

Exiting an Image

Question In my application, is there any means to exit the image programmatically?

Answer Yes, *EmSystemConfiguration>>#exit* exits the image immediately, so try this:

Smalltalk

```
System exit
```

Configured Subsystems

Question How do I see what subsystems I am running?

Answer You can programmatically see what variant of each basic subsystem you are running, try this:

Smalltalk

```
System configuredSubsystems
```

This will answer a *LookupTable*, which on OS/2 may contain something like this:

′ABT′	′VA′
′ABTVER′	′V 3.0b′
′CFS′	′OS/2′
′CG′	′PM′

′CLDT′	′ES′
′CLIM′	′ES′
′CP′	′PM′
′CPM′	′ES′
′CW′	′PM′

Image Copyright

Question How do I go about getting the image copyright programmatically?

Answer You can get a string with the image copyright programmaticaly through executing

Smalltalk

```
System copyright
```

Command Line Arguments

Question Do I have access to the command line arguments?

Answer If you want the Smalltalk equivalent of the C variables argc and argv, send the following:

Smalltalk

```
System commandLine
```

This will return you an array of strings (including the executable name at index 1).

Run-Time Image

Question How do I find out whether the the current image is a run-time image?

Answer IBM Smalltalk has its own notion of whether the running image is a run-time image. You can get this information by sending

Smalltalk

```
System isRuntime
```

EsWeakSet and Finalization

Question How do I go about synchronizing an EsWeakSet with the real state of objects?

Answer EsWeakSet and finalization do work together; that is, finalization keeps a weak set of objects that could potentially be finalized in the next cycle. The finalization cycle is run when the system is idle.

Continue Background Process

Question Is there any way for a process to continue execution after *suspendExecutionUntilRemoved* has been issued?

Answer The example below shows that execution of background processing continues during *suspendExecutionUntilRemoved.*

Smalltalk

```
| shell widget |
shell := AbtShellView newPart openWidget.
[[ (widget := shell widget) notNil and: [ widget isDestroyed not ]]
whileTrue: [ CgDisplay default bell: 100. (Delay forSeconds: 3) wait ]]
  forkAt: Processor userSchedulingPriority.
shell suspendExecutionUntilRemoved.
```

Automatic Log-Off

Question How do I implement a timeout feature that automatically logs off a user after a minute of inactivity?

Answer This is tricky stuff. Our Multimedia parts do something like this for the same purpose. The approach is to replace the handlers in the EventTable for all mouse and keyboard events, and reset a timer each time one of these is hit. See OsWidget>>mtrTrackAllInputActivity in the subapplication AbtMultimediaAuthoringPM/Win. You have to be careful that the code you invoke in the new handlers is safe for all widgets, and reset the handlers when you are done.

I can also think of another way, although I have not tried it— I suspect if you modified the implementors of *callCallbackList:callData:* to log the time, you′d get a hit whenever there was activity (focus change, keystrokes, mouse active, etc). The drawback of this approach is you′d be modifying the base image. Not a crime, but means *you own it.*

Synchronizing a Visual Part

Question How do I synchronize a visual part with the Smalltalk object it is called from?

Answer If you simply want to wait in a script for a window you opened to be closed, the easiest way is to use

Smalltalk

```
myWindow openWidget.
myWindow suspendExecutionUntilRemoved.
... ″whatever you want to happen after the user closed the window″
```

The user interface process has the same restrictions as Presentation Manager (PM), that is, you must return to the polling loop or the user interface (UI) will lock. Therefore, you cannot block on a semaphore in the UI process; if you do, you′ll lock the UI if the semaphore is not posted quickly. When you execute code from a Workspace with *Execute*, that code is evaluated in the UI process. An example is *(Semaphore new) wait.* If I select and execute this, the UI will lock (press and hold Alt+SysRq, or press the **User Break** button on Windows 95/NT, AIX to unlock). However, if I fork around it, it doesn′t lock because it is not running on the UI process.

I don′t understand why you need to stop the user interface, but if you must, here′s a safe way:

Smalltalk

```
[ ″keep waiting condition″ ]
  whileTrue: [ CwAppContext default readAndDispatch ].
```

This is equivalent to going into a peek loop in PM, that is:

Smalltalk

```
while (″keep waiting condition″) [
   if (WinPeekMsg(...))
    WinDispatchMsg(...);
]
```

This keeps the UI responsive, but I don′t recommend it except for implementing modality as btShellView>>#suspendExecutionUntilRemoved does.

Execution Start

Question How does my VisualAge application start execution?

Answer In the Smalltalk image, the default packaging specification prompts you for launch code. This is text that is compiled into a method and called after the runtime image is started just before dropping into the polling loop. In the VisualAge image, the default packaging specification doesn′t prompt you for any launch code, it sends *runtimeStartUp* (don′t forget the capital U) to all resident applications instead. So you either need to supply launch code or a *runtimeStartUp* method. If you are in a VisualAge image but your application is not a VisualAge application, then you can specify EpStandardRuntimeSpecification which will prompt you for launch code and may end up with a smaller run-time image.

Inheritance of Visual Parts

Tip

When inheriting visual parts, you don′t have the same flexibility as when inheriting scripts. In the case where you override the inherited visual part (or “specialize it”), any subsequent changes to the superclass part are not visible to the subclass, as saving the changes causes the regeneration of the abtBuildInternals and other run-time methods, which are completely overridden by the same run-time methods in the subclass.

Here are scenarios for visual parts:

- Class B is inherited from class A. Class B remains unchanged. Class A is changed. The changes to class A will show up in class B.
- Class B is inherited from class A. Class B is changed or specialized. Class A is then changed. The changes to class A will now *not* show up in class B.

As a result, inheriting visual parts should be thought of as a one-time copy. Therefore, you should be careful to stabilize the visual part class you are inheriting from as much as possible before inheriting.

Multithreading

Question

Does VisualAge support OS/2 multithreading ?

Answer

You can create multithreaded-like applications through the use of Smalltalk processes. Although the processes actually run under a single thread, calls to external functions can run on a thread separate from the base Smalltalk. Although the ability to access all OS/2 APIs does exist, some functions such as semaphores are implemented inside Smalltalk to support its scheduling of processes. When you call external code from VisualAge, all of its processing and the external function′s processing are executed on a single thread. This means that while the function is being run, processing for the rest of VisualAge is halted, including processing for the normal operation of its user interface. To overcome this problem, VisualAge enables you to reroute the execution of any external functions to a separate thread so that normal processing can continue while the function is running.

Global Variable

Question How do I create a global variable?

Answer Since the names of global variables are nothing more than keys in the Smalltalk dictionary, a new global variable is added by adding a new key to this dictionary. The global variable should begin with a capital letter. For example, the following statement adds a new global variable called *MyCustomers* with a value of nil:

Smalltalk

```
Smalltalk at: #MyCustomers put: nil.
```

If you subsequently want the variable, just execute:

Smalltalk

```
MyCustomers := <value>.
```

Even with this method, you have to be very careful that *MyCustomers* is not already an existing global (for example, a class) or you may break the system.

Avoid the use of global variables in applications, if at all possible. The use of global variables violates the principle of data encapsulation. Many times, adding a new object or adding behavior to an existing object will provide the same function as a global variable.

Updating Class Attributes in a Run-time Image

Question If a user modifies the value of a class attribute in a runtime image, is the change automatically saved from one execution to the next? In other words, does the run-time image get updated as the user exits the application?

Answer You cannot (normally) update a run-time image. In order to do save changes, you have to choose other techniques, for example, external files (using the CfsFileStream class).

Class Methods at Run-Time

Question Is there a way to check if a method exists in a class (at run-time); something like *allMethodsNamed: selector*.

Answer The following code returns *true* if my class or a superclass contains a method implementation for the message selector and *false* otherwise:

Smalltalk

```
myClass respondsTo: #myMethod)
  ifTrue: [ ″method exists″ ]
  ifFalse: [ ″method doesn′t exist″ ].
```

Objects′s Knowledge of Reference

Question Review the following code:

Smalltalk

```
| a b |
a := SomeClass new.
b := a.
```

When variable b is assigned the value pointed to by variable a, does the object that variable a is pointing to have any knowledge that it is being passed around? I there a way to inform the object when it is referenced?

Answer The variable b (not the object b) is assigned the value pointed to by the variable a. There are no messages sent and no code is triggered.

One approach to solve this problem is to put a wrapper around the object and let others pass the wrapper around. Your class would look something like this:

Smalltalk

```
SomeClass class>>new
 ^ Wrapper holds: self new

SomeClass>>x
 ^ x

SomeClass>>x: aValue
 x := aValue
```

The Wrapper class would look like this:

Smalltalk

```
Wrapper class>>holds: anObject
  ^ self new holds: anObject

Wrapper>>holds: anObject
  wrappedObject := anObject

Wrapper>>x
  wrappedObject aboutToExecute: #x.
  ^ wrappedObject x

Wrapper>>x: aValue
  wrappedObject aboutToExecute: #x:.
  ^ wrappedObject x: aValue
```

Then,

Smalltalk

```
| a b |
a := SomeClass new.
b := a.
b x: 234
```

causes the object in ′a′ to be sent the *aboutToExecute:* message passing #x:.

Smalltalk Class or AbtAppBldrPart

Question What is the difference in using a Smalltalk class as opposed to a nonvisual part when creating business classes and what are the implications?

Answer One of the main differences between a Smalltalk class and a VisualAge nonvisual part is that you have a Composition Editor with the latter. The Composition Editor allows you to do visual programming within the nonvisual part. This means that you can drop parts on the Composition Editor and make connections between them. Obviously you wouldn′t put visual parts, for example a *text* part, on the Composition Editor of a nonvisual part, but you could place any nonvisual part or Smalltalk class on the Composition Editor and make connections between anything defined in the public interface for those parts.

When you want to create a nonvisual part for which you don′t need composition facilities, you can subclass AbtPart instead of AbtAppBldrPart.

Cleaning Up Unused File Handles

Question If you get a walkback while you have a file open, that file remains open until you exit the image. Is there a way to clean up all of these open files?

Answer The best way to address this situation is to act defensively. You can set up an exception handler so that when a walkback occurs, the file handles are automatically closed. Here is an example:

Smalltalk

```
| inputFile stream tempString |
s := ′′.
inputFile := CwFileSelectionPrompter new
  title: ′File to open′;
  prompt.
stream := CfsReadFileStream open: inputFile.
stream isCfsError
  ifTrue: [ self error: stream message ].
[[ stream atEnd ] whileFalse: [
  tempString := tempString, stream next asString ].
stream close
] whenExceptionDo: [ stream close ].
^tempString
```

Initializing Class Variables

Question How can you initialize class variables in the Trailblazer? I would like to initialize a class variable with a pool dictionary.

Answer Probably *toBeLoadedCode* would be the best place. This string is stored with the application in the manager and is evaluated (compiled) when the application is loaded. Since it is evaluated before any of the application′s classes are loaded, you can define pool dictionaries that are referenced in the application′s classes.

You can store a code string to be evaluated with *YourApplication toBeLoadedCode: ′yourcode′* or bring up a simple editor with *YourApplication abtEditToBeLoadedCode*. The TrailBlazer can modify these directly by selecting an application (Browse Applications) and changing the lower pane property to *ToBeLoadedCode.*

Forking and Cycles

Question I have an application in which I display a window containing a graphic that, through the use of a timer, appears to move. That is, the users see an image move across the window to illustrate that information is being downloaded from a database.

To do this, I have used *fork*, *forkAt:*, and a number of other techniques. The problem is that I cannot get the graphic moving smoothly while the domain is initiated. When I run both pieces of code, it appears that the graphics get held up during the actual query call, and the instantiation gets held up by the timer. The resulting behavior is that the graphics appear but don′t move until the query has finished. Would this be the right approach to do this?

Answer Getting a smooth graphic is difficult when you depend on the scheduling algorithm. Instead, it would be better to use signaling from the work process to the status process. For example, for each step through the database table, you signal the status process.

This is often necessary because graphics update runs at the low priority. By forcing graphics updates at regular intervals, you assure your user is seeing true progress, not simply a bouncing ball graphic that indicates passage of time.

Trying to provide a smooth visual with background processes running is difficult because of two factors: user interface processes try to run when everything is idle, and when they finally do, they run to completion (under the assumption that won′t be long). You can try and override this behavior using *yield* and *CwAppContext default syncExecInUI:* <>, but often the results are not smooth and predictable.

Converting Numbers Using printString

Question In an older version of VisualAge I could perform the following:

Smalltalk

```
1.02 asDecimal printString
```

This would produce a string of ′1.02′. If I try to do the same in Version 3.0 it would produce the following string: 1.02d16.15′. How can I make Version 3 produce a result similar to that of Version 2?

Answer The *printString* method is really for debug purposes, since it is not sensitive to national language support (NLS) considerations (period vs. comma, negative indicator, and so on). Instead, use the locale classes (IBM Smalltalk) or AbtConverter classes (VisualAge). For example,

Smalltalk

```
(AbtDecimalConverter new) objectToPrint: 1.02 asDecimal.
```

or

Smalltalk

```
| stream |

stream := WriteStream on: (String new: 128).
Locale default lcNumeric printNumber: 1.02 asDecimal on: stream.
stream contents.
```

Both of these examples answer ′1.02′ (on a machine where the decimal separator is defined as ′.′).

Garbage Collection of Instances

Question Is there a fast way to get rid of unwanted instances? Suppose these instances are in a linked list so that they are always referenced. You could use the *allInstances* method to find all the instances and make them *become: String new*. However, this is very slow, so another thing you could do is create a class variable to store all the instances, then when it is time, you make them *become: String new*. Even if you cache the instances in the class variable, the process of *become: String new* is still very slow. Now, how can you quickly remove these instances so they can be garbage collected?

Answer The best way by far to solve a problem like this is to find the source. Then you don′t have to try and solve the symptom with things like *become:*. You need to determine the critical reference. A linked list is not a problem. The garbage collector is capable of detecting circular references and will collect such objects if they are not referenced by anything outside the circular list. That is, in your case, something else is holding on to something in that linked list. It is difficult to find out which item in the list is the being referenced.

You might try running *System abtScrubImage*. If your linked list goes away, then see which of the steps freed your object by clearing the dependencies, terminating processes, and so on. If it does not, then you probably have a reference either in class or class instance variables or a reference in the Smalltalk dictionary itself.

An object is garbage collected when there are no references to it that ultimately references something in the global (or old) space. There are two types of garbage collections: *flips* and *sweeps.* A flip is lightweight— if an object is referenced by anyone it is included in the flip (called a *flip* because objects are flipped between two areas of memory). A sweep is more costly because it starts in the old space and traces references to the end. Sweeps take 5-10 seconds on my machine, but rarely occur.

The goal is to maximize the recovery of objects in flips, and minimize the sweeps. It is helpful to eliminate *old space* types of references such as globals and class variables, and to avoid *communes* (circular references).

Objects that survive several flips are *tenured* to old space under the assumption that they′re going to be around for a long time and it is a waste to keep copying (flipping) them. In Version 3, there are command-line options to change or limit the size of the various spaces.

4-Digit Years

Question Evaluating the expression *Date today* yields 04-01-96. What I want it to yield is 04-01-1996. Is there a way I can achieve this?

Answer With VisualAge you can do this by specifying it in the settings for an entry field, or programatically like this:

Smalltalk

```
(AbtDateConverter new) showFourDigitYear: true;
   objectToDisplay: Date today.
```

Or, if you are running IBM Smalltalk, you can also use:

Smalltalk

```
lcTime := Locale current lcTime copy.
lcTime dFmt: ′%m-%d-%Y′.
lcTime printDate: Date today on: Transcript.
```

They are basically the same. The problem with *Date today printString* is it always tries to go with the current locale. If you want to override the locale, you can add an entry to your NLS.CNF file: dFmt=′%m-%d-%Y′. This will override the date format returned by the operating system. See EsNlsImageStartUp>>abtActionTable for the other possibilities.

Fixing Date Years

Question When you enter two characters in the year area of a date field, it defaults to 20? Is there a way to get date years to default to the current century rather than the 21st?

Answer The rules for normalization of the year are as follows:

1. 100 < = y < = 9999 → take as is.
2. 0 < = y < = 79 → assume 21st century (2000..2079)
3. 80 < = y < = 99 → assume 20st century (1980..1999)
4. All others are invalid.

You may want to modify the instance method *normalizeYear:* in AbtDateParse class to customize the default.

ABTPATH

Question Is there any way to find out the ABTPATH value (set in config.sys) in VisualAge for Smalltalk?

Answer There is a method available to strings that will allow you to scan the environment variables for the path indicated. It is called *abtScanEnv.* For the ABTPATH value, you can evaluate:

Smalltalk

```
′abtpath′ abtScanEnv
```

Note!

This method is private, and really shouldn′t be used. However, there is no other way to do this.

Double-Triggering of Changed Events

Question How do I make events work for Smalltalk Classes?

Answer When VisualAge was originally created, it was intended that Smalltalk classes from other environments could be ported directly. To accommodate such an ability, the development staff included some code that automatically triggers the feature connections of a part whenever an attribute changes or an action is performed (presumably, in either case, the attributes of a part might change).

One symptom of this problem is the triggering of attribute-changed events twice whenever any attribute changes or an action is performed. The *signalEvent:with:* code in the setter method for an attribute triggers the event and the special code automatically triggers the event, a second time. Besides being a little inefficient, this has the added problem of signaling all events automatically for us, regardless of the code executed.

To handle all events explicitly, the automatic mechanism can be turned off by overriding the following instance methods for each Smalltalk class:

Smalltalk

```
  featuresAffectedByAction: actionName
     ″For objects that do not signal their own change events. Defines the
     features affected by an action. Do not signal any events, the receiver
     handles all change events itself.″

    ^#()
```

Smalltalk

```
  featuresAffectedByAttributeChange: attributeName
     ″For objects that do not signal their own change events.
     Do not signal any events, the receiver handles all change events
itself.″

    ^#()
```

For both methods, the code has been replaced by the simple return of an empty array (*^#()*), indicating that no change-events need to be signaled. For both methods, answering an array containing #self (a Symbol) signals change-events for all the receiver′s features.

Be warned that with these methods overridden, you have the responsibility for triggering all events yourself (VisualAge will do nothing for you automatically). If you have an attribute that is an OrderedCollection, for example, and you add an object to that OrderedCollection in a method (or script) you have to trigger the changed event for that collection yourself.

The ToDoList class, subclass of Object, keeps a list of things to do in an OrderedCollection in an attribute named *list*. The action *addToList:* invokes the method *addToList:*

Smalltalk

```
 addToList: aString

    ″Adds a string to the receiver′s list.″

    self list add: aString.
    self signalEvent: #list with: self list
```

The last line of the above method explicitly signals the event, telling interested parties that the *list* attribute has changed.

Inheriting Visuals Associated with a View Class

Tip

The visual representation is considered a singular entity. When you create a subclass, you get a one-time snapshot of the visual representation of your superclass, which is then overridden when you save your subclass.

Think of it as a method. You can override the method in a subclass, and you can even copy most of the contents down for tweaking in your subclass. However, depending on how and what you change, you may no longer be able to use superclass, and any changes you make in the superclass implementation will no longer affect the behavior of your subclass.

The reasons you must be very careful are basically the same as the reasons why inheritance is not done automatically. Changes can occur that would leave your view in an inconsistent (or at least confused state). For example:

- View1 has Button1.
- View2 inherits from View1. It adds Button2 whose left edge is attached to the right edge of Button1.

Suppose you now update View1 by removing (or replacing) Button1. What happens to View2? Should it lose Button1 too, or does it still need it? If we dropped it, what does that say about the placement of Button2? This is actually a simplistic scenario. Others can lead to much more severe issues. Connections can be very problematic, especially since all connections are placed in a single method.

Now with the archival code, you can make the decisions. You can override the appropriate methods and keep Button1 in View2, or you can modify the method that creates Button2 and change it so that it does not attach itself to Button1. But be warned, once you start down this path, using the Composition Editor to maintain your views is probably not feasible.

Again, I can′t over-emphasize the risks you are taking by doing this. Does this sound like the voice of experience? Yes, unfortunately it is. If you are not fond of or comfortable with scripting today, then this is absolutely not for you!

Garbage Collector Problems

Question I am having a problem that instances are not removed by the garbage collection process. If I open a class MyView 30 times, then MyView has 30 instances even after I forced garbage collection. I don′t expect other objects to reference instances of MyView. Could there be any other reason that instances of MyView are not garbage collected?

Answer A suggestion would be to start doing this:

1. Save your image (so you can come back to it if necessary).
2. Execute *System abtScrubImage*.
3. Check for instances.
    a. If they are still there, you start to suspect class variables or globals in the Smalltalk dictionary.
    b. If they are gone, then start the image again (so you can get those instances back) and execute the pieces of *abtScrubImage* individually to determine what caused them to finally be garbage collected (that is, was it terminating any processes, reinitializing the widgets, or flushing the dependents?) If it was the dependents, then the most common cause is that people send the method *closeWidget* rather than using the action (which actually uses the method *closeWidgetCommand*).

Integer Representation

Question How can I change the internal representation of Integer from 16 bit to 32 bit?

Answer Smalltalk integers are not stored as 16-bit or 32-bit integers. The′re actually 30 bit integers until that′s not enough, then they grow as long as you need them to be.

Method Tracing

Question Has anyone done any method tracing in IBM Smalltalk? I need to count actual method calls by class for a given transaction. The profiler gives me only a percentage of time spent in each method based on samplings. In essence, what I want to do is: just before a method is pushed onto the stack, I′d like to write out a line to a text file with the class and method name. I currently cannot do this because I get a “source code not available” message.

Answer Yes, it′s certainly possible to do this. The easiest way is to act like a debugger and repeatedly step through the code. It′s slow, but it works.

The source code to some of the system is missing, but that shouldn′t matter, since you don′t need source code to find the class name and selector.

Storing Object of Unlimited Size

Question I am using the Swapper (the class parent of Dumper and Loader) to write my objects to disk. I′ve heard that you can′t write any single object to disk that exceeds 65000 bytes. Is there a workaround to solve this?

Answer The limitation is that no single primitive object can be greater than 64 KB in size. (Its actually a few bytes less because of object headers.) Primitive objects that must be less than 64 KB are Strings, DBStrings, ByteArrays, and Arrays. (Note that each array element is a 32-bit pointer, so that an array with 16*1024 elements will exceed the limit because the continguous storage area for the array itself violates the limit.)

The script at the end demonstrates successful storing and retrieving of object collections whose aggregate size is over 4 MB, but no single element violates the 64 KB limit.

If you have a need to store data that contains large primitive objects (such as very large strings), then you may indeed need to use a commercial database package. (Optionally, if you know where these large objects are and if they aren′t too prevalent, you could break them up and reglue them yourself. You would be the best judge of the appropriateness of this approach.)

Some objects, such as Classes and BlockContexts, cannot be swapped out. If your object graph contains any of these, then you will have other problems. (I believe that sorted collections often hang on to a sort block— this may be a source of trouble.)

Smalltalk

```
| objects dumper totalBytes |
objects := OrderedCollection new.
100 timesRepeat: [
  objects add:
    ((String new: 48 * 1024)
      atAllPut: $x;
      yourself)
].
objects do: [:ea |
  ea at: 40 * 1024 put: $*].
dumper := ObjectDumper new.
totalBytes := dumper totalSizeBeforeUnload: objects.
System message:
  ′total bytes in persistend object= ′, totalBytes printString.
dumper unload: objects intoFile: ′foo.obj′.
objects := ObjectLoader new loadFromFile: ′foo.obj′.
objects do: [:ea |
  (ea at: 40 * 1024) = $*
    ifFalse: [ self error: ′bad obj′]
].
System message: ′Everything went fine.′
```

Collections and #do:

Question I was trying to find out how the method *do:* works when I send it to an ordered collection. I found out that AdditiveSequentiableCollection has the instance variable *implementationCollection* and the method *do:*. This method probably uses *do:* from SequencableCollection. Where does *implementationCollection* get initialized and to what?

Answer Collections in IBM Smalltalk aren′t the same as the collections in other Smalltalk implementations. That is, their public protocol is the same, but the implementation has been significantly optimized. The implementation changes depending on the size of the collection. The *implementationCollection* variable is the reference to the collection which is created based on size and characteristics.

If you look in OrderedCollection class>>new, you′ll see that it actually works with an EsLinearOrderedCollection. SortedCollection is more interesting, since you′ll see in SortedCollection (Class)>>new: it creates either an EsInsertionSortedCollection or an EsHeapSortedCollection depending on if there are 16 or fewer elements. The InsertionSorted implemention does an insertion sort, where the heap does a Heap sort. For fewer than 16 elements, the insertion sort is faster than doing a heap.

Thus, the *SortedCollection new* starts life as an insertion-sorted collection, then changes to heap-sorted collection when we get to 17 elements. The ImplementationCollection instance variable keeps track of the details of what object is really representing the list, and if I remember right, this is a strategy pattern.

Fixed Object Space

Question Does IBM Smalltalk have fixed object space?

Answer Yes. We have a message called *makeFixed* that will protect an object from garbage collection. That is, *makeFixed* relieves the garbage collector from having to flip around lots of long-lived objects (at the cost of manual deallocation).

Parsing Strings with Date/Time

Question Is there a method for parsing a string to get an instance of Date? That is, to do something like this:

Smalltalk

```
′08-23-93′ asDate
```

and receive an instance of Date as a result. The same question applies to Strings representing Time.

Answer You might want to write a method (in the String class) using the existing data converters. Try something like this:

Smalltalk

```
asDate
    ^AbtDateConverter new displayToObject: self.
```

Smalltalk

```
asTime
    ^abtTimeConverter new dispalyToObject: self.
```

One point to watch for with this approach, however, is performance. IBM Smalltalk type converters are intended primarily for handling user input. If you peek into the implementation you see that there is a lot of code for error detection, error handling, and locale-specific format support.

If you convert many strings repeatedly, AbtTimeConverter object creation imposes significant overhead. For example, if you implement

Smalltalk

```
String>>asInteger
  ^AbtIntegerConverter new displayToObject: self
```

then this will result the following code to take up to 3000 milliseconds:

Smalltalk

```
  | a b |
  a := ′66543′.
  b := ′10.7′.
  Time millisecondsToRun: [ 1000 timesRepeat: [ b asInteger. a
asInteger ]]
```

For such cases, reusing a cached AbtIntegerConverter will yield a noticeable improvement. You can change your implementation of String>>asInteger as follows:

Smalltalk

```
String>>asInteger
  ^self asNumber
```

I expect this to be over 30 times faster than the AbtIntegerConverter implementation, but it obviously does not solve your problem with dates and times. If you feel you don′t need all the error handling and locale support, *and* you need performance suitable for loop processing, you might consider writing your own simple parser class, with tokenizing and conversion methods that are optimized for your application.

Identity Test on String

Question I have noticed an interesting fact on testing the identity of two string instances:

Smalltalk

```
| a b |
a := ′myString′.
b := ′myString′.
```

Given the above script, I thought that *a == b* would result in *false* while executing *a = b* gives you *true* as the result. But I found IBM Smalltalk returns *true* as the result of the identity test (a == b) and I am not quite sure why a and b are identical.

Answer The reason for this behavior is due to compiler optimization. Here′s an example. Suppose A and B are global variables:

Smalltalk

```
A := ′my string′.
B := ′my string′.
A == B
```

If the code above is evaluated as a group, you′ll get *true*. If you evaluate each line separately, A == B will return *false*.

When you create strings with the single quote, you are creating a literal string. Literal strings cannot be modified so as an optimization the variable a and variable b actually point to the same object. This would usually create problems, but since it′s a literal you can′t modify either one of them, so it doesn′t really matter that they′re the same object.

For example, if you try this code snippet the results will be what you expected.

Smalltalk

```
| a b |
a := 9 printString.
b := 9 printString.
a = b        → true
a  ==  b     → false
```

Replacing Character in String

Question In some other Smalltalk implementations, I can replace characters in a String by using the method:

Smalltalk

```
String>>at:put:
```

In VisualAge, if I try the following:

Smalltalk

```
′dog′ at: 1 put: $D
```

I get a walkback with the following error: “Store into read-only object.” What is happening? And what is the right approach for replacing characters in a string?

Answer In your case, ′dog′ is a literal. (Some other examples of literals are 3.14, #D101, and 1234). You should *never* try and replace characters in a literal string. This is one of the most common sources of errors in other Smalltalk platforms. IBM solves this problem by preventing it altogether. If you want to do this, make a copy of the string first, like this:

Smalltalk

```
′dog′ copy at: 1 put: $D
```

Padding Strings

Question Is there a method that allows me to pad a string with *n* number of characters to the left or right?

Answer EsString>>abtPadWith:upToLength:onRight: is the method you are looking for. Here are a couple of examples of how the method works:

Smalltalk

```
'1' abtPadWith: $0 upToLength: 5 onRight: false    →    '00001'
'abc' abtPadWith: $d upToLength: 4 onRight: true     →    'abcd'
```

Trimming a String

Question How do I trim a string padded with spaces?

Answer If you want to remove only the leading and trailing spaces, use *trimBlanks.* If you want to remove all the white space, use *abrWithoutWhitespace.*

Inhibiting List Selection

Question I need to be able to inhibit list selection when users click on specific items within Cw... and Ew... list widgets, but there doesn't seem to be a public interface for doing so. I know, native *aboutToChangeSelection* events exist down on the GUI platforms, I've implemented them on Windows (not sure about OS/2). The documentation I've read make no mention of any callback return values that could be used towards this end. I'd like to avoid subclassing or extending classes in OSwidget layer if possible.

Answer The Ew... list widgets (EwList and subclasses) have a *selectionPolicy:* interface (have a look at the comment). This should enable you to do what you want.

Resizable Table Rows

Question

I am using VisualAge Professional for OS/2 and WindowBuilder Pro for OS/2 (3.0) and am having difficulty trying to use the multiline text edit policy within an EwTableList widget. I hope someone can come up with a simple solution for what I am trying to do.

We have an interface that is supposed to look and work much like a contemporary spreadsheet. However, we intend to display information in cells that may be several lines long. I have created an EwTableList widget. Within one of the columns, I use an EwTextEditPolicy, passing the *editMode* as multiple lines. Everything works great! I can enter as many lines as I wish.

The problem I have is what occurs when I leave the cell. The cell displays only a single line of text. Ideally, I would like to expand the edited row to a size capable of displaying the complete text. Alternatively, I would like to allow users to resize rows in the same way they can resize columns. I have tried messing with *itemHeight* settings, but they appear to affect every row within the table.

Answer

I am told that you can create dynamic row heights via setting the *itemHeight* to nil (do this in *preInitWindow*). This feature is undocumented, but I have it on good authority that it works. Setting *itemHeight* to nil causes the table to base each row height on the data in that row. I don′t think it allows you to have user-resizable rows. That would probably require subclassing EwTableList itself. You might want to trace through the code in EwTableList (and its superclasses) that deal with *itemHeight.* It is very illuminating.

If you set the *itemHeight* to nil, the EwTableList will send the *ewHeightUsing:* method to the value in each of the cells. Your contents are EsStrings, therefore they should already pick up the implementation there where for each line feed the height is increased by the size of the font.

Adding wmUser Messages

Question

How do I add a *wmUser* message? The C DLL that I′m interfacing to sends wmX messages to my Smalltalk image for processing. I′ve figured out how to add missing Presentation Manager wmX messages via the OSWidget method *eventTableAt:put:with:*, but not *wmUser* messages.

I could subclass OSWidget, add my new event and then reimplement *windowProc:with:with:* to check for my event and process it. Is this the correct or preferred way?

Answer When you say reimplement *windowProc:with:with:* you mean in your subclass, right? This is certainly the most efficient way of doing this. The other option would be to add an entry in the event table and extend OSWidget to add the new event. However, the event table is currently a 1023 element array and arrays can′t be grown, so in order to get WM_USER (4096) in there, you′d need to replace the current array with an all-new one.

New Operating System Event

Question How do I register a new operating system event?

Answer To register a new event use: OSWidget method *eventTableAt: anInteger put: anObject* where *anInteger* is the event number and *anObject* is a symbol (that is, your selector name).

Class Instance Variables

Question Are there class instance variables in IBM Smalltalk?

Answer Yes, they are there. Here is a sample:

Smalltalk

```
Object subclass: #AbtNLSCoordinator
  classInstanceVariableNames: ′currentCodePage ′
  instanceVariableNames: ′′
  classVariableNames: ′CharacterSet CurrentLocaleMapping
     LocaleMappingCharacters ′
  poolDictionaries: ′AbtConstants CfsConstants AbtPrimitiveNLSStrings
     AbtMsgBase ′
```

Class Naming

Question Are there any performance considerations for the length Smalltalk class names and instance variable names? In other words, will a system with a bunch of 40-character names run slower than a system with a bunch of 4-character names. In DB/2, shorter table names relate to faster response time because of more table entries in the hash table.

Answer First you need to understand one of the very important concepts in Smalltalk, the difference between instance identity and equality. Two instance are “==” if they are the same object, which is to say they occupy the same memory location. But “=” may be different. An expression *a = b* answers *true* if the method for “=” in the receivers class answers *true.* That method may be redefined as appropriate for each class. The default method for “=” is the one in Object which is the same method as “==.” But for Strings you will find a different method that checks that the characters are the same.

Symbol is a subclass of String that maintains a SymbolTable as a class variable. It is designed so that only one instance of Symbol has a given spelling. This means that “=” can be redefined (again) to be the test for object identity. The names of classes are Symbols and all the method selectors are Symbols. All the lookups, and the like, can use object identity comparison, just checking the address rather than comparing all the characters. So the length of the names doesn′t matter at all.

Instance variable names are Strings, but they are used only by the compiler. Once compiled, the methods refer to instance variables by slot number rather than by name, so the strings are not compared at run time. There is obviously some size impact that may affect performance to some degree, but only if there is an issue of total available memory.

There is an optimization in IBM Smalltalk that can fool you into thinking that part of what I wrote above is wrong. We would expect that

Smalltalk

```
′abc′ = ′abc′
```

answers *true*, but

Smalltalk

```
′abc == ′abc′
```

should answer *false.* It answers *true* instead?

In IBM Smalltalk, the compiler creates literal strings as read-only objects and makes an optimization to use the same instance for any strings with the same spelling which are compiled in the same method or workspace expression. To get the right answer, we need to make a copy

Smalltalk

```
′abc′ == ′abc′ copy    → false
```

but

Smalltalk

```
#abc == #abc copy    → true
```

Anyway, trying to place any sort of naming restriction on Smalltalk to meet database limitations is not the right approach.

Reusing the Settings View

Question Is it possible to reuse the VisualAge template for the settings view? I am creating custom parts and would like my settings to have the same look and feel as other VisualAge parts.

Answer Are you asking if it′s technically possible, or legally possible? I believe the answer to be “yes” to either question. For the technical side, you can generate archival code and do a find or replace to change the class name to your settings view, and then file that code in.

Exception: #doesNotUnderstand

Question I need to know the IBM Smalltalk way of specifying an exception handler for *doesNotUnderstand*. In Digitalk Smalltalk, it used to look like this:

Smalltalk

```
[self foo]
    on: MessageNotUnderstood
    do: [ self doesNotUnderstandHandler ]
```

What is the IBM Smalltalk equivalent of this?

Answer If you look at the method *doesNotUnderstand:* in Object, you see:

Smalltalk

```
doesNotUnderstand: aMessage
    ″Report to the user that the receiver class
     for superclasses do not implement any methods
     for the message selector contained within the
     argument aMessage.″
^self error: (self doesNotUnderstandError: aMessage)
```

The upshot of this is there is no exception specifically for the *doesNotUnderstand:* method. You would have to catch it by catching all errors, as in this code snippet:

Smalltalk

```
| i |

[ i do: [ :each | each become: nil ] ]
  when: ExError
  do: [ :signal |
    System message: signal argument.
    signal exitWith: nil ]
```

This code is interesting in many ways. You may want to play with it for some time; for example, try to initialize *i* to something that does or does not understand do:, and also try to *exitWith:* something a little more interesting than nil.

Of course, instead of catching all exceptions, you can create the new exception for *doesNotUnderstand:*, and change the method to signal that exception instead.

Creating Classes from Smalltalk Script

Question What′s the proper way to create a class in Smalltalk code (the name is entered in an entry field) and adding a few methods? I′m running in the team environment. I′ve tried various combinations of Compiler>>#evaluate: and Compiler>>#compile:in: but have the following problems:

1. The class I create has no owner. No biggie, but I thought that *EmUser currentUser* would own classes created with *evaluate:*.
2. Methods I compile seem to lose their source code. I stole the code from the SubApplication class method for creating the method for starting run-time views (don′t remember the method name) so I don′t see what I′m doing differently. Could have something to do with the ownership question though.

Answer I see you enjoy this kind of fooling with the environment. Anyway, you can try this after calling Compiler class>>#compile:inClass:

Smalltalk

```
compiledMethod methodClass
    install: compiledMethod
    asPrivate: asPrivateBoolean
    withSource: sourceString
    ifNewAddTo: compiledMethod methodClass controller
    categorizeIn: #(′all′ ′the′ ′categories′ ′you′ ′want′).
```

However, I suggest you instead call *compile:notifying:ifNewAddTo:categorizeIn:* or *compileAsPrivate:notifying:ifNewAddTo:categorizeIn:*, which is the preferred method.

Keyboard Event Handler

Question I need to be able to capture the following key pressed events:

- Shift + F10 Activate Pop-up Menu
- Alt + F6 Switch between associated windows

I′m able to capture single key pressed events but not multiple ones. Is there a way to use IBM Smalltalk to capture multiple key-pressed events other than menu accelerator keys? Does VisualAge provide a mechansim for capturing multiple key pressed events?

Answer Yes it is possible to capture multiple key pressed events. See below for a tehnique to do this.

As to Shift+F10 and Alt+F6, on the Windows 95 version of VisualAge, these two particular events are not trappable; that is, they do not generate Cw event-handler callbacks. I suspect this is because the native widget is trapping these or they have operating system significance. There is some interesting protocol with *interceptEvents:* and CwOverride shells but with the lack of any good documentation I am in the dark on this one.

For the multiple key-pressed events, I assume you are using a method along the lines

Smalltalk

```
aWidget
  addEventHandler: KeyPressMask
  receiver: receiver
  selector: #keyPressed:clientData:callData:
  clientData: nil.
```

This will get you a callback each time a key is pressed for *aWidget.* Implementing the three keyword method *keyPressed:clientData:callData* on the receiver object results in calling it each time a key is pressed. If you look at the *callData* object (the third argument to the message), it is an instance of a CwKeyEvent. CwKeyEvent has a state that can indicate whether keys such as Alt, Shift, or Ctrl were pressed together with the original key character. It is a little esoteric, but the state is an integer that represents a single byte, bits of which are set on or off depending on which key was pressed. Fortunately, CwConstants contains pool dictionary entries that identify the masks and the best thing to do is to hide this complexity on an extension of CwKeyEvent:

Smalltalk

```
CwKeyEvent>>#isAltPressed
  ^( self state & Mod1Mask ) ~~ 0

CwKeyEvent>>#isShiftPressed
  ^( self state & ShiftMask ) ~~ 0

CwKeyEvent>>#isCtrltPressed
  ^( self state & ControlMask ) ~~ 0
```

In your callback method, if you want to see whether Shift and F10 are pressed together, the code would be along the lines

Smalltalk

```
keyPressed: aWidget clientData: clientData: callData: callData
 ( callData isShiftPressed and: { callData keysym = XKF10 } )
    ifTrue: [ ... ].
```

This assumes you can see the pool dictionary CwConstants in your class.

Changing Object Class

Question Does IBM Smalltalk provide a way to change the class of an object, something equivalent to ObjectWork′s *changClassToThatOf:*?

Answer There is an undocumented primitive to do this, but you have to write your own method. Try something like this:

Smalltalk

```
Object>>changeToClass: aBehavior
    ″Change the  Class of the receiver to aBehavior″
    ″Note -- does not verify compatible class shape″
    <primitive: VMprObjectClassColon>
    ^self primitiveFailed
```

Cleaning Up the Image

Question During development, my image has grown far too big. Is there an easy way for me to clean up the image?

Answer The image has probably grown for two reasons:

1. You have loaded a lot of applications in it, or
2. You have a lot of instances alive in it.

I would first consider unloading applications you are not working with or starting with a clean image (the one you originally started from) and loading only the applications you need. In the second case, if you know the class of the instances that are bothering you, you could try

Smalltalk

```
aClass allInstances do: [ :i | i becomes: nil ]
```

That would free the memory used by these instances.

Finding Methods Containing a String

Question How would one locate a string within a given class or application in IBM Smalltalk? I thought this would be trivial, given the rich set of text comparison tools, but I can′t find any tools or menu selections to make this easy. For example, I want to search through application A to find all methods that contain the text string B. Does anyone have a swatch of code I could use for this? I′d like to extend the Envy browsers to make this function accessible, but code that one can evaluate in a workspace would be wonderful.

Answer If you are looking for methods referencing a given literal, sending

Smalltalk

```
System allMethodsReferencingLiteral: 3.14
```

will open a References browser listing all methods referencing 3.14.

The following code searches for a literal string in source of all the methods of all classes. (Change Object allSubclasses.... to something more meaningful to reduce time and hits):

Smalltalk

```
| str pat methods |
methods := OrderedCollection new.
str := ′Elephant′  ″Search for this string″
pat := Pattern new: str.
Object allSubclasses do: [ :cls |
  cls selectors do: [ :sel |
    (pat match: (cls sourceCodeAt: sel) index: 1) notNil
      ifTrue: [ methods add: (cls compiledMethodAt: sel)]
  ].
  cls class selectors do: [ :sel |
    (pat match: (cls class sourceCodeAt: sel) index: 1) notNil
      ifTrue: [ methods add: (cls class compiledMethodAt: sel)]
  ]
].

MethodBrowser new
  label: ′References to: ′, str;
  literal: str;
  openOn: methods
```

Free Disk Space

Question Does anybody know how to retrieve free disk space information on Windows platforms?

Answer The GetDiskFreeSpace function call returns the following information:

1. Number of sectors per cluster
2. Number of bytes per sector
3. Number of free clusters on the disk
4. Total number of clsuters on the disk

Pointers to variables where the above information can be returned must must be passed to the GetDiskFreeSpace function. To get the total amount of free disk space, you must multiply the sectors per cluster by the bytes per sector by the number of free clusters.

Listed below is code that will call the GetDiskFreeSpace function, calculate the amount of free space on C:, and display that value in a message prompter.

Smalltalk

```
| oscall freeSpace sectorsPerCluster bytesPerSector
  freeClusters totalClusters rc |

oscall := OSCall new.
sectorsPerCluster := ByteArray new: 4.
bytesPerSector := ByteArray new: 4.
freeClusters := ByteArray new: 4.
totalClusters := ByteArray new: 4.

rc := oscall getDiskFreeSpace: ′C:′
  lpSectorsPerCluster: sectorsPerCluster
  lpBytesPerSector: bytesPerSector
  lpFreeClusters: freeClusters
  lpClusters: totalClusters.

freeSpace := (sectorsPerCluster abtAsInteger) *
  (bytesPerSector abtAsInteger) *
  (freeClusters abtAsInteger).

CwMessagePrompter message: ′Free space on disk C: is ′,
  freeSpace printString.
```

Using LookupTable Instead of Dictionary

Tip

I just did some timings of LookupTable and Dictionary. LookupTable is faster, typically about 15% to 20%, but the difference is greatly influenced, sometimes dwarfed, by the selection of the key class.

The table below shows the test code run with both Dictionary and LookupTable and four kinds of keys. In each case a random number between 0.0 and 1.0 is selected. The number is used, sometimes after is is transformed, as a key.

	Strings	**Floats (0..1)**	**Nice Ints 1..1000**	**Bad Ints 1..huge**
Dictionary	136	22500	60	61
LookupTable	120	18600	50	51

The keys are formed like this:

Strings A printString of the random number

Floats The raw random number in the 0.0–1.0 range

Nice Ints An integer in the 1–1000 range; 1000 is the collection size.

Bad Ints An integer in the 1–7654000 range (arbitrary range)

Here are some observations:

- Integers happen to be extra nice keys because they are their own hashes and seem to distribute well over the dictionary and table size.
- Strings hash quickly to a nice integer.
- Floats hash by truncating to an integer and answering that integer. That is always zero for the selected keys. (A fact I didn′t know until I tried to explain the awful results.)
- (Note that the integer tests actually run with approximately 640 keys, because of the way random keys are generated, whereas the float and string tests have 1000 keys. Cases with 100 integers run with a result of about 100 and are thus not twice as good as strings as might be assumed from the raw data in the table.)
- Symbols also make great dictionary keys, because each has an associated unique integer hash. However, it is not good practice to create a large number of new symbols; use symbols only for method names.

The following test routine is almost the same for all tests. It generates up to 1000 keys and inserts them into either a dictionary or a table. It also builds a set that contains the keys. The timing test consists of accessing each key twice, once to fetch the value and once to set the value.

Smalltalk

```
| randy coll set n slot key |
n := 1000.
coll := Dictionary new.          ″ ← THIS LINE CAN CHANGE ″
randy := EsRandom new.
set := Set new.
n timesRepeat: [
   key  := randy next.
   slot := key printString.     ″ ← THIS LINE CAN CHANGE ″
   coll at: slot put: randy next.
   set add: slot ].
^ Time millisecondsToRun: [
   set do: [ :e |
      coll at: e put: (coll at: e) ]
   ]

136 137      ″ ← This line shows one or more timing results  ″
```

In the following, the example above would be shown as:

Smalltalk

```
coll := Dictionary new.
slot := key printString.
 136 137
```

Comparison with Random Strings As Keys

Smalltalk

```
coll := Dictionary new.
slot := key printString.
 136 137

coll := LookupTable new.
slot := key printString.
 119 121
```

Comparison with Floating Point Numbers As Keys

Smalltalk

```
coll := Dictionary new.
slot := key.
 22480

coll := LookupTable new.
slot := key.
 18548 18877
```

Comparison with Integers in the 1–1000 Range As Keys

Smalltalk

```
coll := Dictionary new.
slot := (key * n) truncated + 1.
 59 59 60 65

coll := LookupTable new.
slot := (key * n) truncated + 1.
 51 49 53
```

Comparison with Integers in the 1–7654000 Range As Keys

Smalltalk

```
coll := Dictionary new.
slot := (key * n) truncated * 7654 + 1.
 64 61 60

coll := LookupTable new.
slot := (key * n) truncated * 7654 + 1.
 51 51
```

Concatenation and Streams

Tip

Strings can be formed from smaller strings by using the concatenation message (comma) or streams. Here is a comparison of the two:

Smalltalk

```
lumpSomeStrings: arrayOfStrings

  | bigLump |
  bigLump := String new.
  arrayOfStrings
    do: [ :aString |
      bigLump := bigLump, aString ].
  ^ bigLump
```

This is simple and straightforward, but note that each assignment to bigLump involves copying the previous contents of bigLump to a new place and copying aString behind it. When bigLump becomes long, there is a lot of copying going on.

An alternative and better solution when strings become large uses streams and works like this:

Smalltalk

```
lumpSomeStrings: arrayOfStrings
  | aStream |
  aStream := WriteStream on: String new.
  arrayOfStrings
    do: [ :aString |
      stream nextPutAll: aString ].
  ^ stream contents
```

Basically, a stream continues to add on new stuff behind previous stuff in a large working collection (called the stream contents). If the working collection fills up, the stream allocates a bigger collection and copies the data to it.

It is possible to prevent all copying by specifying the size of the collection:

Smalltalk

```
lumpSomeStrings: arrayOfStrings
  | aStream length |                          ″← changed″
  length := arrayOfStrings                     ″← new″
    inject: 0                              ″← new″
    into: [ :len :str | len + str len ].        ″← new″
  aStream := WriteStream on: (String new: length). ″← changed″
  arrayOfStrings
    do: [ :aString |
      stream nextPutAll: aString ].
  ^ stream contents
```

(The *inject:into:* method is not often used, but it should be! Here it is used to sum the lengths of all of the strings in *arrayOfStrings* and in one simple expression!)

All of these examples answer the same string (at least, accoring to #=).

Comparing Floating-Point Values

Tip

This expression answers *false,* but it looks like it should answer *true*:

Smalltalk

```
(45.0 * 1.15 * 10.0) = (45.0 * 10.0 * 1.15)
```

So, is it a problem? No. Why?

I used a *printHex* method to look at *raw* floating point values:

Expression	Result	Notes
1.5d0 printHex	′1.8000000000000e0′	Test case that converts evenly

The value is in hex; the exponent is hex and is the number of bits to shift the value. For the other values, it shows:

Expression	Result	Notes
1.15 printHex	′1.2666666666666e0′	Not exact conversion
45.0 printHex	′1.6800000000000e5′	Exact conversion

Expression	Result	Notes
10.0 printHex	′1.4000000000000e3′	Exact conversion

Note how 1.15 seems to run forever. It is not represented exactly in binary. Now, let′s look at the results of the calculation:

Smalltalk

```
(45.0 * 1.15 * 10.0) printHex ′1.2BFFFFFFFFFFe9′
(45.0 * 10.0 * 1.15) printHex ′1.2C0000000000e9′
```

Note how the first expression answers a value just a tiny bit smaller; adding a one bit at the rightmost position makes the first equal to the second. It′s just off the minimum value that a float can be and still be different.

Such are the hazards of comparing floats for equality. The top three rules in using floating point values are:

1. Do not compare floats for equality.
2. Do not use floats for monetary computations.
3. Do not compare floats for equality.

Immutable Objects and Identity Objects

Tip

There are two similar terms that are often confused: *Identity* and *Immutable*. Here is a definition of both:

- Identity object

  An object where each object with the same value is the same object. Identity objects include characters, small integers, true, false, nil, and symbols.

  The reason why each of these objetcs is an identity object differs, but the result is the same. If a and b refer to identity objects, and a=b is *true,* then a==b is always *true* as well. Thus, (2+2)==(1+3) is always *true.*

- Immutable object

  An object whose contents are never modified. Instead of modifying the contents, a new object is answered.

Two objects with the same value are different objects: 2.0 + 2.0 produces a 4.0 that is different from the 4.0 produced by 1.0+3.0; both have the same value but are not the same object.

Immutable objects include large integers, floats, and anything else you happen to code to be immutable. If a and b refer to immutable objects, and a=b is *true,* then a==b may or may not be *true* as well.

Identity objects and immutable objects are not opposite terms. In fact, all identity objects are immutable objects. Here is some related information:

- Mutable objects

  Mutable objects have internal states that can be changed. Most objects are mutable, including most of the objects people typically write.

  Points, for example, are mutable. One can write (p x: p x + 1) to increment the x value. The result is stored in the same instance of point. No new point is answered. All variables that pointed to the point still point to it with its new value.

- Small integers

  Small integers are immutable because they are encoded directly in the object pointer. This is an optimization and not something esoteric in the basics of OOP or Smalltalk. It is a hack, but one that provides a huge gain in performance.

  Ideally such optimizations should go unnoticed, and this one almost does. It requires some bit of odd code to tell when an integer is a small integer, or where the boundary is with large integers. (See *IBM Smalltalk: The Language* for some code.)

  The encoding in current Smalltalk systems usually takes 2-3 bits of a 32-bit address for flags. With two bits, 30 bits are left for encoding an integer. However, this internal representation is never accessible to Smalltalk code. Users, even those who dig into the innards, see just integers; the class may differ depending on the size, but the direct encoding into an object pointer is never seen. This is also true with characters, which are often encoded in the object pointer as well.

- Adding

  Now, when it is time to add, say 1, to a small integer, the encoded value is converted into a regular hardware integer value, 1 is added, and the result is converted back into some kind of integer. The answer is always a new object.

Characters cannot be added to, but they do have a value that can be converted to an integer. This integer can be incremented and converted to a different character (if it is not too big).

In neither case is the immutable object changed internally; the result is always a new value, and all variables that hold the original value still hold it.

Chapter 5. ENVY

In this chapter, we cover the ENVY programming environment.

Working with the Team Environment

Question We expect to have to put in quite a number of hours on our VisualAge for Smalltalk project on a tight schedule. We are running in the Team environment at the office. We would also like to do some development at home. We have access to our Team server library drive remotely via LAN Distance and Netware. Rather than have to go through all that is required to export the application or its parts, we′d like to simply have a client environment at home as well as at work. The license agreement seems to allow this so, from a technical point of view, how should we set up the environment? Are there any issues with respect to synchronizing the home client image with that at the office? Do we need to make our image consistent or recache the pointers?

Answer Principally speaking, I do not see how your home setup differs from your office setup. Basically you share the same library across your remote LAN extension. Since nothing is really locked in the library, there should be no problem loading your editions at home.

The only risk I can think of is working on the same edition of a class in two or more images, since every method save implies a release of the current method edition into the class. Therefore, you might run into problems when trying to version or release the class, since you may not have all the currently released methods in your image. But even then a ″reload current″ on the class will do the trick.

You also may want to consider taking your image with you so that you are working with the same open editions. The alternative is to version your code before leaving home (or leaving the office), and then just load it from the library when you arrive.

Actually, there is really no need to cart the image back and forth. Here is a simple process that you can follow to do nontrivial development at both the office and at home. This process requires no new software or hardware. It typically takes me 15 minutes or less to do the versioning and exporting and zipping required to prepare my work for travel:

1. Bootstrap your home system with a starter manager and a matching image. You can create the manager by exporting the latest version of each configuration map or application loaded in your image.
2. Going from home to work: Version off the day′s changes (up to the application or map level) and export them to a new library. Zip it and put it on diskette. Import the code into the office manager when you get to work. Load your changes into the image and start working.
3. Going from work to home: version off the day′s changes (up to the application or map level) and export them to a new library. Zip it and put it on diskette. Import the code into your home manager when you get back home. Load your changes into the image and start working.

Don′t worry about having Version 999 or whatever. The only thing that matters is the name that you use to version the code when you ship. If this is a psychological barrier, then start your version numbers at 0.1 and let them grow to 0.nnn. When you are ready to ship, version it as 1.0. Versioning often is a good habit. It makes for excellent checkpoints. Versioning is a tool. It is your friend. Use it.

Shared Library Password Protection

Question How do I protect certain components of my library against unauthorized access? For example, in a university environment, rather than sharing code, students need to hide their work from other students.

Answer The VisualAge for Smalltalk Professional Server is meant for sharing information, but it does have password control. You should be able to define a password for each student to limit their access to only their work in a common repository. Then define an application for each student where they and their professor are the only group members. This is covered in the *VisualAge for Smalltalk User′s Guide*.

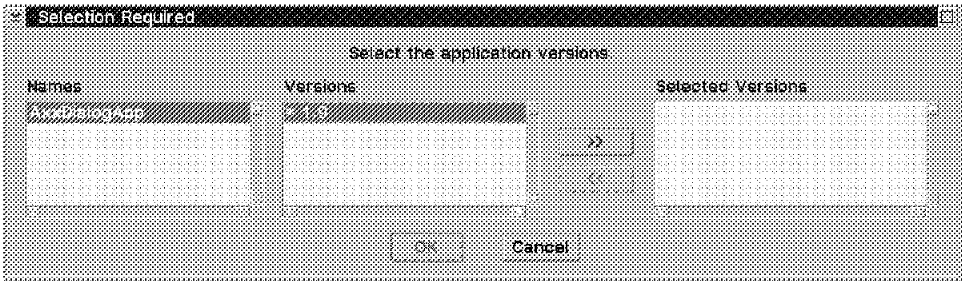
Libraries

Tip The Professional version of VisualAge for Smalltalk permits the creation of libraries. These libraries are useful for moving applications from one installation to another. They can also be used for making backups or for providing a reusable library to share between development installations.

Import and Export

Question How do I import and export applications from a library file?

Answer In the VisualAge for Smalltalk Organizer, open the menu titled Applications. Select submenu **Import/Export**. Select menu item **Import Applications...** In the resulting file dialog, locate the library file (with .dat extension) and click **OK**.



The contents of the library file will be presented. Select the application and version that you want to import and click the >> button (this includes the selected application and version in the list of things to import). Click **OK**. The version(s) of the application(s) you selected will be imported into your current library, but they will not be loaded into your image.

Loading Applications

Question How do I load applications from the library?

Answer In the VisualAge for Smalltalk organizer, open the menu titled **Applications**. Select submenu **Load**. Select menu item **Available Applications...**

A list of applications that have not been loaded in the image will be presented. Select the application and version that you want to load and click the >> button (this includes the selected application and version in the list of things to load). Click **OK**. The versions of the applications you selected will be loaded into your image. Save the image.

Moving Components Between Libraries

Tip You can move components between two libraries directly without exporting or importing them. For example, to move your applications from a Version 3.0 Manager to a Version 3.0a Manager without exporting or importing, do the following:

From within your Version 3.0a image that is connected to the Version 3.0a manager, choose **Import**, and specify the Version 3.0 (old) manager as the file to import from. It will then bring up the list of all applications in the manager. You can choose all that you need to import over.

Remember: Your Manager is essentially a normal .DAT file.

Archival Code

Question I′m using VisualAge for Smalltalk Professional. I use only import/export, and not filein/fileout. Do I still need to generate archival code?

Answer No, you don′t need the archival code. The primary purpose of archival code is for sharing code between standalone images. With the Professional product, import/export is much preferred. So, with VisualAge for Smalltalk Professional, there may be only two cases when you might want archival code:

- If you want to move code to a ″Standard″ image
- If you are creating several copies of a part and want to modify the filed-out version of it to file it in.

Identifying the Released Version

Question How do I see which version is released? I know how to determine which version of a class is currently loaded in my image, but how do I determine which version is the released one without having to Load Released and then examine which one responds?

Answer There are many ways to find out which class version is the released version in an application. One way is to browse the application editions. From the Application Manager, select the application you want, then browse the application editions. Select the edition you want to see; the class versions listed are the released versions.

Change Management for Flat Files

Question I am using VisualAge for Smalltalk Professional for change management of my Smalltalk code. However, I also have some flat files that are used as part of the product as well (metadata and the like). Does VisualAge for Smalltalk Professional have any capability to manage these too, or am I forced to use a different approach for my non-Smalltalk product files?

Answer If your team is comfortable with the VisualAge for Smalltalk Professional version control system, there′s nothing stop you from saving the contents of those files as Smalltalk comments in some method. VisualAge for Smalltalk Professional manages applications, classes, and methods by storing them in its library file. It also expects the methods to compile under Smalltalk but comments don′t have to compile.

You can easily extract the text of methods and write them out to files again in Smalltalk. Besides, other revision-control systems are doing just that in their own way.

A much more sophisticated approach is to look into the support provided by the LibraryObjects application. This allows you to associate any object that can be persisted by the ObjectDumper with a given class in an application. All that you need to add to this is some trivial CommonFileSystem code to read your files into a byte array (for storing) and some code to write the byte array back out to a named file (for restoring).

This gives you the version control that you are looking for. It has the added benefit that your files can be binary or text. (You can′t store binary files in the comment fields unless you use something like uuencode to turn them into a stream of hexadecimal charaters.)

Date Stamp for Versioned Classes

Question An editioned class or application has a date-created stamp. What about after you′ve versioned the class or application? How can I see from the browser or manager what the date of the versioning is?

Answer Versions are really just editions, but in a versioned state. Therefore, even though you are looking at a versioned edition, you can still find out its timestamp, but it is the timestamp when the ***edition*** was created.

If you want the version name to include the timestamp as part of its signature, do the following: From the Transcript menu, select **Smalltalk tools**

→ **Open Preferences Workspace**. Scroll down a bit to find the following line within the Configuration Management section:

Smalltalk

```
EmTimeStamp showVersionTimeStamps: false
```

Change *false* to *true*, swipe the line, and execute.

Note

This still shows only the timestamp of the edition, that is, when the edition was created, not when it was versioned.

Applications and Subapplications

Question I am not quite sure when to use applications and when to use subapplications. Would someone please explain the differences and the criteria generally used for determining when a set of classes are considering a subapplication?

Answer Subapplications are used for functional grouping, or for class extentions that need platform-specific functions or supports. By default, when you load an application, all its subapplications will also be loaded.

Especially for platform specific support, an application will have some configurable expressions to specify which of its subapplications should not be loaded on a certain platform.

Connecting to a Cloned Library

Question Can I connect an old image to a cloned library?

Answer Yes, you can, but recache your pointers!

Never connect an old image to a cloned library without having recached your pointers first! Otherwise your cloned repository could get corrupted.

Reloading an Application

Question How do I, in code, reload the currently loaded edition of an application?

Answer Try this:

Smalltalk

```
EmClassDevelopment imageBuilder loadApplications:
  (OrderedCollection new add: <App Name>; yourself)
```

Loading Previous Editions of Application

Question Is there a fast way to load all recent class editions of the entire application? Each time, when I get a native image from the server and don′t create a new version of the application in advance, I have to load the last edition of all classes manually. Is there a faster way of doing this?

Answer One way you can manage your code so that the transition to a new image is less painful is to create a configuration map pointing to the application editions you are interested in. If you do this, you can load in all your applications with one operation.

Deleting an Application

Question Can I, if I have installed VisualAge for Smalltalk Professional, delete an application by simply selecting delete from the applications menu on the organizer?

Answer Deleting really just deletes an application from a configuration map, it does not delete the application from the library. If your application is not yet associated with any configuration map, the best you can do is unload it from your image and try to forget it exists.

Creating a new application does affect the library manager. Unfortunately, there is no way to remove your application from the library except to purge your application and clone the library.

Another way would be to start with a new manager, and import the applications that you want from the old into the new.

Question How do I delete an application when there is no configuration map?

Answer Unload the application. (Check the ″How Do I...″ help from Transcript). Deleting really just deletes an application from a configuration map, it does not delete the application from the manager. In your case, your application is not yet associated with any configuration map and the best you can do is unload it from your image and try to forget it exists.

Creating a New Application

Question Does creating a new application immediately write to the library manager?

Answer Yes. You can see this by creating an application and looking at the list of applications in an Applications Editions browser. The name of your new application appears in the list. Creating a new application does affect the library. Unfortunately, there is no way to remove your application except to purge it and clone the manager. Or, as we do, start with a new manager and import the applications that you want from the old into the new.

Removing an Application

Question How can I remove an application from the loadable applications list?

Answer You, or your library supervisor, can purge the application if it is versioned using an Application Editions browser. (See the *IBM Smalltalk User′s Guide*.) Make sure that you unload the application version before purging it.

Deleting Old Versions of Classes

Question Is there a way to delete old versions of classes from user-defined applications?

Answer No, a version is immutable. Once you version the application, you have to open a new edition to make changes. When we are restructuring applications, we open an edition, move or delete the affected classes, then version the application as obsolete or “do not use.” If the intention is to reduce the manager′s size by deleting the classes, be aware that deleting doesn′t shrink the manager, only cloning does.

Getting Rid of Damaged Classes

Question How do I get rid of damaged classes when my image refuses to let me delete the class?

Answer Version and release your classes, applications, and configuration maps. Get a fresh image and reload your stuff.

Question How do I use the exporting option to get rid of unwanted versions of my library instead of using cloning?

Answer Get a new copy of the ABTMGR30.DAT file from the CD and start with that. In addition to an online copy of the original manager, we keep a base manager that has all of the original application versions plus any PTFs and third-party applications. Since we keep this base manager up to date, it becomes very easy for us to purge by exporting just our applications to a copy of the base.

Prompting for Ownership

Question How do I get an image to prompt the user for ″Who does this image belong to?″

Answer Try this:

Smalltalk

```
EmUser resetImageOwner  ″sets the image owner to nil″
```

Then save the image and exit VisualAge for Smalltalk. When VisualAge for Smalltalk starts, you get the ″who does this image belong to″ prompt.

Moving Development Code

Question How do I move development code between libraries?

Answer The best way is by creating a new library, then exporting a configuration map or applications to the library. Exporting requires the parts to be versioned because editions cannot be exported.

Distributing an Application for OS/2 and AIX

Question How do I distribute an application for both OS/2 and AIX clients?

Answer You must package on the target environment. We access a single manager on an AIX machine for all environments using TCP/IP and the EMSRV support; thus no import/export is necessary, just a load and package.

Exporting an Application to a Non-LAN PC

Question How do I export an application to another PC not attached to a LAN?

Answer You can create an export file that contains the application that you want to copy to the other VisualAge for Smalltalk workstation.

Keep in mind the following:

- The component being exported must be a version.
- The component version you want to export cannot already exist in the target library. If it does already exist in that library, the system exports only those components not already there.

Here′s what you do. From your source image:

- Open an Application Manager browser from the transcript (**Smalltalk Tools** → **Manage Applications**).
- From the list of applications, select the application version you want to export.
- From the **Application** menu option, select **System** → **Export To.**
- In the displayed prompter, specify the path and file name for the target library.
- Click on **OK**. The system moves a copy of the application version from the library you are currently using to the target library.

If you cannot fit this file on one diskette, try using saveram or some other tool that can split and pack files.

From the image on the VisualAge for Smalltalk workstation where you wish to import the application:

- Open the Application Manager Browser
- From the **Application** menu option, select **Select** → **Import** → **Applications**.
- In the displayed prompter, specify a path and file name for the library storing the component.
- Click on **OK**. A window is displayed, prompting you for the name and version of the application you want to import.
- Select an application; then select a version of the application.
- Click on **OK**. A copy of the application version is moved from the selected library into the library you are currently using.

Associating .DAT files with Features

Question How can I figure out which feature was installed or loaded from which .DAT file. Does there exist a cross-reference list so I can identify the .DAT file with a given feature?

Answer To figure out which .DAT is which, you can look at the .CTL files, which are found in the feature subdirectory. The first word in the file tells you the name of the feature as you see it in the Load Features list. The rest of that first line is just information that tells VisualAge for Smalltalk under what conditions to load the feature. The important information is the rest of the file. It lists the configuration map versions that are loaded for the feature and the .DAT file they are loaded from. You can use this list to determine whether you have used a .DAT file or not.

Manager Consistency

Question How can I check my library manager for consistency so that I don′t end up with a badly corrupted repository where data is lost?

Answer You can use the EmLibraryStatistics to check your manager for consistency. To use EmLibraryStatistics, read the comments in the class method *checkConsistencyOf:*.

Exporting a Pool Dictionary

Question Suppose I define a new pool dictionary for my application. How can I make sure that this pool dictionary is loaded when I export the application to another image?

Answer There are a few methods that you will need to implement to en sure that your pool dictionaries are defined in the image prior to loading your application. They are *toBeLoadedCode:, wasRemovedCode:* and *installToBeLoadedCode.* Details can be found in the *IBM Smalltalk User′s Guide*.

Exporting Configuration Maps

Question I′d like to export a whole bunch of configuration maps (specific versions of specific configuration maps) to an empty .DAT library. I′d like to do this without the user interface so that I can run this unattended overnight. Do you have a script that can do this?

Answer You can do this without a script. Select export from the pop-up menu in the configuration maps pane (not the versions, but names) of the configuration maps browser. Then select the target library, and you′ll be prompted for *all* map or version pairs up front.

Manager Growth Rate

Question What is the approximate rate of the Manager file growth on the server? I heard about 1 MB per day per developer. Is it true?

Answer It comes down to how much code your developers can crank out or change per day. If they migrate some existing application into a brand new VisualAge for Smalltalk library, chances are the library will grow with however much code is being migrated into the manager.

If your developers write Smalltalk code from scratch (keyboard input) chances are that they won′t be able to type anywhere near 1 million key strokes of desirable Smalltalk code per day. One can try for half that on a DBCS input. Modifying an existing method may store incremental changes only (I don′t deal with that algorithm.) If you make changes via the VisualAge for Smalltalk Composition Editor, the run-time code is stored as various methods, so there won′t be much difference in the way method source code is stored. This may allow for method changes at a faster rate than manual input.

A 1 GB drive should give you very comfortable room for development. Even though you may not need that much, a 1 GB drive should be cheap enough today to buy that luxury, where a 300 MB or 500 MB drive would risk the later discovery that you′ll have to spend time to move the manager to a larger drive.

Anyway, make sure that your team has a good reliable means to back up the shared manager file. You don′t really need to back up each developer image file, as the files can be rebuilt from the manager or library file. And make as frequent (daily, weekly) a backup schedule as you can afford.

Class Changes versus Alternatives

Question What is the difference between class changes and class alternatives?

Answer I found the following information in an ENVY/Manager user manual. I hope the information is helpful.

- *File Out Changes*

  File out methods in the primary class editions that are different from the corresponding methods in the alternative class editions.

- *File Out Alternatives*

  File out methods in the alternative class editions that are different from the corresponding methods in the primary class editions.

“Primary” seems to indicate the editions that appear in the left pane of a changes browser. “Alternative” method editions appear in the right pane of the browser.

Moving Windows Library to AIX

Question We are developing in Windows now but are considering also doing some AIX development. My understanding is that this would require us to move our manager to AIX. Is the procedure for doing this documented somewhere? I took a quick look through the *VisualAge for Smalltalk Users Guide* where other library administration tasks are documented, but didn′t see anything. I assume that there′s more to it than just copying the manager file, but what?

Answer

You should reinstall the server code on AIX following the instructions in the READMEs, and so on, and then copy your library file from Windows over the library that was added during the server install on AIX. Also, all of your clients will need to update their abt.cnf to point to the new library location and to use EMSRV as the transport mechanism.

Identical Users

Question

In our team development environment, we have two distinct users who have the same name. I suppose this happened from importing classes from another library, or maybe there is a space after the name. In any case, can anyone make a recommendation as to how best to get rid of one of these users? Right now, they are both owners of different classes and managers of applications.

Answer

Assign new owners (or managers) to the classes and applications created under the user that you don′t want. To do this, use the **Become New Owner** and **Become New Manager** options of a **Group Members** menu.

Next you might want to check for any classes still owned by the user (Select **Queries** → **Classes Owned By** from the **Smalltalk Tools** menu). Finally, purge the user.

You should also understand how you got into this situation. There are two inportant parts to a user:

1. The external name. This is what you see in the browsers and this does not have to be unique.
2. The unique name. This is used by the system to differentiate users.

In your case, the users have unique underlying names, but their visible names happen to be the same. If you wish, you could simply change the visible name as described above.

The lesson to learn is that you should not define new users without making sure that the unique names are consistent. The easiest way to do this is to export and import the users explicitly or to allow the definitions to come over as a byproduct of code import or export.

Classes Available in the Library

Question Is there any way to develop a snooper that can be used to compare the loaded classes against what is available in the library? A function like that would provide the user with the capability of selectively picking up fixes on a periodic basis into an existing image. Is it feasible to develop such a utility that could be run on demand?

Answer The Team version/release control features give you the functionality that you are looking for. The way you can work with it is to have configuration maps for the various major components of the product, and applications within those configuration maps. Whenever you open an application to make a change, you immediately release the application′s edition to the configuration map (opening the configuration map as well if necessary).

Now, to get the latest and greatest, check out your configuration maps to see if any new application editions have been released, and if so, you can load them if you are interested. Once you have open application editions, you can see any newly released classes by examining the application in the Application Manager. If the released class does not match the one currently loaded in your image, it will have a > symbol next to it.

To further assist you, there is a **Query** menu option under **Smalltalk Tools** on the Transcript window. The choice for unreleased classes will print a list (to the Transcript) of every class in your image (by application) that is not the released version for that application. You can also use this in addition to Open Class Editions to help you locate all the changes you have made leading up to a build so you know what you need to version and release.

ENVY/Windows95/Novell

Question What about accessing ENVY/Manager library using Windows 95 clients and Netware servers?

Answer IBM has recently certified ENVY/Manager library access using Windows 95 clients and Netware 3.12 and 4.1 servers.

Windows 95 clients must use only the Novell Client32 for Windows 95 requestor. IBM has tested library access using the Microsoft Client for Netware Networks and has found problems that may result in corrupt libraries if this requestor is used. The problem has been logged with Microsoft Australia Technical Support as Problem 1570370.

The Client32 for Windows 95 requestor must be used only in conjunction with two patches available from Novell:

- c3295c.exe (Client32 for Windows 95 Update C)
- landr5.exe (server patch)

If these two patches are not used, users may experience severe performance problems when reading from attached servers and may also find that locks are not correctly released if more than one image is running concurrently on the same machine. The two patches will correct both of these problems.

The Client32 for Windows 95 requestor has many parameters that may be changed in the Advanced Settings dialog. IBM has tested with default settings provided on installation which in most cases trade data integrity for performance (for example, True Commit set to off). Although other settings have not been tested, changing any of the settings in the "Performance, Cache" parameter group to less optimistic values should not present any problems.

ENVY/Manager′s release lock mode parameter should be set to *true* when using this configuration.

Chapter 6. Microsoft Windows

In this chapter, we cover VisualAge for Smalltalk on the Microsoft Windows platform.

Sizing the Windows 95 Toolbar

Tip

The Windows 95 toolbar sizes improperly at run time, potentially covering other visual parts. To work around this problem, adjust the *framingSpec* of the tool bar so that its bottom edge is attached to another widget. This will prevent the toolbar from covering other widgets. A fix for this problem is available in the downloadable update.

Windows 95 TreeView

Tip

The Windows 95 TreeView traps the collapse and expand events but does not properly signal the corresponding VisualAge events. Therefore, connecting to the *itemExpanded* or *itemCollapsed* event of the TreeView does not work. The AbtCwTreeView methods below correct the problem.

Smalltalk

```
collapseNode: aWidget clientData: clientData callData: callData
  ″Private - The node collapse callback was called.″

  self signalEvent: #itemCollapsed with:
     ( self partForNode: callData value )
```

Smalltalk

```
expandNode: aWidget clientData: clientData callData: callData
  ″Private - The node expand callback was called.″

  self signalEvent: #itemExpanded with:
     ( self partForNode: callData value )
```

As an alternative, you can register Smalltalk callbacks in your parts to handle the *itemExpanded* and *itemCollapsed* events. Use code like that shown below:

Smalltalk

```
widget
  addCallback: #expandCallback
  receiver: self
  selector: #expandNode:clientData:callData:
  clientData: nil.

widget
  addCallback: #collapseCallback
  receiver: self
  selector: #collapseNode:clientData:callData:
  clientData: nil.
```

You would then need to implement the *expandNode:clientData:callData:* and *collapseNode:clientData:callData:* selectors in your parts to handle the registered events. See the *IBM Smalltalk Programmer's Reference* for more information about callbacks.

Deactivation of OLE Embedded WordPro Document

Tip

An unhandled exception occurs when deactivating an embedded, in-place activated OLE client container that contains a Lotus WordPro document. To work around this problem, when using WordPro documents, specify an *activationPolicy* of XmOPEN in the OLE Client part to disable in-place activation.

Copying from Windows Explorer into an OLE Client Part

Tip

Use copy and paste to share OLE objects between the Windows Explorer and an OLE Client part. Dropping an OLE object that was dragged from the Windows Explorer onto an OLE Client part does not work.

Generating OLE Methods

Tip

The OLE Class Generator generates set methods that signal the wrong event name. For example, the generated name of the change symbol for the attribute *count* is #countChanged. However, the generated *count:* setter method contains the following code:

Smalltalk

```
self signalEvent: #count with: aValue
```

The *signalEvent* code will have no effect because the change symbol for the does not match the signaled event name. To fix this, change the OlexPropertyDescription>>#abtSetSelectorWithInterace: method. Change the code lines:

Smalltalk

```
nextPutAll: ′self signalEvent: ′ ;
nextPutAll: selectorName asSymbol printString ;
```

to:

Smalltalk

```
nextPutAll: ′self signalEvent: ′ ;
nextPutAll: ( selectorName , ′Changed′ ) asSymbol printString;
```

Free Disk Space on Windows Platform

Question Does anybody know how to retrieve free disk space information on Windows platforms?

Answer The GetDiskFreeSpace function call returns the following information:

1. Number of sectors per cluster
2. Number of bytes per sector
3. Number of free clusters on the disk
4. Total number of clsuters on the disk

Pointers to variables where the above information can be returned must be passed to the GetDiskFreeSpace function. To get the total amount of free disk space, you must multiply the sectors per cluster by the bytes per sector by the number of free clusters.

Listed below is code that will call the GetDiskFreeSpace function, calculate the amount of free space on C:, and display that value in a message prompter.

Smalltalk

```
| oscall freeSpace sectorsPerCluster bytesPerSector
  freeClusters totalClusters rc |

oscall := OSCall new.
sectorsPerCluster := ByteArray new: 4.
bytesPerSector := ByteArray new: 4.
freeClusters := ByteArray new: 4.
totalClusters := ByteArray new: 4.

rc := oscall getDiskFreeSpace: ′C:′
  lpSectorsPerCluster: sectorsPerCluster
  lpBytesPerSector: bytesPerSector
  lpFreeClusters: freeClusters
  lpClusters: totalClusters.

freeSpace := (sectorsPerCluster abtAsInteger) *
  (bytesPerSector abtAsInteger) *
  (freeClusters abtAsInteger).

CwMessagePrompter message: ′Free space on disk C: is ′,
  freeSpace printString.
```

Chapter 7. National Language Support

In this chapter, we cover questions related to national language support of VisualAge for Smalltalk.

Code Page Support

Question Does VisualAge for Smalltalk support my particular code page?

Answer VisualAge for Smalltalk makes an operating system call to translate the string. If the operating system call doesn′t support the conversion between the code pages, then it will try to translate with the translate tables you provide. The operating system calls are different for each platform. To see if the call is supported on your machine, take any string and pass the message *convertToCodePage:*

Smalltalk

```
′Hello World′ convertToCodePage: 37
```

where the number passed is the code page you wish to convert to.

If the code page is not supported, the method *convertToCodePage:* will return an AbtError with errorText set to ′Conversion from codepage <...> to codepage <...> is not supported′.

Code Page and MPR Problems

Tip The code page is stored in the MPR file so it can be correctly translated to the new code page. For example, on my machine on Windows, character 252 = ü (U umlaut). On OS/2, that same character is 129. However, I don′t have to modify the MPR file I generated on Windows because code points are automatically translated for text stored in MPR files. The ABTRULES.NLS file contains the mappings, and AbtCharacterTranslator handles the translations. All you have to do is this:

1. Generate your external strings with: *YourApp abtGenerateExternalStringsIfNecessary*.
2. Add *abtExternalizedStringBuildingInfo* method to your application.
3. Add *loaded* method that calls *self abtRegisterExternalStrings.*

4. Add *unloaded* method that calls *self abtUnregisterExternalStrings.*

Required CAT Files

Tip The following script helps you to determine, based on your packager output log, which *.CAT files you need with your run-time image. Just save the output log to a file and then run this script:

Smalltalk

```
| stream tokens file |
file := (CwFileSelectionPrompter new)
  title: ′Packaging Messages′; prompt.

(file isNil) ifTrue: [^nil].
stream := CfsReadFileStream open: file.
[stream atEnd] whileFalse: [
  tokens := stream nextLine subStrings.
  ((tokens size > 0) and: [((tokens at: 1)
    indexOfSubCollection: ′NlsCat′ startingAt: 1) > 0]) ifTrue: [
    Transcript cr; show: ′References messages in: ′,
      ((Smalltalk at: (tokens at: 1) asGlobalKey) at: ′!!CATALOGNAME′)]].
stream close.
```

Decimal Point Representations

Question Is it possible to change the default decimal point for the current local representation which probably is a comma?

Answer The following code results in 0,75 if the local representation of a decimal uses a comma:

Smalltalk

```
Locale current lcNumeric printNumber: (3/4) asFloat on: Transcript
```

The next code will result in 0.75, and the reason is that the compiler needs to be able to recognize a float from a method named ″,″.

Smalltalk

```
(3/4) asFloat printOn: Transcript
```

When you use *printString* on a Float, you are getting a representation that can be highlighted and evaluated (compiled). For occasions where you need locale-specific output, you instead use the *printNumber:on:* method.

Using Multiple National Languages

Question How do I use multiple national languages with VisualAge for Smalltalk?

Answer See the *VisualAge for Smalltalk User′s Guide*, for a complete discussion of how national language support (NLS) enablement works.

Adding French Characters to English

Question How do I allow input of French characters in my English application?

Answer I′ve been running Windows 95 lately, and for typing French characters I use the ″U.S. International″ keyboard layout. That maps certain deadkeys, such as ′ (apostrophe) followed by e gives è (e accent grave). I′m not certain OS/2 has this same keyboard layout scheme, if not you could implement it yourself fairly easily using the text field modify verify callback. ′ (single quote) followed by a character (e) adds accent grave, back quote followed by a character (a, e, u) adds accent aigu, Shift+6 (caret) followed by a character (a, e, o, u) adds accent circonflexe,″ double-quote followed by a character (e, i) adds accent umlaut, Accent character followed by space adds character (such as ′ + space = ′), followed by c adds c cedilla. The only drawback I′ve found to this scheme is that if you need to type lots of quoted text, it is irritating because you have to remember to type an extra space. Of course, the other option is to switch your keyboard to the French layout. For U.S. users it will change most notably the location of the A, Q, M, period, comma, numbers, and so on. The other scheme I′ve seen, called *Transparent Language*, is to map the up/down arrow to cycle through accented characters. This is a good approach if characters are rarely entered (Macintosh uses a similar scheme, but uses the control key).

Problems with National Language Support in Version 3

Question I have a problem with national language support in Version 3. Why?

Answer This could be a problem with the nlspath search. (I believe that the dynamic example does not rely on the search; it expects you to tell it the exact file.) Are there any old (nontranslated) MPRs earlier in the NLS search path. The search starts at the working directory and then works its way down the entries in nlspath. You can see which MPR each nlsgroup is bound to by executing this code fragment:

Smalltalk

```
AbtMRIManager requestsAndResolutions do: [ :aRequest |

  Transcript cr; nextPutAll: aRequest request printString.
  aRequest isResolved
  ifTrue: [
     Transcript tab;tab;  nextPutAll: aRequest
     resolution fileSpec fullName ]
  ifFalse: [ Transcript tab;tab; nextPutAll: ′not resolved.′ ] ].
```

Code Page Conversion between OS/2 and Windows

Question How can I develop an application with VisualAge for Smalltalk for OS/2, and then reuse it from VisualAge for Smalltalk for Windows, where the code page in OS/2 is 850 and in Windows it′s 437?

Answer Windows uses code page 819 (ISO 8859-1) regardless of what the code page is set to in DOS. The code page for DOS my be 437, but Windows ignores it. Code page 819 doesn′t exist in OS/2.

Although OS/2 doesn′t support code page 819, code pages 819 and 850 have consistent mappings for points 32-126. After that, it is hit or miss as to whether or not they match.

In Version 3 of VisualAge for Smalltalk, we automatically map the characters in label strings, messages, and the like to the correct code points for you. In Version 2.0, however, you have to do the mappings yourself. The easiest way to do this is to select the separate application strings option for the application you want to run in both OS/2 and Windows. Next, export the strings to a file. Edit this file (under Windows) and change all the characters that aren′t mapped correctly. After editing the file, import the strings (again under Windows). When you run the application under Windows, the new strings will be displayed.

The problem does not exist if you are using DB2/2 version 2.1. They take care of the code page translation for you.

Double-Byte Character Set

Tip

When installing on a Windows DBCS machine, please use the default specified on the Select Folder dialog. Because the English version of InstallShield does not support DBCS strings, translated strings appear incorrectly in the list of existing folder names.

Tip

In the Simplified Chinese environment on AIX, several of the dialogs may have strings truncated (vertically and horizontally) because of the system font used by VisualAge for Smalltalk. By installing a different set of font aliases for Simplified Chinese or changing the named fonts for several VisualAge for Smalltalk dialog controls (button labels, headings and subheadings, icon text, and text), the truncated strings will appear correctly. To change the VisualAge named fonts (used for settings pages and icon text), please refer to the ″VISUALAGE NAMED FONTS″ section of the Preferences Workspace. The Preferences Workspace can be displayed by selecting **Options** → **Preferences Workspace** from the VisualAge for Smalltalk Organizer dialog.

Appendix A. Special Notices

This publication is intended to help VisualAge for Smalltalk developers avoid common programming pitfalls and as a guide for frequently asked questions about the IBM Smalltalk language, VisualAge for Smalltalk, and its features. The information in this publication is not intended as the specification of any programming interfaces that are provided by VisualAge for Smalltalk. See the PUBLICATIONS section of the IBM Programming Announcement for VisualAge for Smalltalk for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (″vendor″) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX
AIX/6000
Application System/400
APPN
AS/400
CICS
Common User Access
CUA
DATABASE 2
DB2
DB2/2
DB2/400
DB2/6000
IBM
IMS
MQ
MQSeries
MVS
MVS/ESA
Object Connection
OS/2
OS/400
Presentation Manager
PS/2
RPG/400
SOM
SQL/400
TalkLink
VisualAge
VisualGen
VisualInfo

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.
PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.
UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.
Microsoft, Windows, and the Windows 95 logo
are trademarks or registered trademarks of Microsoft Corporation.

HP, HP-UX	Hewlett-Packard Company
Sun, Solaris	Sun Microsystems, Inc.
Lotus, Notes	Lotus Development Corporation

Other trademarks are trademarks of their respective companies.

Appendix B. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

International Technical Support Organization Publications

For information on ordering these ITSO publications see "How to Get ITSO Redbooks" on page 275.

- *VisualAge: Concepts and Features*, GG24-3946
- *VisualAge and Transaction Processing in a Client/Server Environment*, GG24-4487
- *AS/400 Application Development with VisualAge for Smalltalk*, SG24-2535
- *World Wide Web Server Development with VisualAge for Smalltalk*, SG24-4734
- *VisualAge: Building GUIs for Existing Applications*, GG24-4244
- *VisualAge for Smalltalk Distributed*, SG24-4521
- *VisualAge for Smalltalk and SOMobjects*, SG24-4390
- *OO Programming with Client Access for OS/400 and ODBC using VisualAge for Smalltalk*, SG24-4718
- *Application Development with VisualAge for Smalltalk and MQSeries*, SG24-2117
- *Object-Oriented Application Development with VisualAge for C++ for OS/2*, SG24-2593
- *Programming with VisualAge for C++ for Windows*, SG24-4782
- *IBM VisualAge for Cobol for OS/2: Workframe User Guide*, SG24-4604
- *IBM VisualAge for Cobol for OS/2: Primer*, SG24-4605
- *IBM VisualAge for Cobol for OS/2: Object-Oriented Programming*, SG24-4606
- *Visual Modeling Technique, Object Technology using Visual Programming*, SG24-4227

Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

Other Publications

These publications are also relevant as further information sources:

- *VisualAge for Smalltalk User′s Guide*, SC34-4518
- *VisualAge for Smalltalk Programmer′s Guide to Building Parts for Fun and Profit*, SC34-4496
- *IBM Smalltalk Programmer′s Reference*, SC34-4493
- *IBM Smalltalk User′s Guide*, SC34-4536
- *IBM Smalltalk: The Language*, by David N. Smith, Benjamin/Cummings Publishing Company, ISBN: 0-8053-0908
- *VisualAge and Transaction Processing in a Client/Server Environment*, by Andreas Bitterer, Michel Brassard, William Nadal, and Chris Wong, Prentice Hall PTR, 1996, ISBN: 0-13-460874-7
- *AS/400 Application Development with VisualAge for Smalltalk*, by Andreas Bitterer, Masahiko Hamada, John Oosthuizen, Gino Porciello, and Håkon Rambek, Prentice Hall PTR, 1997, ISBN: 0-13-520453-4
- *World Wide Web Programming: VisualAge for C++ and Smalltalk*, by Andreas Bitterer and Marc Carrel-Billiard, Prentice Hall PTR, 1997, ISBN: 0-13-612466-6
- *Object-Oriented Application Development with VisualAge for C++ for OS/2*, by Marc Carrel-Billiard, Peter Jakab, Isabelle Mauny, and Rainer Vetter. Prentice Hall PTR, 1996, ISBN: 0-13-242447-9
- *Programming with VisualAge for C++ for Windows*, by Marc Carrel-Billiard, Michael Friess, and Isabelle Mauny, Prentice Hall PTR, 1997, ISBN: 0-13-618208-9

- *VisualAge for Smalltalk Distributed*, by Walter Fang, Sven Guyet, Randy Haven, Matti Vilmi, and Eduardo Eckmann, Prentice Hall PTR, 1996, ISBN: 0-13-570805-2
- *VisualAge for Smalltalk SOMsupport*, by Walter Fang, Raymond Chu, and Markus Weyerhäuser, Prentice Hall PTR, 1997, ISBN: 0-13-570813-3
- *IBM Smalltalk Programming for Windows and OS/2*, by Dan Shafer and Scott Herndon, Prima Publishing, 1995, ISBN: 1-55958-749-0
- *Visual Modeling Technique*, by Daniel Tkach, Walter Fang, and Andrew So, Addison-Wesley, 1996, ISBN: 0-8053-2574-3
- *Smalltalk with Style*, by Edward Klimas, Suzanne Skublics, and David Thomas, Prentice Hall PTR, 1996, ISBN: 0-13-165549-3
- *Object Technology in Application Development*, by Daniel Tkach and Richard Puttick, Benjamin/Cummings Publishing Company, ISBN: 0-8053-2572-7
- *Designing Object-Oriented Software*, by Rebecca Wirfs-Brock, Brian Wilkerson, and Lauren Wiener, Prentice Hall PTR, 1994, ISBN: 0-13-629825-7
- *TCP/IP Tutorial and Technical Overview*, by Eamon Murphy, Steve Hayes, and Matthias Enders, Prentice Hall PTR, 1995, ISBN: 0-13-460858-5
- *Object-Oriented Interface Design: IBM Common User Access Guidelines*, SC34-4399

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL http://www.redbooks.ibm.com.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO: type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Home Page on the World Wide Web**

  http://w3.itso.ibm.com/redbooks

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications
Publications Customer Support
P.O. Box 29570
Raleigh, NC 27626-0570
USA

IBM Publications
144-4th Avenue, S.W.
Calgary, Alberta T2P 3N5
Canada

IBM Direct Services
Sortemosevej 21
DK-3450 Allerød
Denmark

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1)001-408-256-5422 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to `softwareshop@vnet.ibm.com`

- **On the World Wide Web**

Redbooks Home Page	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to `announce@webster.ibmlink.ibm.com` with the keyword `subscribe` in the body of the note (leave the subject line blank).

IBM Redbook Order Form

Please send me the following:

Title	**Order Number**	**Quantity**

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number
- Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Glossary

A

abstract class. A class that provides common behavior across a set of subclasses but is not itself designed to have instances that work.

abstraction. A simplified description or view of something that emphasizes characteristics or purposes relevant to the user while suppressing details that are immaterial or distracting.

accessor methods. Methods that an object provides to define the interface to its instance variables. The accessor method to return the value of an instance variable is often called a *get* method or *getter* method, and the accessor method to assign a value to an instance variable is called a *set* method or *setter* method.

access plan. The control structure produced during compile time that is used to process SQL statements encountered when the program is run.

action. In VisualAge, a function or operation that a part can perform upon receiving a message. Actions enable a part′s public interface to give other parts access to its behaviors. Compare to *event* and *attribute*.

activate. To make a resource of a node ready to perform the functions for which it was designed. Contrast with *deactivate*.

activator. In distributed processing environments, a program running in the background (sometimes called a ″daemon process″) that initially activates other programs the first time they are called.

active. (1) Able to communicate on the network. A token-ring network adapter is active if it is able to transmit and receive on the network. (2) Operational. (3) Pertaining to a node or device that is connected or is available for connection to another node or device. (4) Currently transmitting or receiving.

adapter. Hardware card that allows a device, such as a PC, to communicate with another device, such as a monitor, a printer, or other I/O device.

address. (1) In data communication, the IEEE-assigned unique code or the unique locally administered code assigned to each device or workstation connected to a network. (2) A character, group of characters, or a value that identifies a register, a particular part of storage, a data source, or a data sink. The value is represented by one or more characters. (3) To refer to a device or an item of data by its address. (4) The location in the storage of a computer where data is stored. (5) In word processing, the location, identified by the address code, of a specific section of the recording medium or storage.

Advanced Program-to-Program Communication (APPC). (1) IBM′s architected solution for program-to-program communication, distributed transaction processing, and remote database access. A transaction program (TP) using the APPC API can communicate with other TPs on systems that support APPC. (2) An implementation of the Systems Network Architecture (SNA) logical unit (LU) 6.2 protocol that enables interconnected systems to communicate and share the processing of programs.

agent. A VisualAge part used to encapsulate business logic and data access executed outside a VisualAge for Smalltalk image.

allocate. A logical unit (LU) 6.2 application program interface (API) verb used to assign a session to a conversation for the conversation′s use. Contrast with *deallocate*.

American National Standard Code for Information Interchange (ASCII). The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphics characters.

analog. Pertaining to data consisting of continuously variable physical quantities. Contrast with *digital*.

anonymous FTP. Using the FTP function of the Internet anonymously by not logging in with an actual, secret login ID and password. Often permitted by large, host computers that are willing to share openly some of the files on their system to outside users who otherwise would not be able to log in. See also FTP.

API. See *application program interface*.

APPC. See *Advanced Program-to-Program Communication*.

application. (1) The use to which an information processing system is put; for example, a payroll application or an order-entry application. (2) A collection of defined and extended classes that provides a reusable piece of functionality. An application contains and organizes functionally related classes. It also can contain subapplications and specify prerequisites.

application layer. In Open Systems Architecture, the layer of the OSI reference model that provides a means for application processes residing in different open systems to exchange information.

application manager. (1) A team member who is responsible for the overall state of an application. An application manager coordinates the activities of the application′s developers and assigns ownership of classes to team members. (2) The browser from which users can create, delete, manage, or configure applications in their image.

application program. (1) A program written for or by a user that applies to the user's work. Some application programs receive support and services from a special kind of application program called a network application program. (2) A program used to connect and communicate with stations in a network, enabling users to perform application-oriented activities.

Note: Do not use the term *application* in place of *application program.*

application program interface (API). An architected functional interface supplied by an operating system or other software system. The interface enables an application program written in a high-level language to use specific data or functions of the underlying system.

application programmer. A person who primarily writes and modifies programs for application data. Contrast with *system programmer*.

Archie. An Internet tool for finding files stored on *anonymous FTP* sites. You need to know the exact file name or a substring of it.

architecture. A logical structure that encompasses operating principles including services, functions, and protocols.

argument. A data element included as part of a message. Arguments provide additional information that the receiver can use to perform the requested operation. Binary messages and keyword messages take arguments. In a keyword message, a colon (:) following a keyword indicates that an argument is required.

ARPANet. (Advanced Research Projects Agency Network), the precursor to the *Internet.* Developed in the late 1960s and early 1970s by the US Department of Defense as an experiment in wide-area-networking that would survive a nuclear war. See also Internet.

array literal. A literal that is an indexed sequence of literals. The symbol # precedes this sequence and parentheses enclose the sequence. For example, #(5 7 9) is an array of three integers.

ASCII. (American Standard Code for Information Interchange), this is the world-wide standard for the code numbers used by computers to represent all the upper and lower-case Latin letters, numbers, punctuation, etc. There are 128 standard ASCII codes each of which can be represented by a 7-digit binary number, 0000000 through 1111111.

atomic. (1) Pertaining to the smallest element in a composite object that can be manipulated independently. (2) Pertaining to a set of operations performed such that either all the operations performed or none of the operations are performed. (3) Pertaining to a set of operations that cannot be interrupted, such as a critical section of a parallel program.

attribute. (1) In VisualAge, data that represents a property of a part. (For example, a customer part could have a name attribute and an address attribute.) Attributes enable a part′s public interface to give other parts access to its properties. An attribute can itself be a part, with its own behavior and attributes. Compare to *event* and *action*. (2) Information that describes the characteristics of system objects or program objects.

attribute-to-attribute connection. A connection from an attribute of one part to an attribute of another part. When one attribute is updated, the other attribute is updated automatically. See also *connection*.

attribute-to-script connection. A connection from an attribute of a part to a script. The connected attribute receives its value from the script, which can make calculations based on the values of other parts. See also *connection*.

authority. The right to do something on the system or to use an object, such as a file or document, in the system.

authorization list. A list that gives a group of users one or more types of access to objects (such as files or programs) or data in the objects (such as records in a file). It consists of a list of two or more user IDs and their authorities for system resources.

B

backbone. A high-speed line or series of connections that forms a major pathway within a network. The term is relative, as a backbone in a small network is likely to be much smaller than many nonbackbone lines in a large network. See also Network.

bandwidth. The transmission capacity of the lines that carry the Internet′s electronic traffic. Historically, it′s imposed severe limitations on the ability of the Internet to deliver all that we are demanding it deliver, but fiber-optic cables will ensure that bandwidth soon will be essentially limitless and free.

Basic Input/Output System (BIOS). In IBM personal computers with PC I/O channel architecture, microcode that controls basic hardware operations such as interactions with diskette drives, fixed disk drives, and the keyboard.

baud. In common usage the baud rate of a modem is how many bits it can send or receive per second. Technically, baud is the number of times per second that the carrier signal shifts value; for example a 1200 bit/second modem actually runs at 300 baud, but it moves 4 bits per baud. See also bit, modem.

behavior. (1) The set of external characteristics that an object exhibits. (2) The abstract class that provides common behavior for class and metaclass objects.

binary. (1) Pertaining to a system of numbers to the base two; the binary digits are 0 and 1. (2) Pertaining to a selection, choice, or condition that has two possible different values or states.

bind. The process by which the output from the SQL precompiler is converted to a usable structure called an *access plan*. This process is the one during which access paths to the data are selected and some authorization checking is performed.

bit. (binary digit) A single digit number in base 2, in other words, either a 1 or a zero. The smallest unit of computerized data. *Bandwidth* is usually measured in bits per second. See also bandwidth, bps, byte, kilobyte, megabyte.

block. A Smalltalk object consisting of one or more statements, enclosed in brackets [ ], passed as arguments or used as the receiver of messages that implement control flow. Blocks can define temporary variables for their own use.

Bps. (bits per second) A measurement of how fast data is moved from one place to another. A 28.8 modem can move 28,800 bits per second. See also bandwidth, bit.

browse. A way of looking at a file that does not allow you to change it.

browser. (1) A window that supports one or more programming activities, such as creating new classes or methods, modifying existing classes or methods, or viewing library members. (2) Software that enables users to browse through the cyberspace of the World Wide Web. See also Client, URL, WWW.

buffer. (1) A portion of storage used to hold input or output data temporarily. (2) A routine or storage used to compensate for a difference in data rate or time of occurrence of events, when transferring data from one device to another.

byte. A set of bits that represent a single character. Usually there are 8 bits in a byte, sometimes more, depending on how the measurement is being made.

C

cascaded messages. Multiple messages sent to the same receiver object. The messages are separated by a semicolon (;).

category. (1) On the VisualAge Composition Editor, a selectable grouping of parts represented by an icon in the left-most column. Selecting a category displays the parts belonging to that category in the next column over. (2) A logical association of a group of methods within a class, with a name assigned by the class developer.

CGI Link. A stand-alone executable program that receives incoming CGI requests and routes them to the VisualAge application. CGI Link runs on the HTTP server, which does not have to be the same as the machine running the VisualAge application.

CGI Link session data. A Web Connection nonvisual part that holds a persistent data object. You can use CGI Link Session Data to store an application-specific object that remains valid from one CGI query to the next, for the duration of a **session**.

change-event symbol. In VisualAge, the code used to signal that an attribute has changed in value.

character. A symbol used in printing. For example, a letter of the alphabet, a numeral, punctuation, or any other symbol that represents information.

character literal. A literal that is any single character preceded by a dollar sign ($).

CICS. See *Customer Information Control System*.

class. The specification of an object, including its attributes and behavior. Once defined, a class can be used as a template for the creation of object instances. *Class,* therefore, can also refer to the collection of objects that share those specifications. A class exists within a hierarchy of classes in which it inherits attributes and behavior from its superclasses, which exist closer to the root of the hierarchy. See also *inheritance, metaclass, polymorphism, private class* and *public class*.

class definition. The definition of a class, containing:

- Class name

- Type of class
- Immediate superclass for the class
- Instance, class, and class instance variables
- Pool dictionaries that the class uses.

class developer. A team member who develops and changes classes. The team member who created an edition of a class is that edition′s class developer. Contrast with class owner.

class extension. An extension to the functionality of a class defined by another application. The extension consists of one or more methods that define the added functionality or behavior. These methods cannot modify the existing behavior of the defined class; they can only add behavior specific to the application that contains the extended class.

class hierarchy. A tree structure that defines the relationships among classes. A class has subclasses down the hierarchy from itself and superclasses up the hierarchy from itself. The methods and variables of a class are inherited by its subclasses.

class instance variable. Private data that belongs to a class. The defining class and each subclass maintain their own copy of the data. Only the class methods of the class can directly reference the data. Changing the data in one class does not change it for the other classes in the hierarchy. Contrast with *class variable*.

class method. A method that provides behavior for a class. Class methods are usually used to define ways to create instances of the class. Contrast with *instance method*.

class owner. Team member responsible for the integrity of that class in an application edition. The class owner is responsible for releasing class versions. Contrast with class developer.

class variable. Data that is shared by the defining class and its subclasses. The instance methods and class methods of the defining class and its subclasses can directly reference this data. Changing the data in one class changes it for all of the other classes. Contrast with *class instance variable*.

client. A software program that is used to contact and obtain data from a server software program on another computer, often across a great distance. Each client program is designed to work with one or more specific kinds of server programs, and each server requires a specific kind of client. A Web browser is a specific kind of client. See also browser, server.

client object. An object that requests services from other objects.

client/server. The model of interaction in distributed data processing in which a program at one location sends a request to a program at another location and awaits a response. The requesting program is called a client, and the answering program is called a server.

code page. A font component that associates code points and character identifiers. A code page also identifies how undefined code points are handled.

collection. (1) In Smalltalk, a set of elements in which each element is an object. (2) In SQL, a set of objects created by the SQL/400 licensed program that consists of and logically classifies a set of objects, such as tables, views, and indexes.

command. (1) A request for performance of an operation or execution of a program. (2) A character string from a source external to a system that represents a request for system action.

comment. A set of characters enclosed in double quotation marks. Smalltalk ignores comments and does not execute them.

Common Gateway Interface. A standard protocol through which a Web server can execute programs running on the server machine. CGI programs are executed in response to requests from Web client browsers.

Common Object Request Broker Architecture (CORBA). An architectural standard proposed by the Object Management Group (OMG), an industry standards organization, for creating object descriptions that are portable among programming languages and execution platforms.

Common Programming Interface Communications (CPI-C). An IBM communications architecture that defines a programming interface for peer-to-peer communications that is common across different environments and platforms. Also referred to as CPI Communications or CPI-C.

Common User Access (CUA). An IBM architecture for designing graphical user interfaces that uses a set of standard components and terminology.

component. A functional grouping of classes and related files within a product. See also *system component*.

composite part. A part that contains other parts; it can also contain data and behavior of its own. For example, a user interface view is a composite part composed of subparts such as entry fields, push buttons, and text.

Composition Editor. In VisualAge, a view that is used to build a graphical user interface and to make connections among parts.

concrete class. A subclass of an abstract class that is a specialization of the abstract class. For example, the concrete class, OrderedCollection, is a subclass of the abstract class, Collection.

configuration. (1) A description of a group of components that identifies, for each component, the component edition or version that is part of the group. (2) The arrangement of a computer system or network as defined by the nature, number, and chief characteristics of its functional units. More specifically, the term may refer to a hardware configuration or a software configuration. (3) The devices and programs that make up a system, subsystem, or network.

configuration file. The collective set of definitions that describes a configuration.

configuration map. A named group of application editions. A configuration map usually represents a product or one of its major parts.

connection. (1) In VisualAge, a formal, explicit relationship between parts. Connections define the ways in which parts communicate with one another. Making connections is the basic technique used for building any VisualAge application. See also *attribute-to-script*, *attribute-to-attribute*, *event-to-script*, and *event-to-action* connection. (2) A linkage between nodes. Connections are established and released at the Network, Session, and Presentation Layers.

construction from parts. A software development technology in which applications are assembled from reusable and existing software components known as parts.

control language. The set of all commands with which users request functions from the system.

controller. A unit that controls input/output operations for one or more devices.

conversation. In SNA, a logical connection between two transaction programs using an LU 6.2 session. Conversations are delimited by brackets to gain exclusive use of a session.

CORBA. See *Common Object Request Broker Architecture*.

CPI-C. See *Common Programming Interface Communications*.

CUA. See *Common User Access*.

Customer Information Control System (CICS). An IBM licensed program that enables transactions entered at remote terminals to be processed by user-written applications. It includes facilities for building, using, and maintaining databases.

Cyberspace. Term originated by author William Gibson in his novel *Neuromancer*, the word Cyberspace is currently used to describe the whole range of information resources available through computer networks.

D

data area. A system object used to communicate data such as common language variable values between the programs within a job and between jobs. A data area is identified to the system as a specific object type. The system-recognized identifier for the object type is *DTAARA.

database file. A system object of the type *FILE that contains descriptions of how input data is to be presented to a program from internal storage and how output data is to be presented to internal storage from a program. The collection of all database files makes up the database (all the data files stored in the system).

database manager. A VisualAge or IBM Smalltalk database component that models a database management system in order to provide the interface between an application and the database management system.

data description specifications (DDS). A format for describing the user′s database files or device files to the system. Describing a file in DDS is similar to filling in information on a form that is arranged in columns and rows. The most common characteristics, such as the names and lengths of fields, are described by putting entries in specific columns on the form. Another part of the form allows special parameters that describe less common and more varied characteristics. The finished specifications are then used as the source for creating the file.

data integrity. (1) The condition that exists as long as accidental or intentional destruction, alteration, or loss of data does not occur. (2) Preservation of data for its intended use.

data processing. The systematic performance of operations upon data; for example, handling, merging, sorting, and computing.

data queue. A way to communicate and put data used by several programs in a job or between jobs. The data queue is identified to the system as a specific type of object. The system-recognized identifier for the object type is *DTAQ.

data structure. The syntactic structure of symbolic expressions and their storage allocation characteristics.

data transfer. (1) The result of the transmission of data signals from any data source to a data receiver. (2) The movement, or copying, of data from one location and the storage of the data at another location.

deactivate. To take a resource of a node out of service, rendering it inoperable, or to place it in a state in which it cannot perform the functions for which it was designed. Contrast with *activate*.

deallocate. A logical unit (LU) 6.2 application program interface (API) verb that terminates a conversation, thereby freeing the session for a future conversation. Contrast with *allocate*.

debugger. A software tool used to detect, trace, and eliminate errors in computer programs or other software.

default. Pertaining to an attribute, value, or option that is assumed when none is explicitly specified.

delimiter. (1) A character used to indicate the beginning or end of a character string. (2) A bit pattern that defines the beginning or end of a frame or token on a LAN.

dependent LU. Any logical unit (LU) that receives an ACTLU over a link. Such LUs can act only as secondary logical units (SLUs) and can have only one LU-LU session at a time. Contrast with *independent LU*.

destination. Any point or location, such as a node, station, or particular terminal, to which information is to be sent.

device. An input/output unit such as a terminal, display, or printer.

dictionary. In Smalltalk, an unordered collection whose elements are accessed by an explicitly assigned external key. See also *pool dictionary*.

digital. (1) Pertaining to data in the form of digits. Contrast with *analog*. (2) Pertaining to data consisting of numerical values or discrete units.

disabled. (1) Pertaining to a state of a processing unit that prevents the occurrence of certain types of interruptions. (2) Pertaining to the state in which a transmission control unit or audio response unit cannot accept incoming calls on a line.

display. (1) To present information for viewing, usually on a terminal screen or a hard-copy device. (2) A device or medium on which information is presented, such as a terminal screen.

Display. (1) A Smalltalk command that executes the selected code and displays the result. (2) In IBM Smalltalk, an X/Motif concept that models the user′s hardware display. The functions of the X/Motif Display object are implemented in the IBM Smalltalk CgDisplay class.

distributed application. A workstation application that runs in cooperation with programs running on other processes or machines. Client/server applications are a subset of distributed applications.

distributed computing environment (DCE). A set of services and tools that support the creation, use, and maintenance of distributed applications in a heterogeneous computing environment.

Distributed System Object Model (DSOM). An extension to SOM enabling SOM objects to reside on multiple network nodes.

DLL. See *dynamic link library*.

domain. (1) An access method, its application programs, communication controllers, connecting lines, modems, and attached terminals. (2) In SNA, a system services control point (SSCP) and the physical units (PUs), logical units (LUs), links, link stations, and all the associated resources that the SSCP has the ability to control by means of activation requests and deactivation requests.

domain name. The unique name that identifies an Internet site. Domain names always have two or more parts, separated by dots. The part on the left is the most specific, and the part on the right is the most general. A given machine may have more than one domain name but a given domain name points to only one machine. Usually, all of the machines on a given network will have the same thing as the right-hand portion of their domain names, for example, gateway.mynetwork.com.br, mail.mynetwork.com.br, www.mynetwork.com.br, and so on. It is also possible for a domain name to exist but not be connected to an actual machine. This is often done so that a group or business can have an Internet e-mail address without having to establish a real Internet site. In these cases, some real Internet machine must handle the mail on behalf of the listed domain name. See also IP Number.

dynamic data exchange (DDE). A communication mechanism between processes that enables two applications to exchange data in a client/server relationship.

dynamic link library (DLL). A file containing data and code objects that can be used by programs or applications during loading or at run time but are not part of the program′s executable (.EXE) file.

E

EBCDIC. Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters.

edition. In the VisualAge or IBM Smalltalk team programming environment, a software component that is subject to further change. A software component can have one or more editions, identified by a time stamp stating the date and time of the edition′s creation. Many changes can be made to a single edition of a class. In contrast, every change to a method creates a new edition of that method. In its broadest sense, edition can include scratch edition and version.

EHLLAPI. Emulator high-level language application programming interface. A programming interface that enables a workstation application to communicate with a mainframe application. EHLLAPI operates in conjunction with a terminal (such as 3270) emulator.

e-mail. (Electronic mail) Messages transmitted over the Internet from user to user. E-mail can contain text, but also can carry with it files of any type as attachments.

enabled. (1) On a LAN, pertaining to an adapter or device that is active, operational, and able to receive frames from the network. (2) Pertaining to a state of a processing unit that allows the occurrence of certain types of interruptions. (3) Pertaining to the state in which a transmission control unit or an audio response unit can accept incoming calls on a line.

encapsulation. The hiding of a software object′s internal data representation. The object provides an interface that queries and manipulates the data without exposing its underlying structure.

end user. A person, device, program, or computer system that utilizes a computer network for the purpose of data processing and information exchange.

enterprise. A business or organization that consists of two or more sites separated by a public right-of-way or a geographical distance.

Ethernet. A very common method of networking computers in a *LAN*. Ethernet will handle about 10,000,000 bits/second and can be used with almost any kind of computer. See also bandwidth, LAN.

event. A representation of a change that occurs to a part. The events on a part′s public interface enable other interested parts to receive notification when something about the part changes. For example, a push button generates an event signaling that it has been clicked, which might cause another part to display a window. Compare to *attribute*.

event-to-action connection. A connection that causes an action to be performed when an event occurs. See also *connection*.

event-to-script connection. A connection that causes a script to run when an event occurs. See also *connection*.

exception. An abnormal condition such as an I/O error encountered in processing a data set or a file.

execute. To perform the actions specified by a program or a portion of a program.

execution. The process of carrying out an instruction or instructions of a computer program by a computer.

expression. In Smalltalk, the syntactic representation of one or more messages. An expression can consist of subexpressions representing the receiver and arguments of the message. The expression can also cause the assignment of its result to one or more variables.

external source. The format of Smalltalk source code that is filed out to an external file. See also *file in* and *file out*.

F

feature. (1) A major component of a software product that can be ordered separately. (2) In VisualAge, an action, attribute, or event that is available from a part′s public interface and to which other parts can connect. See also *attribute* and *event*.

field. A group of related bytes (such as name or amount) that are treated as a unit in a record.

file. (1) A generic term for the object type that refers to a database file, a device file, or a save file. The system-recognized identifier for the object type is *FILE. (2) In the hierarchical file system, a piece of related information (data), such as a document. (3) In SQL, the term is generally referred to as a table.

file in. A Smalltalk command for compiling external definitions of applications, classes, and methods from a text file.

file name. (1) A name assigned to or declared for a file. (2) The name used by a program to identify a file.

file out. A Smalltalk command for writing definitions of applications, classes, and methods to an external text file.

firewall. A combination of hardware and software that protects a local area network (LAN) from Internet hackers. It separates the network into two or more parts and restricts outsiders to the area outside the firewall. Private or sensitive information is kept inside the firewall.

first-in first-out (FIFO). A queuing technique in which the next request to be processed from a queue is the request of the highest priority that has been on the queue for the longest time.

fixed-length record. A record having the same length as all other records with which it is logically or physically associated.

form. An HTML element that can include entry fields, push buttons, and other user-interface controls through which users can enter information. Sometimes called a *fill-in form*.

format. (1) A specified arrangement of such things as characters, fields, and lines, usually used for displays, printouts, or files. (2) To arrange such things as characters, fields, and lines.

FQDN. (Fully Qualified Domain Name) The official name assigned to a computer. Organizations register names, such as *ibm.com* or *utulsa.edu*. They then assign unique names to their computers, such as *watson5.ibm.com* or *tornado.cs.utulsa.edu*.

framework. A library of classes, intended for reuse, that fall within a particular domain (for example, a communications framework or a graphics framework).

free-form surface. In VisualAge, the large open area of the Composition Editor window. The free-form surface holds the visual parts contained by the views a user builds and representations of the nonvisual parts that an application includes.

FTP. (File Transfer Protocol) The basic Internet function that enables files to be transferred between computers. You can use it to download files from a remote, host computer, as well as to upload files from your computer to a remote, host computer. (See Anonymous FTP).

function. (1) A specific purpose of an entity, or its characteristic action. (2) In data communications, a machine action such as carriage return or line feed.

G

garbage collection. A Smalltalk process for periodically identifying unreferenced objects and deallocating their memory.

gateway. A device and its associated software that interconnect networks or systems of different architectures. The connection is usually made above the reference model network layer.

GET. One of the methods used in HTTP requests. A GET request is used to retrieve data from an HTTP server. See also POST.

GIF. (Graphics Interchange Format) A graphics file format that is commonly used on the Internet to provide graphics images in Web pages.

global variable. A variable that any method in any object can access.

graphical user interface (GUI). A type of interface that enables users to communicate with a program by manipulating graphical elements rather than by entering commands. Typically, a graphical user interface includes a combination of graphics, pointing devices, menu bars, overlapping windows, and icons.

graphics. (1) The making of charts and pictures. (2) Pertaining to charts, tables, and their creation.

group member. A team member who belongs to a group that is responsible for developing an application.

GUI. See *graphical user interface*.

H

hardware. Physical equipment as opposed to programs, procedures, rules, and associated documentation.

header. The portion of a message that contains control information for the message such as one or more destination fields, name of the originating station, input sequence number, character string indicating the type of message, and priority level for the message.

HLLAPI. High-level language application programming interface. A programming interface that enables a workstation application to communicate with a mainframe application. HLLAPI usually operates in conjunction with a terminal emulator.

host. (1) A computer that ″hosts″ outside computer users by providing files, services or sharing its resources. (2) Any computer on a network that is a repository for services available to other computers on the network. It is quite common to have one host machine provide several services, such as *WWW* and *USENET*. See also Node, Network.

host computer. (1) The primary or controlling computer in a multicomputer installation or network. (2) In a network, a processing unit in where a network access method resides.

host variable. A variable in an SQL statement used for substituting data values into the statement at execution time.

HTML (hypertext markup language).. The basic language that is used to build hypertext documents on the World Wide Web. It is used in basic, plain ASCII-text documents, but when those documents are interpreted (called rendering) by a Web browser such as Netscape, the document can display formatted text, color, a variety of fonts, graphic images, special effects, hypertext jumps to other Internet locations and information forms.

HTTP (hypertext transfer protocol). The protocol for moving hypertext files across the Internet. Requires a HTTP client program on one end, and an HTTP server program on the other end. HTTP is the most important protocol used in the World Wide Web (WWW). See also Client, Server, WWW.

HTTP request. A transaction initiated by a Web browser and adhering to HTTP. The server usually responds with HTML data, but can send other kinds of objects as well.

hypertext. Text in a document that contains a hidden link to other text. You can click a mouse on a hypertext word and it will take you to the text designated in the link. Hypertext is used in Windows help programs and CD encyclopedias to jump to related references elsewhere within the same document. The wonderful thing about hypertext, however, is its ability to link– using HTTP over the Web – to any Web document in the world, yet still require only a single mouse click to jump clear around the world.

I

icon. A small pictorial representation of an object.

image. A Smalltalk file that provides a development environment on an individual workstation. An image contains object instances, classes, and methods. It must be loaded into the Smalltalk virtual machine in order to run.

implementor. For any given message selector, a method that implements that selector.

IMS. Information Management System/Virtual Storage.

inactive. (1) Not operational. (2) Pertaining to a node or device not connected or not available for connection to another node or device.

independent LU. A logical unit (LU) that does not receive an ACTLU over a link. Such LUs can act as primary logical units (PLUs) or secondary logical units (SLUs) and can have one or more LU-LU sessions at a time. Contrast with *dependent LU*.

index. A set of pointers that are logically arranged by the values of a key. Indexes provide quick access and can enforce uniqueness on the rows in a table.

inheritance. A relationship among classes in which one class shares the structure and behavior of another. A subclass inherits from a superclass.

initialize. In a LAN, to prepare the adapter (and adapter support code, if used) for use by an application program.

input/output (I/O). (1) Pertaining to a device whose parts can perform an input process and an output process at the same time.
(2) Pertaining to a functional unit or channel involved in an input process, output process, or both, concurrently or not, and to the data involved in such a process.

inspector. A Smalltalk tool for viewing the data of any Smalltalk object.

instance. An object that is a single occurrence of a particular class. An instance exists in memory or external media in persistent form. Compare to persistent object.

instance method. In Smalltalk, a method that provides behavior for particular instances of a class. Messages that invoke instance methods are sent to particular instances, rather than to the class as a whole. Contrast with *class method.*

instance variable. Private data that belongs to an instance of a class and is hidden from direct access by all other objects. Instance variables can be accessed only by the instance methods of the defining class and its subclasses.

interface. (1) A shared boundary between two functional units, defined by functional characteristics, common physical interconnection characteristics, signal characteristics, and other characteristics as appropriate. (2) A shared boundary. An interface may be a hardware component to link two devices or a portion of storage or registers accessed by two or more computer programs.
(3) Hardware, software, or both, that links systems, programs, or devices.

interface aids. In VisualAge, when used together with the RecordStructure classes, the interface aids convert existing C or COBOL definitions into Smalltalk objects.

Internet. The vast collection of interconnected networks that all use the TCP/IP protocols and that evolved from the ARPANET of the late 1960′s and early 1970′s. By July of 1995, the Internet was connecting roughly 60,000 independent networks into a vast global net.

intranet. A private *network* inside a company or organization that uses the same kinds of software that you would find on the public *Internet*, but that is only for internal use. As the Internet has become more popular, many of the tools used on the Internet are being used in private networks, for example, many companies have Web servers that are available only to employees.

IP. (Internet Protocol) The rules that provide basic Internet functions. See TCP/IP.

IP Number. An Internet address that is a unique number consisting of four parts separated by dots, sometimes called a *dotted quad*. (For example: 198.204.112.1) Every Internet computer has an IP number and most computers also have one or more domain names that are plain language substitutes for the dotted quad.

ISDN. (Integrated Services Digital Network) A set of communications standards that enable a single phone line or optical cable to carry voice, digital network services and video. ISDN is intended to eventually replace our standard telephone system.

iterative development. A software development process that allows progress in stages. At the end of each stage, the result is verified by end users. Through such verification, requirements are dynamically identified and refined while the product is under development.

J

Java. Java is a new programming language invented by Sun Microsystems that is specifically designed for writing programs that can be safely downloaded to your computer through the Internet and immediately run without fear of viruses or other harm to your computer or files. Using small Java programs (called *applets*, Web pages can include functions such as animations, calculators, and other fancy tricks. We can expect to see a huge variety of features added to the Web using Java, since you can write a Java program to do almost anything a regular computer program can do, and then include that Java program in a Web page.

job control language (JCL). A problem-oriented language designed to express statements in a job that are used to identify the job or describe its requirements to an operating system.

journal. A system object that identifies the objects being journaled, the current journal receiver, and all journal receivers on the system for the journal. The system-recognized identifier for the object type is *JRN.

journaling. The process of recording, in a journal, the changes made to a physical file member. Journaling allows the programmer to reconstruct a physical member by applying or removing the changes in the journal to a saved version of the physical file member.

journal receiver. A system object that contains journal entries added when changes are made to an object, for example, when an update is made to a file being journaled. The system-recognized identifier for the object type is *JRNRCV.

JPEG. (Joint Photographic Experts Group) The name of the committee that designed the photographic image-compression standard. JPEG is optimized for compressing full-color or gray-scale photographic-type, digital images. It doesn′t work well on drawn images such as line drawings, and it does not handle black-and-white images or video images.

K

kbps. (kilobits per second) A speed rating for computer modems that measures (in units of 1024 bits) the maximum number of bits the device can transfer in one second under ideal conditions.

keyword. (1) One of the predefined words of an artificial language. (2) One of the significant and informative words in a title or document that describes the content of that document.

(3) A name or symbol that identifies a parameter. (4) A part of a command operand that consists of a specific character string (such as DSNAME=).

keyword message. A message that takes one or more arguments. A keyword is an identifier followed by a colon (:). Each keyword requires one argument, and the order of the keywords is important. ′hello′ at: 2 put: $H is an example of a keyword message; at: and put: are keyword selectors, 2 and $H are the arguments. See also *message.*

kilobyte. A thousand bytes. Actually, usually 1024 (2^10) bytes. See also byte, bit.

L

LAN. See *local area network*.

library. (1) A shared repository represented by a single file. It stores source code, object (compiled) code, and persistent objects, including editions, versions, and releases of software components. (2) A system object that serves as a directory to other objects. A library groups related objects and allows the user to find objects by name. To identify a specific object to the system, a user needs only to provide the object name, the object type, and the name of the library containing the object.

link. (1) The logical connection between nodes including the end-to-end link control procedures. (2) The combination of physical media, protocols, and programming that connects devices on a network. (3) In computer programming, the part of a program, in some cases a single instruction or an address, that passes control and parameters between separate portions of the computer program. (4) To interconnect items of data or portions of one or more computer programs. (5) In SNA, the combination of the link connection and link stations joining network nodes.

listserv. An Internet application that automatically serves mailing lists by sending electronic newsletters to a stored database of Internet user addresses. Users can handle their own subscribe/unsubscribe actions without requiring anyone at the server location to personally handle the transaction.

literal. An object that can be created by the compiler. A literal can be a number, a character string, a single character, a symbol, or an array. All literals are unique: Two literals with the same value refer to the same object. The object created by a literal is read-only; it cannot be changed.

literal text. Text in an HTML Text part that is passed to the client browser exactly as entered. You can use literal text to code HTML tagging that is not directly supported by the Web Connection parts.

load. A system operation that links the compiled code for a software component from a library into an active image. Loading also performs other operations that enable the component to run, such as linking prerequisites.

local address. In SNA, an address used in a peripheral node in place of an SNA network address and transformed to or from an SNA network address by the boundary function in a subarea node.

local area network (LAN). (1) A network in which devices are connected to one another for communication and can be connected to a larger network. See also *token ring*. (2) A network in which communications are limited to a moderately sized geographic area such as a single office building, warehouse, or campus, and do not generally extend across public rights-of-way. Contrast with *wide area network*.

local node. In the subsystem, the node from which one views the rest of the OSI network—the node for which resources are defined. Contrast with *remote node*.

logical file. A description of how data is to be presented to or received from a program. This type of database file contains no data, but it defines record formats for one or more physical files. The record formats allow a developer to present different views of the data in a physical file.

logical unit (LU). In SNA, a port through which a user gains access to the services of a network. A logical unit can support two types of sessions—with the host, and with other LUs. See *logical unit 6.2*.

logical unit (LU) 6.2. A type of logical unit that supports general communication between programs in a distributed processing environment. LU 6.2 is characterized by (1) a peer relationship between session partners, (2) efficient utilization of a session for multiple transactions, (3) comprehensive end-to-end error processing, and (4) a generic application program interface (API) consisting of structured verbs that are mapped to a product implementation. Synonym for *Advanced Program-to-Program Communications*.

Login. The account name used to gain access to a computer system. Not kept secret (unlike password).

LU type. In SNA, the classification of a LU-LU session in terms of the specific subset of SNA protocols and options supported by the logical units (LUs) for that session, namely:

- The mandatory and optional values allowed in the session activation request
- The usage of data stream controls, function management headers (FMHs), request unit (RU) parameters, and sense codes
- Presentation services protocols such as those associated with FMH usage.

LU types 0, 1, 2, 3, 4, 6.1, 6.2, and 7 are defined.

M

machine-readable information (MRI). Language-sensitive information associated with a computer program, such as program integrated information or softcopy documentation.

management services. In SNA, one of the types of network services in control points (CPs) and physical units (PUs). Management services are the services provided to assist in the management of SNA networks, such as problem management, performance and accounting management, configuration management and change management.

megabyte. A million bytes. A thousand kilobytes. See also byte, bit, kilobyte.

message. In Smalltalk, a communication from one object to another that requests the receiving object to execute a method. A message consists of a reference to the receiving object, followed by a selector indicating the requested method, and (in many cases) arguments to be used in executing the method. There are three types of messages: binary, keyword, and unary.

metaclass. The specification of a class; the complete description of a class′s attributes, behavior, and implementation. Every class has a metaclass, of which it is the sole instance. Contrast with *class*.

method. Executable code that implements the logic of a particular message for a class. In VisualAge, methods are also called scripts. See also *class method*, *instance method*, *private method*, and *public method*.

MIME. (Multipurpose Internet Mail Extensions) A set of Internet functions that extend normal e-mail capabilities and enable nontext computer files to be attached to e-mail. Nontext files include graphics, spreadsheets, formatted word-processor documents, sound files, and so on. Files sent by MIME arrive at their destination as exact copies of the original so that you can send fully formatted word processing files, spreadsheets, graphics images and software applications to other users via simple e-mail. Besides email software, the MIME standard is also universally used by Web servers to identify the files they are sending to Web clients, in this way new file formats can be accommodated simply by updating the browsers′ list of pairs of MIME types and appropriate software for handling each type. See also browser, client, server.

model. A nonvisual part that represents the state and behavior of a real-world object, such as a customer or an account. Contrast with view.

mode name. A symbolic name for a set of session characteristics. For LU 6.2, a mode name and a partner LU name together define a group of parallel sessions having the same characteristics.

module. A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to or output from an assembler, compiler, linkage editor, or executive routine.

monitor. (1) A functional unit that observes and records selected activities for analysis within a data processing system. Possible uses are to show significant departures from the norm, or to determine levels of utilization of particular functional units. (2) Software or hardware that observes, supervises, controls, or verifies operations of a system.

Multiple Virtual Storage (MVS). An IBM licensed program whose full name is the Operating System/Virtual Storage (OS/VS) with Multiple Virtual Storage/System Product for System/370. It is a software operating system controlling the execution of programs.

multireceive dialog. A dialog that takes one data record from the object that initiates the dialog, sends it to a remote program, and returns multiple records from the remote program to the initiating object. Compare to *simple dialog*.

N

NetBIOS. See *Network Basic Input/Output System*.

network. (1) A configuration of data processing devices and software connected for information interchange. (2) An arrangement of nodes and connecting branches. Connections are made between data stations.

network address. In SNA, an address, consisting of subarea and element fields, that identifies a link, a link station, or a network addressable unit. Subarea nodes use network addresses; peripheral nodes use local addresses. The boundary function in the subarea node to which a peripheral node is attached transforms local addresses to network addresses and vice versa. See *local address.*

Network Basic Input/Output System (NetBIOS). An operating system interface for application programs used on IBM personal computers that are attached to an IBM Token-Ring Network.

network layer. (1) In the Open Systems Interconnection reference model, the layer that provides for the entities in the transport layer the means for routing and switching blocks of data through the network between the open systems in which those entities reside. (2) The layer that provides services to establish a path between systems with a predictable quality of service. See *Open Systems Interconnection (OSI).*

network management. The conceptual control element of a station that interfaces with all of the architectural layers of that station and is responsible for the resetting and setting of control parameters, obtaining reports of error conditions, and determining whether the station should be connected to or disconnected from the network.

nil. The object in Smalltalk that means *no value*. All variables initially refer to nil. It is the single instance of the UndefinedObject class.

node. (1) Any device, attached to a network, that transmits and/or receives data. (2) An endpoint of a link, or a junction common to two or more links in a network. (3) In a network, a point where one or more functional units interconnect transmission lines.

node name. In VTAM, the symbolic name assigned to a specific major or minor node during network definition.

nonvisual class. A class in a VisualAge application that specifies a nonvisual part. For example, Person, Address, and BankAccount are nonvisual classes.

nonvisual part. A part that has no visual representation at run time. A nonvisual part typically represents some real-world object that exists in the business environment. Contrast with *view, visual part.*

notebook. In VisualAge, a view that resembles a bound notebook, containing pages separated into sections by tabbed divider pages. A user can turn the pages of a notebook or select the tabs to move from one section to another.

O

object. (1) The basic building block in Smalltalk development. An object is anything that exhibits behavior. All code and data in Smalltalk must be part of an object. (2) On the AS/400 system, an object is a named storage space consisting of a set of characteristics that describes itself and, in some cases, data. Some examples of objects are programs, files, and libraries.

object code. Compiler or assembler output that is itself executable machine code or is suitable for processing to produce executable machine code. Contrast with *source code*.

object factory. A nonvisual part capable of dynamically creating new instances of a specified part. For example, during the execution of an application, an object factory can create instances of a new class to collect the data being generated.

object-oriented programming. A programming methodology built around objects and based on sending messages back and forth between those objects. The basic concepts of object-oriented programming are encapsulation, inheritance, and polymorphism.

object persistency. A characteristic that enables objects to exist beyond the time in which their creating application runs. One use of object persistency is sharing objects among programmers in a development environment.

open system. (1) A system with specified standards that therefore can be readily connected to other systems that comply with the same standards. (2) A data communications system that conforms to the standards and protocols defined by Open Systems Interconnection (OSI).

open systems architecture (OSA). A model that represents a network as a hierarchical structure of layers of functions; each layer provides a set of functions that can be accessed and that can be used by the layer above it.

Note: Layers are independent in the sense that implementation of a layer can be changed without affecting other layers.

Open Systems Interconnection (OSI). (1) The interconnection of open systems in accordance with specific ISO standards. (2) The use of standardized procedures to enable the interconnection of data processing systems.
Note: The OSI architecture establishes a framework for coordinating the development of current and future standards for the interconnection of computer systems. Network functions are divided into seven layers. Each layer represents a group of related data processing and communication functions that can be carried out in a standard way to support different applications.

operating system. Software that controls the execution of programs. An operating system can provide services such as resource allocation, scheduling, input/output control, and data management.

operation. (1) A defined action, namely, the act of obtaining a result from one or more operands in accordance with a rule that completely specifies the result for any permissible combination of operands. (2) A program step undertaken or executed by a computer. (3) An action performed on one or more data items, such as adding, multiplying, comparing, or moving.

option. (1) A specification in a statement, a selection from a menu, or a setting of a switch that may be used to influence the execution of a program. (2) A hardware or software function that may be selected or enabled as part of a configuration process. (3) A piece of hardware (such as a network adapter) that can be installed in a device to modify or enhance device function.

P

panel. (1) A formatted display of information that appears on a terminal screen. Contrast with *screen*. (2) In computer graphics, a display image that defines the locations and characteristics of display fields on a display surface.

parameter. (1) A data element included as part of a message to provide information that the object might need. In Smalltalk, generally referred to as an argument. (2) A variable that is given a constant value for a specified application and that may denote the application. (3) An item in a menu or for which the user specifies a value or for which the system provides a value when the menu is interpreted. (4) Data passed between programs or procedures.

part. A self-contained software object with a standardized public interface, consisting of a set of external features that allow the part to interact with other parts. The parts on the VisualAge parts palette can be used as templates to create instances of objects.

parts palette. In the VisualAge Composition Editor, an organized collection of visual and nonvisual parts used in building composite parts for an application. The parts palette is organized into categories. Application developers can add parts to the palette for use in defining applications or other parts.

password. In computer security, a string of characters known to the computer system and a user, who must specify it to gain full or limited access to a system and to the data stored within it.

path. (1) In a network, any route between any two nodes. (2) The route traversed by the information exchanged between two attaching devices in a network. (3) A command in IBM Personal Computer Disk Operating System (PC DOS) and IBM Operating System/2 (OS/2) that specifies directories to be searched for commands or batch files that are not found by a search of the current directory.

PATH_INFO. A CGI variable, usually transmitted to the CGI program in the form of an environment variable. The PATH_INFO variable contains all path information from the URL following the name of the CGI executable. For a Web Connection application, this information is the same as the VisualAge part name.

persistent object. Instances stored outside of the image. A persistent object must be loaded into virtual or real storage before it can process messages sent to it.

personal computer (PC). A desk-top, free-standing, or portable microcomputer that usually consists of a system unit, a display, a monitor, a keyboard, one or more diskette drives, internal fixed-disk storage, and an optional printer. PCs are designed primarily to give independent computing power to a single user and are inexpensively priced for purchase by individuals or small businesses. Examples include the various models of the IBM personal computers and the IBM Personal System/2 computer.

physical connection. The ability of two connectors to mate and make electrical contact. In a network, devices that are physically connected can communicate only if they share the same protocol.

physical file. A description of how data is to be presented to or received from a program and how data is actually stored in the database. A physical file contains one record format and one or more members.

physical unit (PU). In SNA, a type of network addressable unit (NAU). A physical unit (PU) manages and monitors the resources (such as attached links) of a node, as requested by a system services control point (SSCP) through an SSCP-PU session. An SSCP activates a session with the physical unit in order to indirectly manage, through the PU, resources of the node such as attached links.

pointer. (1) An identifier that indicates the location of an item of data. (2) A data element that indicates the location of another data element. (3) A physical or symbolic identifier of a unique target.

polymorphism. The ability of different objects to respond to the same message in different ways. Different objects can have very different method implementations for the same message. An object can send a message without concern for its underlying implementation.

pool dictionary. A dictionary object whose keys define variables that can be shared by multiple classes. All methods for a class can access the variables in a pool dictionary if the class declares the pool dictionary as part of its scope.

port. (1) A place where information goes into or out of a computer, or both. For example, the serial port on a personal computer is where a modem would be connected. (2) On the Internet port often refers to a number that is part of a URL, appearing after a colon (:) right after the domain name. Every *service* on an Internet server listens on a particular port number on that server. Most services have standard port numbers; Web servers normally listen on port 80. Services can also listen on nonstandard ports, in which case the port number must be specified in a URL when accessing the server. (3) Refers to translating a piece of software to bring it from one type of computer system to another. See also domain name, server, URL.

POST. One of the methods used in HTTP requests. A POST request is used to send data to an HTTP server. See also GET.

prerequisite application. An application required by another application for it to function successfully. An application can extend or reference one or more of the prerequisite application′s classes. In team development, prerequisite applications are particular versions or editions of applications.

presentation layer. In the Open Systems Interconnection reference model, the layer that provides for the selection of a common syntax for representing information and for transformation of application data into or from this common syntax.

primary logical unit (PLU). In SNA, the logical unit (LU) that contains the primary half-session for a particular LU-LU session. Each session must have a PLU and secondary logical unit (SLU). The PLU is the unit responsible for the bind and is the controlling LU for the session. A particular LU may contain both primary and secondary half-sessions for different active LU-LU sessions.

primary part. In a composite part constructed with the VisualAge Composition Editor, the subpart whose public interface is fully exposed on the public interface of the composite part. The primary part is transparently visible to parts outside the composite part and is the subpart with which most interaction will take place.

private class. In VisualAge or IBM Smalltalk, a class that is not part of the system API but is provided as part of the internal functioning of the system. A private class is not visible outside its containing application. If you use a VisualAge or IBM Smalltalk class marked as private, you might have to modify your code in order for it to work with a future release of VisualAge or IBM Smalltalk. Contrast with *public class*.

private method. In VisualAge or IBM Smalltalk, a method that is not part of the system API but is provided as part of the internal functioning of the system. If you use a method marked as private, you might have to modify your code in order for it to work with a future release of VisualAge or IBM Smalltalk. Contrast with *public method*. Individual application development projects can use the public and private designations as a means of organizing their code.

process. In Smalltalk, a sequence of actions described by expressions and performed by the system′s virtual machine.

property. A unique characteristic of a part.

protocol. (1) The set of all messages to which an object will respond. (2) Specification of the structure and meaning (the semantics) of messages that are exchanged between a client and a server. (3) A set of semantic and syntactic rules that determine the behavior of functional units in achieving communication. (4) In SNA, the meanings of and the sequencing rules for requests and responses used for managing the network, transferring data, and synchronizing the states of network components. (5) A specification for the format and relative timing of information exchanged between communicating parties.

proxy. An application gateway from one network to another for a specific network application like Telnet of FTP, for example, a firewall′s proxy Telnet server performs authentication of the user and then lets the traffic flow through the proxy as if it were not there. Function is performed in the firewall and not in the client workstation, causing more load in the firewall. Compare with socks.

PRPQ. Programming Request for Price Quotation. A customer request for a price quotation for a licensed program to be designed especially for a particular group of customers or an application. Documentation for the program is provided only to those customers who order the PRPQ.

pseudocode. An artificial language used to describe computer program algorithms without using the syntax of any particular programming language. The code requires translation before execution.

public class. A class that is provided as part of the VisualAge or IBM Smalltalk API. A public class is visible to applications other than its containing application. Public classes are designed to function with future releases of VisualAge or IBM Smalltalk and with operating systems supported by VisualAge or IBM Smalltalk. Contrast with *private class*.

public interface. A set of external features that enable a part to interact with other parts. A part′s public interface is made up of three characteristics: attributes, actions, and events.

Public Interface Editor. A VisualAge view used to create and modify attributes, actions, and events, which together make up the public interface of a part.

public method. A method that is provided as part of the VisualAge or IBM Smalltalk API. Public methods are designed to function with future releases of VisualAge or IBM Smalltalk and with operating systems supported by VisualAge or IBM Smalltalk. Contrast with *private method*. Individual application development projects can use the public and private designations as a means of organizing their code.

Q

query specification. A database query definition. All queries issued to a database by VisualAge or IBM Smalltalk must be defined by a query specification.

quick form. In the VisualAge Composition Editor, a menu option that enables application developers to quickly create a default view for a part.

R

receive. To obtain and store information transmitted from a device.

receiver. The object that receives a message. Contrast with *sender*.

record. (1) (ISO) In programming languages, an aggregate that consists of data objects, possibly with different attributes, that usually have identifiers attached to them. In some programming languages, records are called structures. (2) (TC97) A set of data treated as a unit. (3) A set of one or more related data items grouped for processing. (4) In VTAM, the unit of data transmission for record mode. A record represents whatever amount of data the transmitting node chooses to send.

RecordStructure. An object that contains information about the format, structure, and types of the data it contains.

RecordStructure classes. Classes that provide a flexible, open architecture for converting Smalltalk objects to and from data types of other languages.

release. A system operation on a component that changes its containing component′s configuration. Releasing a component adds its released edition or version to the configuration for its containing component. When a containing component is loaded into an image, the released editions or versions of the components it contains are also loaded.

remote. Concerning the peripheral parts of a network not centrally linked to the host processor and generally using telecommunication lines with public right-of-way.

remote node. Any node other than the local node. Contrast with *local node*.

repository. (1) An organized, shared body of information that can support business and data-processing activities. (2) In VisualAge or IBM Smalltalk, the multiuser library that stores components such as applications, classes, and methods created by application developers. It stores source code, object code, and persistent objects.

request. A service primitive issued by a service user to call a function supported by the service provider.

reset button. A type of push button that can appear on a form. A reset button restores all input fields to their default states.

resource. (1) Any facility of the computing system or operating system required by a job or task, and including main storage, input/output devices, the processing unit, data sets, and control or processing programs. (2) In the NetView program, any hardware or software that provides function to the network.

response. A service primitive issued by a service user to complete the procedures associated with a confirmed service.

return code. (1) A value (usually hexadecimal) provided by an adapter or a program to indicate the result of an action, command, or operation. (2) A code used to influence the execution of succeeding instructions.

return value. An object or data type that a receiver object passes to a sender object in response to a message.

router. A network device that enables the network to reroute messages it receives that are intended for other networks. The network with the router receives the message and sends it on its way exactly as received. In normal operations, they do not store any of the messages that they pass through.

routine. Part of a program, or a sequence of instructions called by a program, that may have some general or frequent use.

RPG. A programming language designed for writing application programs for business data processing requirements. The application programs range from report writing and inquiry programs to applications such as payroll, order entry, and production planning.

S

scratch edition. A mutable and private copy of an application for a user who is not necessarily the application′s manager. The scratch edition only exists in that user′s image. Using a scratch edition, one can modify an application version, and the existing classes contained in it, without actually creating a new edition. Each scratch edition has << >> displayed around the edition timestamp or version name. Contrast with edition, version.

screen. An illuminated display surface; for example, the display surface of a CRT or plasma panel. Contrast with *panel*.

script. A series of Smalltalk statements that implement an action for a part. Scripts are equivalent to Smalltalk methods.

Script Editor. A VisualAge view that enables a developer to implement part behavior by writing Smalltalk scripts (methods). The Script Editor shows all the objects that can be referenced by a method and all previously defined methods of a part′s class.

selector. The component of a message that specifies the requested operation. There are three kinds of selectors: binary, keyword, and unary.

sender. An object that sends a message to another object. On the level of code implementation, the sender is considered to be the sending method within the class or instance that issues the message. Contrast with *receiver*.

server. (1) A computer that provides services to multiple users or workstations in a network; for example, a file server, print server, or mail server. (2) An object that performs one or more tasks on behalf of a client. The server can be a computer (a file server), a specific process on a server, or a distributed object. A single server machine could have several different server software packages running on it, thus providing many different servers to clients on the network. See also client, network.

service. (1) A specific behavior that an object is responsible for exhibiting. (2) In network architecture, the capabilities that a layer and the layers closer to the physical media provide to the layers closer to the end user. (3) A set of service primitives that a layer provides to the layer above it.

service object. A nonvisual part that gives access to the outside world of a VisualAge for Smalltalk image.

session. (1) A connection between two application programs that allows them to communicate. (2) In SNA, a logical connection between two network addressable units that can be activated, tailored to provide various protocols, and deactivated as requested. (3) The data transport connection resulting from a call or link between two devices. (4) The period of time during which a user of a node can communicate with an interactive system, usually the elapsed time between logon and logoff. (5) In network architecture, an association of facilities necessary for establishing, maintaining, and releasing connections for communication between stations. (6) A series of CGI queries that come from the same client and belong to the same logical sequence. A session is identified by a unique session key, which is generated by VisualAge. A session begins when a client initially connects (without a session key) and ends when a specified timeout period has elapsed since the last connection.

session key. A unique string, generated automatically by VisualAge, that identifies a session. All requests and replies carry the session key in hidden HTML data fields.

session layer. The layer that provides the services that organize and synchronize communications between functional units in different open systems located in the presentation layer.

settings view. A view of a part that provides a way to display and set the attributes and options associated with the part.

simple dialog. A dialog that takes one data record from the initiating object and sends it to a remote program, which returns the result to the initiating object. Compare to *multireceive dialog*.

Smalltalk (ST). (1) A complete programming environment for developing object-oriented applications. Smalltalk is a pure implementation of object-oriented concepts; every entity in the environment is an object. (2) The name of the programming language that the Smalltalk programming environment supports. (3) In the IBM Smalltalk programming environment, the name of the System Dictionary containing the global variables.

SNA. See *Systems Network Architecture*.

socket. (1) In the TCP/IP environment, a socket is an endpoint for communication between processes or applications. A socket that can send data to and receive data from a remote node is a connected socket. (2) Synonym for *port*.

socks. Software to intercept and redirect all TCP/IP requests at the firewall. It handles data to and from applications such as Telnet, FTP, Mosaic, and Gopher. Provides users in a secured network access to resources outside the network by directing data through the firewall. Firewall users must use client programs specifically designed to work with the *sockd* server.

software. (1) Programs, procedures, rules, and any associated documentation pertaining to the operation of a system. (2) Contrast with *hardware*.

SOM. See *system object model*.

source code. Compiler or assembler input, written in a source language. Contrast with *object code*.

source file. A file of programming code that is not compiled into machine language. A source file can be created by the specification of FILETYPE(*SRC) on the Create command. A source file can contain source statements for such items as high-level language programs and data description specifications (DDS).

SQL. Structured Query Language. A language used to access relational databases.

SQL Editor. An interactive tool for creating structured query language (SQL) statements. It consists of a set of dialogs that prompt the user for information about database tables and use that information to generate SQL statements.

statement. A language syntactic unit consisting of an operator, or other statement identifier, followed by one or more operands.

stored procedure. A procedure stored in a database system that contains SQL and other control statements.

structured query language (SQL). A language used to access relational databases.

subapplication. An application contained by another application. Using subapplications, one can organize the classes of an application into a tree of subapplications or isolate the parts of an application that are platform-specific.

subclass. A class that inherits behaviors and specifications (in other words, methods and variables) from another class. Contrast with *superclass*.

subclass type. In VisualAge or IBM Smalltalk, an indication of how a subclass inherits instance variables from its superclass.

submit button. A type of push button that can appear on a form. A submit button initiates a connection to the HTTP server and sends a CGI query, using the data from the input fields as parameters.

subpart. A part that is embedded within a composite part.

subsystem. A secondary or subordinate system, or programming support, usually capable of operating independently of or asynchronously with a controlling system.

superclass. A class from which another class inherits behaviors and specifications (in other words, methods and variables). Contrast with *subclass*.

symbol. In Smalltalk, an object that represents a string used as a name within the system. A symbol literal is a sequence of characters preceded by the pound sign (#) with no embedded blanks, such as #George or #messageSelector.

synchronous. (1) Pertaining to two or more processes that depend on the occurrences of a specific event such as common timing signal. (2) Occurring with a regular or predictable timing relationship.

system. In data processing, a collection of people, machines, and methods organized to accomplish a set of specific functions.

system component. A component that manages storage of code and access to that stored code. A library file is a system component that stores and manages code. A user object is a system component that represents a person who can use a library.

system object model (SOM). An object-structured protocol that enables applications to access and use objects and object definitions, regardless of the programming language created that them, with no need to recompile the application.

systems management. The process of monitoring, coordinating, and controlling resources within open systems.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks.

T

TCP/IP. (Transmission Control Protocol/Internet Protocol) The basic programming foundation that carries computer messages around the globe via the Internet. The suite of protocols that defines the Internet. Originally designed for the UNIX operating system, TCP/IP software is now available for every major kind of computer operating system. To be truly on the Internet, your computer must have TCP/IP software.

team programming. Development of a system, program, or application suite by a team of two or more programmers or application developers.

tear-off attribute. An attribute that an application developer has exposed to work with as though it were a standalone part.

Telnet. An Internet protocol that lets you connect your PC as a remote workstation to a host computer anywhere in the world and to use that computer as if you were logged on locally. You often have the ability to use all of the software and capability on the host computer, even if it′s a huge mainframe.

temporary variable. A variable whose scope is limited to the Smalltalk method or block in which it is defined. A temporary variable takes an assigned value.

terminal. A device that is capable of sending and receiving information over a link; it is usually equipped with a keyboard and some kind of display, such as a screen or a printer.

token ring. A network with a ring topology that passes tokens from one attaching device (node) to another. A node that is ready to send can capture a token and insert data for transmission.

tool bar. In the VisualAge Composition Editor, the strip of icons along the top of the free-form surface. The tool bar contains tools to help construct composite parts. These tools are also available through the Tools pull-down menu of the Composition Editor window.

transaction. (1) In client/server transaction processing, a business activity or set of activities that transforms a database from one state to another (for example, making an airline reservation or custom ordering an automobile). (2) In an SNA network, an exchange between two programs that usually involves a specific set of initial input data that causes the execution of a specific task or job. Examples of transactions include the entry of a customer's deposit that results in the updating of the customer's balance, and the transfer of a message to one or more destination points.

transaction program. A program that processes transactions in or through a logical unit (LU) type 6.2 in an SNA network. Application transaction programs are end users in an SNA network; they process transactions for service transaction programs and for other end users. Service transaction programs are IBM-supplied programs that typically provide utility services to application transaction programs.

Transcript window. The main controlling window in Smalltalk.

Transmission Control Protocol/Internet Protocol (TCP/IP). A set of protocols that allow cooperating computers to share resources across a heterogeneous network.

U

uniform resource locator (URL). A standard identifier for a resource on the World Wide Web, used by a Web browser to initiate a connection. The URL includes the communications protocol to use, the name of the server, and path information identifying the object to be retrieved on the server.

usability. The quality of a system, program, or device that enables users to easily understand and conveniently use it.

user interface (UI). The hardware, software, or both that enables a user to interact with a computer. In VisualAge, user interface normally refers to the visual presentation with which a user interacts and its underlying software.

user profile. A file that contains the user′s password, the list of special authorities assigned to a user, and the objects the user owns. It is used by the system to verify the user′s authorization to read or use objects, such as files or devices, or to run the jobs on the system. Each user profile must have a unique name.

user space. In OS/400 application programming interfaces, an object consisting of a collection of bytes that can be used for storing any user-defined information. The system-recognized identifier for the object type is *USRSPC.

V

variable. (1) A storage place within an object for a data element. The data element is an object, such as a number or date, stored as an attribute of the containing object. (2) In VisualAge, a part that receives an identity at run time. A variable by itself contains no data or program logic; it must be connected in such a way that it receives runtime identity from a part elsewhere in the application.

version. In the VisualAge or IBM Smalltalk team programming environment, an edition of a software component that cannot be changed. Each version has a version name. Contrast with *edition*.

view. A composite visual part. A view can display and change the underlying nonvisual objects of an application. In VisualAge, views are both the end result of developing an application and the basic unit of composition of user interfaces. Compare to *visual part*.

Virtual Storage Access Method (VSAM). An access method for direct or sequential processing of fixed and variable-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry-sequence), or by relative-record number.

Virtual Telecommunications Access Method (VTAM). An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability.

visible class. A class that another class can subclass or refer to by name in a method. *Visible* refers to the scope in which the class name can be used. For a class in a given application, visible classes include:

- All classes defined in the same application
- All public classes defined in any subapplication
- All prerequisite classes, including prerequisites of prerequisites to the lowest level

visual part. A part that has a visual representation at run time. Visual parts, such as windows, push buttons, and entry fields, make up the user interface of an application. Compare to *view*. Contrast with *nonvisual part*.

visual programming tool. A tool, such as VisualAge, that provides a means for specifying programs graphically. Application programmers write applications by manipulating graphical representations of components.

W

WAN. (Wide Area Network)— Any internet or network that covers an area larger than a single building or campus. See also Internet, LAN, network.

wide area network (WAN). A data communications network designed to serve an area of hundreds or thousands of miles—for example, public and private packet-switching networks, and national telephone networks.

widget. An object that provides a user-interface abstraction; for example, a scrollbar widget. Widgets support obtaining input from the user and displaying output to the user.

window. (1) A rectangular area of the screen with visible boundaries in which information is displayed. Windows can overlap on the screen, giving the appearance of one window being on top of another. (2) In the VisualAge Composition Editor, a part that can be used as a container for other visual parts, such as push buttons.

workstation. (1) An I/O device that allows either transmission of data or the reception of data (or both) from a host system, as needed to perform a job; for example, a display station or printer. (2) A configuration of I/O equipment at which an operator works. (3) A terminal or microcomputer, usually one connected to a mainframe or network, at which a user can perform tasks.

World Wide Web. (WWW) (W3) (the Web) An Internet client-server distributed information and retrieval system based upon HTTP that transfers hypertext documents across a varied array of computer systems. The Web was created by the CERN High-Energy Physics Laboratories in Geneva, Switzerland in 1991. CERN boosted the Web into international prominence on the Internet.

List of Abbreviations

AIX	Advanced Interactive eXecutive
APAR	authorized program analysis report
APPC	advanced program-to-program communication
APPN	advanced peer-to-peer networking
BMP	bitmap
CAE	Client Application Enabler
CDROM	compact disk read only memory
CG	common graphics
CGI	Common Gateway Interface
CICS	customer information control system
CLI	call level interface
CORBA	Common Object Request Broker Architecture
CPI-C	common programming interface for communications
CPU	central processing unit
CRC	class responsibility collaborators
CUA	Common User Access
DAP	developer assistance program
DBA	database administrator
DBF	database file
DBMS	database management system
DB2	Database 2
DDCS	distributed database connection services
DDL	database definition language
DDM	Distributed Data Management
DDS	data description specifications
DLL	dynamic link library
DSN	data set name
DSOM	distributed system object model
ECS	electronic customer support
EPI	external presentation interface
ESA	enterprise systems architecture
FTP	File Transfer Protocol
GIF	graphic interchange format
GPF	general protection fault
GUI	graphical user interface
HLLAPI	high level language application program interface
HPFS	high performance file system
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPD	Hypertext Transfer Protocol daemon
IBM	International Business Machines Corporation
IDL	interface definition language

ILE	integrated language environment
IMS	information management system
IPC	inter-processor communication
IPMD	IBM presentation manager debugger
ISO	International Organization for Standardization
ISP	Internet service provider
ISV	independent software vendor
ITSO	International Technical Support Organization
JDBC	Java Database Connectivity
LAN	local area network
LF	logical file
MB	megabyte
MIME	Multipurpose Internet Mail Extensions
MLE	multiline edit
MQ	message queueing
MRI	machine readable information
MVS	multiple virtual storage
NLS	national language support
NT	new technology
ODBC	Open Database Connectivity
OIDL	object interface definition language
OLTP	on-line transaction processing
OMT	object modeling technique
OOP	object-oriented programming
OOSE	object-oriented software engineering
ORB	object request broker
OS	operating system
OS/2	Operating System/2
PCS	PC/Support
PM	presentation manager
PTF	program temporary fix
RAM	random access memory
RDBMS	relational database management system
RFT	request for technology
RPC	remote procedure call
RPG	report program generator
SGML	standard generalized mark-up language
SHTTP	Secure Hypertext Transfer Protocol
SOM	system object model
SQL	structured query language
SSL	secure sockets layer
TCP/IP	Transmission Control Protocol/Internet Protocol
URL	uniform resource locator
VM	virtual machine
VMT	visual modeling technique
WWW	world wide web

Index

ITSO Redbook Evaluation

VisualAge for Smalltalk Handbook Volume 1: Fundamentals
SG24-4828-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@vnet.ibm.com

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction ____________

Please answer the following questions:

Was this redbook published in time for your needs? Yes____ No____

If no, please explain:

____

____

____

____

What other redbooks would you like to see published?

____

____

____

Comments/Suggestions: **(THANK YOU FOR YOUR FEEDBACK!)**

____

____

____

____

____

Printed in U.S.A.

SG24-4828-00