PLAINTIFF'S EXHIBIT
3153
Comes v. Microsoft

Distributing Applications with the Distributed Application Development Toolkit for OS/2

Mike Starkey and Rob Stevenson
IBM Canada Laboratory
1150 Eglinton Ave. East
North York, Ontario M3C 1H7
Canada

Abstract

The proliferation of networks has provided a rich environment for creating distributed applications. Distributed applications are those applications where some parts run on one machine and other parts run on other machines. Separating the parts of a program may be done for a number of reasons. One reason may be that one computer has an operating system that provides a better visual environment than one that might perform calculations better or faster. Whatever the reasons for distributing an application, it is not a trivial task. Writing applications that support communication among machines with different operating systems and different representations of data types can be tedious and error-prone. The Distributed Application Development Toolkit for OS/2 addresses this problem by providing simple notebooks for selecting communication options and generating all of the necessary code for distributing the application.

Introduction

Distributing applications across multiple computers has become an important part of application development. Different computers have different strengths. By placing parts of the application on different computers to take advantage of these strengths, the application can be much improved over one running on a single computer.

However, the differences between these computers are deeper than the physical appearance of the machines or their user interface. The differences are imbedded in the architecture of the machines and are manifested in everything from the low-level data interaction mechanisms to the programming environment and the operating system. Additionally, these architectures differ in the way they run software, which can be confusing for users.

Handling these differences in architectures, programming environments and computer characteristics is difficult for programmers who wish to take advantage of distributed applications. Programmers must also be aware of the middleware characteristics they need. For example, a basic TCP/IP implementation is a good protocol for prototyping but does not handle some of the features that users expect, such as guaranteed message delivery when disconnected or multiple levels of data protection or user authentication.

The Distributed Application Development Toolkit for OS/2 handles these differences in middlewares, machine architectures and application characteristics. Users can choose the

HIGHLY CONFIDENTIAL

MSC 00545179

MSC0000545179

middlewares they wish to use by selecting from a list of supported middlewares and then setting the middleware-specific options using notebooks. Options that are not compatible are disabled. Once the options have been selected, the code can be generated.

The granularity of distribution is an important consideration. The Distributed Application Development Toolkit for OS/2 allows distribution at the part level. A part is the encapsulation of behaviour and state. The behaviour is described by some interface description. The description is generally contained in either a C++ class definition or in an interface definition language such as OSF's DCE IDL. The behaviour is described by the implementation of the interface. To distribute a part, the interface is all that is required for generating the code to support the communication among parts. There will be semantics about the behaviour that the programmer may want to capture in the tool, but this is done through selecting options in the notebooks.

The Visual Builder tool handles the definition of parts and their interfaces. Visual Builder allows parts to be defined hierarchically. The top-level part becomes the application. The Distributed Application Development Tool for OS/2 takes the parts in this top-level part and allows them to be distributed on multiple computers.

A Sample Application

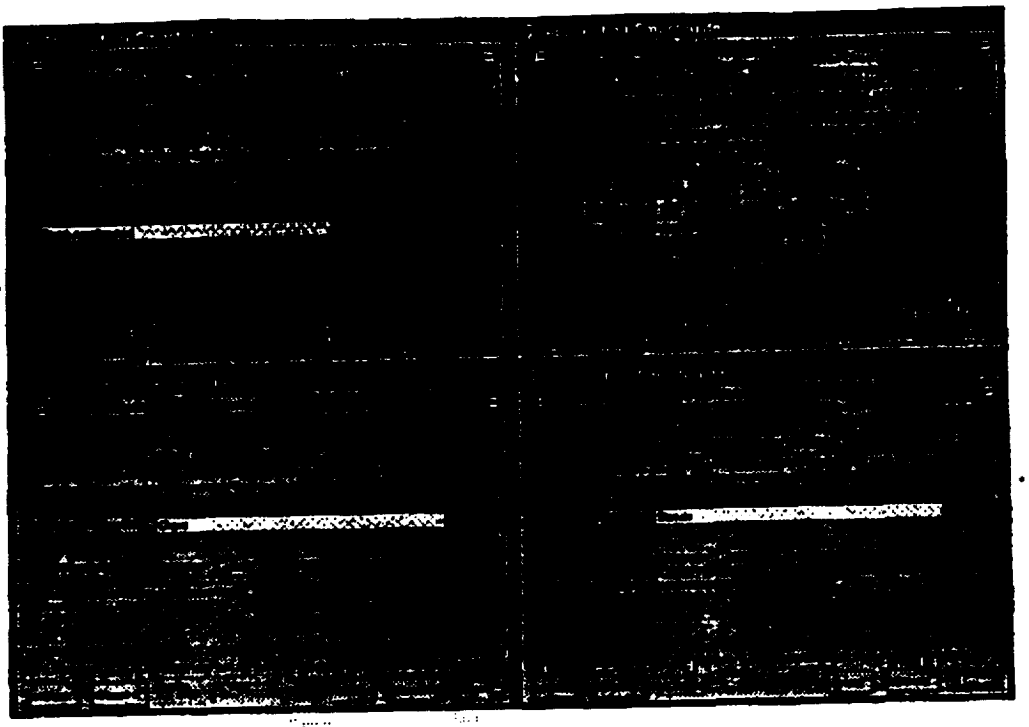
To illustrate the simplicity of creating a distributed application, we will use a simple distributed stack as an example. The C++ interface to the stack appears in Figure 1.

```
#ifndef _Stack_hpp_
#define _Stack_hpp_

#include <string.hpp>
#include <iseq.hpp>

class Stack {
public:
    void push(string element);
    string pop();
private:
    iSequence<string> stack_;
};

#endif
```

Figure 1. The interface to the Stack class.

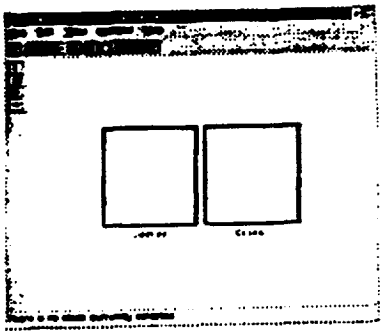
The class has two methods: push() and pop(), which push a string onto the stack and pop one off

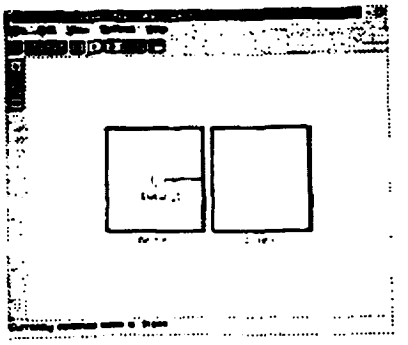
HIGHLY CONFIDENTIAL



Next, create the object in the server partition. Open the application folder and select Distribute/Edit partitions. The Partition window appears with the two partitions that were created in the Smart Guides. Select File/Import via .hpp and import the Stack.hpp file. The class window shows the currently selected class. Ensure that the Stack class is selected and select the object icon on the frame window. Then create an instance of the class in the server partition by pressing the left mouse button.

Next, create an association from the client partition. The association ensures that a client proxy is generated in the client partition. Select the association icon. Select the client partition and then the object in the server partition. Middleware options can be changed for the partition, object or association if required. For TCP/IP, the default options allow a working program to be generated, compiled and run. The Partition window can now be saved and exited using the File/Exit selection.





HIGHLY CONFIDENTIAL

the stack, respectively. Figure 2 shows the implementation of the stack class.

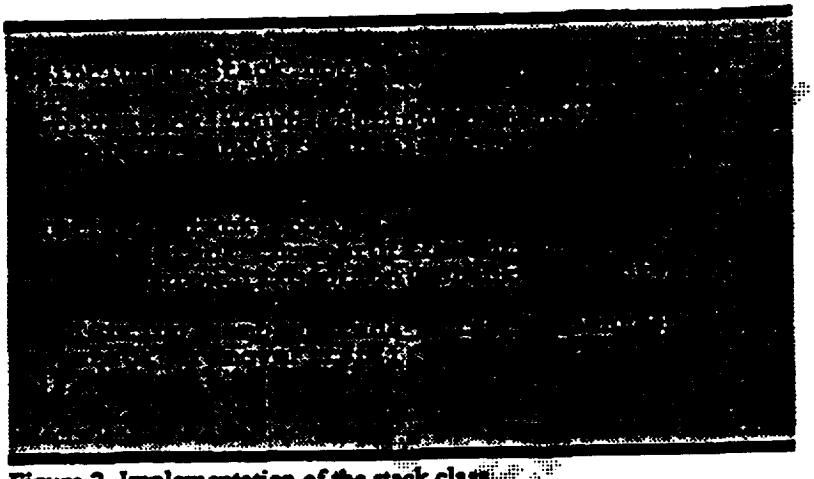


Figure 2. Implementation of the stack class.

The pop() method also raises an exception if there are no elements left in the stack. This illustrates how exceptions are passed from the server to the client in a distributed application.

Creating the Distributed Application Infrastructure

First, you create the application with two empty partitions using the WorkFrame component of VisualAge C++. Open the VisualAge C++ folder and double-click on the Project Smarts icon. Select the Distributed Logic Application option. Give the application a name; for example, *Stack Example*. Next select a two-tier application. Name the client and server partitions; for example, *Client* and *Server*. Select TCP/IP as the protocol for the client and select Visual Builder as the option. Next, select TCP/IP for the server partition but do not select Visual Builder. Select Done when complete and specify a location for the application folder. Figure 3 shows you the main steps in using WorkFrame to create your initial partitions.

HIGHLY CONFIDENTIAL

MSC 00645182

MSC0000545182

Generating the code

The code that supports the distribution now needs to be generated for each partition. Open the client partition folder. Select Distribute/Generate Partition Parts. This generates a number of files including class-level files. These files provide the packaging and unpackaging of the parameter data and convey any class-level middleware options that have been set.

The class-level files are:

- Stack.hpd - Common definition file
- Stack.hpc - Client packaging/unpackaging code
- Stack.hps - Server unpackaging/packaging code
- Stack.dec - DCE client support routines
- Stack.des - DCE server support routines
- Stack.ded - DCE common support routines
- Stack.idl - DCE IDL definition file

Only one set of class files is required for all instances of the class. These files are generated into the _shared_ subdirectory and show up in all of the partition folders.

For each instance of the class, a set of files are generated. A shared file is generated which contains the identifier of the object. This file is shared between the server partition where the object resides and all of the client partitions that access the object.

The object level files are:

- Stack_1.hpd - Instance 1 common definition file
- Stack_1.hpc - Instance 1 client setup file
- Stack_1.hps - Instance 1 server setup file
- Stack_1.vbc - Instance 1 client Visual Builder interface definition file
- Stack_1.vbs - Instance 1 server Visual Builder interface definition file

The Stack_1.hpd is generated into the _shared_ directory. The Stack_1.hpc and Stack_1.vbc files are generated into the client partition folder.

In addition to the class and instance files, a partition-level file is generated. The file contains the partition setup code. There are some methods that can be performed on the environment and appear as methods on the partition class. This class is called DADTPartition. One instance of this class is created for each partition. The interactions that are available on the partition are a generateEvent() method for generating user events easily. The text specified as the parameter to this method appears in the Event Viewer. The second method that is only generated for server or agent partitions is a stopListening() method. This method causes the server or agent to stop receiving calls from clients.

The partition-level files that are generated for the client partition are:

- client.hpd - Partition setup file and DADTPartition class definition
- client.vbe - Visual Builder part interface to the DADTPartition class

Next, generate the files for the server partition. Enter the server partition and select Distribute/Generate Partition Parts. This generates the server object files and the server partition-level files. To instantiate the objects, the server process must contain a main(). The server can be created using Visual Builder and instantiating all of the objects described by the .vbs files. However, in this sample you will generate the main(). Use the Distribute/Generate Partition Main() option to do this.

The following additional files now appear in the server partition:

- server.hpd - Partition setup file and DADTPartition class definition
- server.vbe - Visual Builder part interface to the DADTPartition class
- server.cpp - File containing the main() function

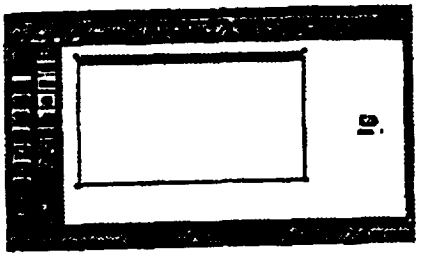
Using the Distributed Object

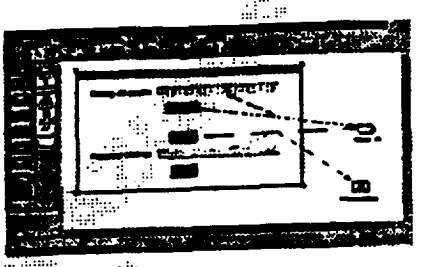
Now that the object has been defined and all of the options have been set for the partition, class, object and association, it can be used to develop a distributed application. To produce the client application, select Project/Visual in the client partition. Use the File/Import menu to import Stack_1.vbc. Then create a new visual part; for example, *Stack Window*. Use the Options/Add menu to add the Stack_1* part. Place it somewhere in the Composition Editor.

Place two push-buttons and two entry fields on the frame window. Change the text of one push-button to Push and the text of the other to Pop. Connect the buttonClick event of the Push push-button to the push() action of the Stack_1 part. Then connect one of the entry fields as the element parameter to the push() call.

To connect the pop() event connect the buttonClick event to the push() action on the Stack_1 part. Connect the actionResult to the other entry field. Since the pop() method can raise an exception, connect up an exception display window. Add the message window to the Visual Builder surface. Connect the exceptionOccurred option on the pop() connection to the message window as the showException attribute. Use the File/Save and Generate option to generate the part source. Then use the File/Save and Generate option to generate the part main(). Visual Builder can now be closed since the application is complete.

HIGHLY
CONFIDENTIAL





Building the Application

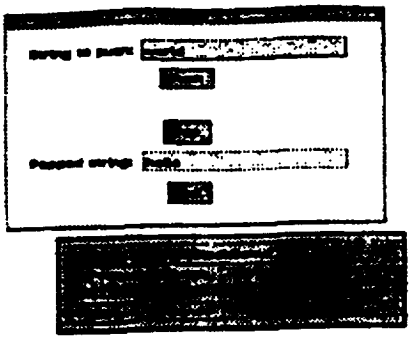
The client and server partitions must now be built to form two processes. In each partition, the compiler and link options must be modified. To change the compiler option select Options/Compiler and select the C++ Compiler. Using the Object tab, select Multithreaded library and Dynamic linkage. To modify the link options, use the Options/Link option. Use the Template tab and check the template option. The Compile::C++ Compiler should be associated. For the client partition, the Generation option should be changed to generate a PM application.

Running the Application

The TCP/IP protocol does not have a name service that can be used for distributed objects. Our implementation over TCP/IP includes a shared file for registering the location of the server containing the object. The location of this file must be defined for all processes using the object. This location can be set using the Object and Association notebooks in the Partition window or can be set at run time. The variable that is set at run time is IDATCP::NAMESERVICE. For this example, we'll assume that both processes are run on the same machine. Select the View/Tools setup and use the Variables Add option to add the value of c:\tcp.ns for the variable IDATCP::NAMESERVICE. Set this in both partitions. Then execute the srvr.exe process in the server partition and start the client process in the client partition by executing vbmain.exe.

To test the application, push "hello", then push "world". Then pop "world", pop "hello". Select pop again and an exception is displayed. Note that this exception was raised on the server and transported to the client where it is displayed in an exception window. Figure # shows the Exception window. To stop the application, use Ctrl-C in the srvr.exe window and exit theclient window.

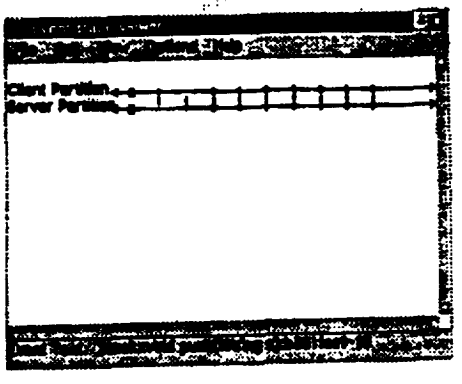
HIGHLY
CONFIDENTIAL



Running the Application with the Event Monitor

Debugging distributed applications can be difficult. The event trace facility helps you debug distributed applications by showing all of the method calls between application threads. Tracing the application does not require recompiling the application code. To start the event monitor, type **evmon /pr** on an OS/2 command line. Ensure that the enable trace box is checked. Specify a port greater than 1000; for example, *10001*. Select the host connection as local and specify a filename; for example, *c:\stack.edb*. Press continue. Now start the server and then the client again. Perform the same operations in the client window. The client and server processes can now be terminated.

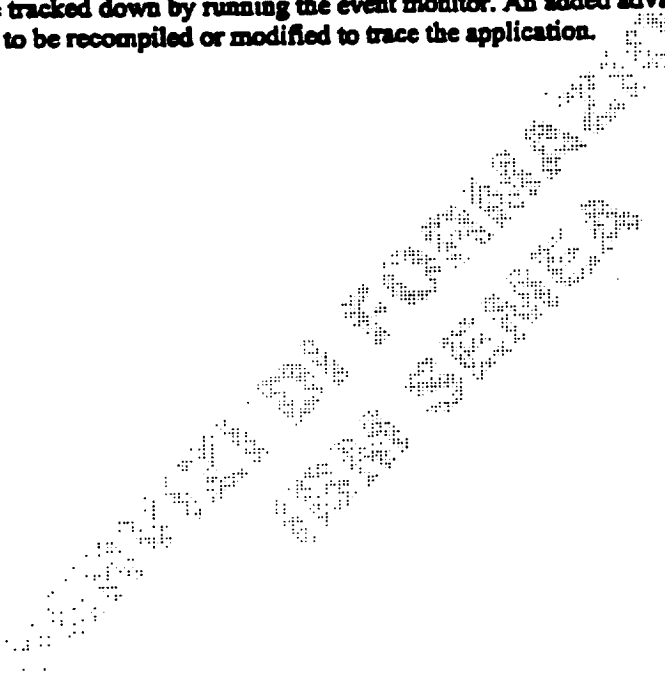
To view the event trace log, the event monitor needs to be closed down. Bring up the window list using the Ctrl-Esc key sequence. Select the Event Monitor and close it. To view events, select the Project/Analyze and Distributed Trace. Use the File/Open option to open the **c:\stack.edb** file. Two horizontal lines are displayed representing the two threads, one in the client and the other in the server. The lines connecting the threads show the method calls between the threads. Selecting one of these lines displays the method that was called.



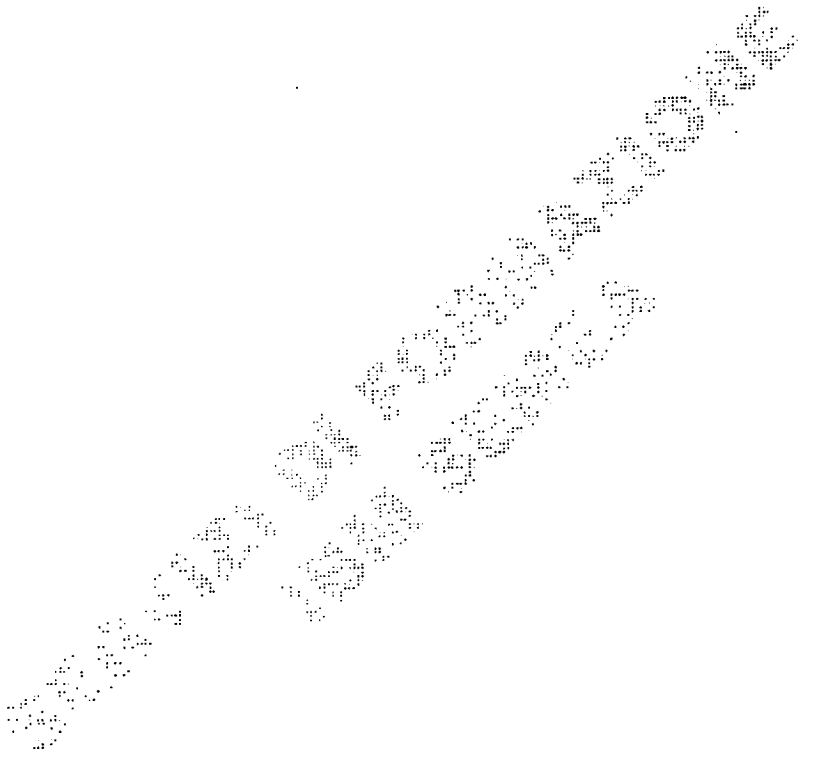
HIGHLY
CONFIDENTIAL

Conclusions

The distributed application development toolkit for OS/2 makes creating distributed applications easy. Integration with the Visual Builder enables programmers to develop distributed applications, possibly without writing a line of code. Problems with lost communication between objects can be tracked down by running the event monitor. An added advantage is that the code does not need to be recompiled or modified to trace the application.



HIGHLY
CONFIDENTIAL



HIGHLY
CONFIDENTIAL

MSC 00545188

VisualAge C++ Version 3.0

FACT SHEET

Attendees Choice Award for the Best Object Technology Product

Object World - Boston, March 22,1995

Best New Object Technology Development Product

Object World - Boston, March 22,1995

"IBM has seen the future, and it is objects."

InformationWeek - April 1995

Product Description

- New VisualAge development environment for C++ programmers
- Visual construction of parts
- Complete professional C++ programming environment with 32-bit compiler and extensive development tools
- Based on IBM Open Class - complete C++ class libraries
- Significant upgrade version to IBM's C Set ++ product.

Key features

- Visual construction of parts
- Object-oriented access to relational data
- IBM Open Class libraries
- Direct-to-SOM
- New development environment
- New browser & 32-bit linker

Key Messages

- The new visual age of C++ development has arrived.
- Powerful multi-platform C++ environment from the desktop to the mainframe.
- Extension of the VisualAge family to include Smalltalk and C++.

Operating Environments

- Version 3
  - OS/2 2.1
  - OS/2 Warp
- Next release, late 1995
  - Windows 95
  - Windows NT
  - Power Macintosh

Requirements

Hardware

- 80386 minimum (80486 or higher recommended)
- 80386 machines should have an 80387 coprocessor for floating point applications

Display

- VGA minimum (SVGA recommended)

Memory

- C++ Development
  - 12MB RAM minimum (16MB RAM recommended)
- C++ Visual Development
  - 16MB RAM minimum (24MB RAM recommended)

Disk Space

- 110MB for all tools
- 70MB for samples and documentation
- 30MB (minimum) for swap space

Software

- IBM OS/2 V2.11 or higher (OS/2 Warp recommended)

Part Numbers and Prices

Media	Part No.	US List Price
3.5" Disks + Docs	30H1664	$525
CD-ROM only	30H1665	$449
CD-ROM w/ Docs	30H1666	$489

6/95

HIGHLY CONFIDENTIAL

VisualAge C++ Version 3.0

PART NUMBER, PRICE, UPC

Product Packages	Media	Part Number	UPC Code	US List Price
VisualAge C++ v3.0				
VisualAge C++ v3.0	3.5" + Docs	30H1664	0-87944-13076-5	$525.00
VisualAge C++ v3.0	CD-ROM only	30H1665	0-87944-13077-2	$449.00
VisualAge C++ v3.0	CD-ROM w/ Docs	30H1666	0-87944-13078-9	$489.00
Program Upgrades (From C Set ++ V2.0 and V2.1)				
VisualAge C++ v3.0	3.5" + Docs	30H1681	0-87944-13081-9	$299.00
VisualAge C++ v3.0	CD-ROM only	30H1682	0-87944-13082-6	$225.00
VisualAge C++ v3.0	CD-ROM w/ Docs	30H1683	0-87944-13083-3	$265.00
Promotional Upgrade Offer* (Competitive Upgrade and First Step Upgrade)				
VisualAge C++ v3.0	3.5" + Docs	30H1775	0-87944-13084-0	$423.00
VisualAge C++ v3.0	CD-ROM only	30H1776	0-87944-13085-7	$349.00
VisualAge C++ v3.0	CD-ROM w/ Docs	30H1777	0-87944-13086-4	$389.00
Additional License(s)				
VisualAge C++ v3.0	(Single)	30H1675		$429.00
VisualAge C++ v3.0	(Qty 5)	30H1676		$2,039.00
VisualAge C++ v3.0	(Qty 10)	30H1677		$3,864.00
VisualAge C++ v3.0	(Qty 50)	30H1678		$18,031.00
Publications Program Packages				
VisualAge C++ v3.0	Standard Ref. Man.	30H1679	0-87944-13087-1	$56.00
VisualAge C++ v3.0	Extended Ref. Man.	30H1680	0-87944-13088-8	$249.00
IBM Experience C++ (A multi-media tutorial)				
IBM Experience C++	CD-ROM only	03H4441	0-87944-13447-3	$135.00

Note*: Promotional Upgrade Offer in effect until October 27, 1995. For more information about this offer, please refer to the Campaign Information section.

6/95

HIGHLY CONFIDENTIAL

VisualAge C++ Version 3.0

TECHNICAL SUPPORT

Dealer Support

Support is available for our dealers by calling 1-800-IBM-PROD.

Product Support

Please have your customers return their Registration Card enclosed in the VisualAge C++ packaging to receive their customer number.

If you or your customers have problems or questions:

- Check Appendix D of IBM C/C++ Tools: Programming Guide. The solutions to common problems and questions are documented in this appendix.
- Review VisualAge C++'s README file.

CompuServe

CompuServe users have 24-hour access to IBM OS/2 Developers' Forums.

- Enter GO OS2DF1, then post questions to appropriate sections of the forum.
- To register, call 1-800-524-3388 #239

Internet

- Workframe/2 questions: workframe@vnet.ibm.com
- Other VisualAge C++ questions: cset2@vnet.ibm.com

Software Defect Support Line

To report software defects, have customer number ready and call 1-800-237-5511 (24 hours a day, 7 days a week).

6/95

MSC 00545191

HIGHLY CONFIDENTIAL

MSC0000545191

VisualAge C++ Version 3.0

TECHNICAL SUPPORT, cont'd.

TalkLink

- Carry on electronic conversations with the worldwide IBM community or utilize any of the following services via IBMLink™:

- To register and begin using TalkLink, call 1-800-547-1283.

Q & A Bulletin Boards

- Over 52 forums with the latest technical info

- Exchange questions and answers with thousands of IBM customers, 24 hours a day

IBM Support

- OS/2 Support info

- Submit/view problem reports and IBM responses

- Order OS/2 Corrective Service Disks (CSDs)

OS/2 Software Library

- View/choose fixes

- Download service packs and CSDs

News & Announcements

- Up-to-the-minute product/service info

- Latest info on Beta software and CSDs

HIGHLY
CONFIDENTIAL

VisualAge C++ Version 3.0

BENEFITS/KEY MESSAGES

Benefits

Feature	Benefit
Visual Application Construction from Parts	• Developers can assemble applications visually using pre-defined parts, giving them: – Rapid development with minimal coding – Dramatically reduced training times – The ability to use less skilled developers for parts assembly, and more skilled developers for coding parts
Object-oriented Access to Relational Data	• Existing relational data can be accessed from the visual builder with very little coding • Applications accessing data can be assembled visually
IBM Open Class Libraries	• Applications can be easily ported to other platforms • Developers are shielded from complex system interfaces • Complex applications can be developed very quickly
Direct-to-SOM	• SOM Objects can be created using C++ syntax, no need to learn other languages (IDL) or tools • Allows developers to create distributed object applications more easily • SOM enables the creation of objects that can be shared by several languages
New Development Environment	• Developers can move quickly and easily between tools • Environment is simple and easy to learn • Projects can be set up automatically
New Browser	• Users can browse uncompiled code • Browser is fast and easy to find
Pre-compiled Header Files	• Allows much faster compiling
New 32-bit Linker	• Linkage is 2-3 times faster than with V2.1

6/95

MSC 00545193

HIGHLY CONFIDENTIAL

MSC0000545193

VisualAge C++ Version 3.0

BENEFITS/KEY MESSAGES, cont'd.

Key Messages

VisualAge C++ Version 3.0

The new generation of C++ has arrived with the introduction of visual application construction from parts. Developer productivity is enhanced without losing the power and scalability of C++.

IBM's C++ and Open Class are designed to provide a powerful multi-platform environment that scales from the desktop to the mainframe including OS/2 Warp, AIX, Sun Solaris, MVS, and OS/400, and in the future: Windows NT, Windows 95, and others.

VisualAge C++ extends the power of IBM's VisualAge family of products to include both Smalltalk and now C++. Together these languages account for more than 90% of object-oriented development!

Experience C++

- Low cost learning tool
- Ease of use
- Multimedia is state of the art

HIGHLY CONFIDENTIAL