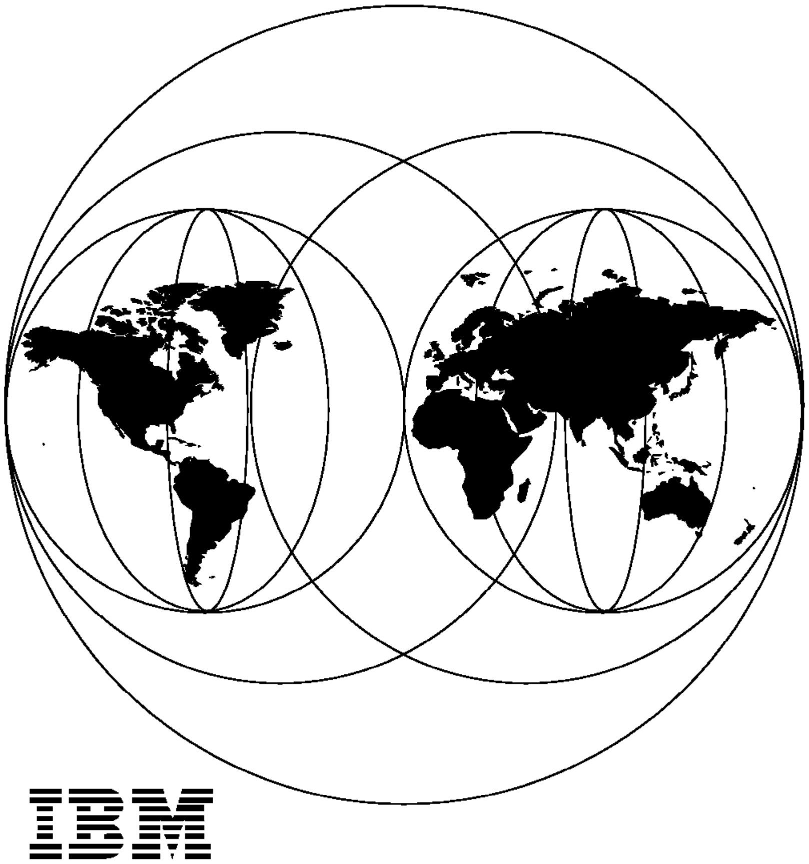
International Technical Support Organization

GG24-4177-00

AIX Application Development and How To Migrate and Enhance Your Legacy Applications

May 1995



International Technical Support Organization
San Jose Center

International Technical Support Organization

GG24-4177-00

AIX Application Development and How To Migrate and Enhance Your Legacy Applications

May 1995

Take Note!

Before using this information and the product it supports, be sure to read the general information under “Special Notices” on page xvii.

First Edition (May 1995)

This edition applies to Version 1.1.1 and 1.2.0 of AIXwindows Interface Composer/6000, 5756-027, Version 2.1.0 of Configuration Management Version Control/6000, 5765-207, Version 1.1.0 of DATABASE 2/6000, 5765-172, Version 3.1.3 of Micro Focus COBOL, Version 6.0.36.3.2 of ORACLE Relational Database Management System, Version 1.2.2 of Software Development Environment Workbench/6000, 5696-037, and Version 1.2.0 of Software Development Environment Integrator/6000, 5696-137 for use with the AIX Operating System 3.2.4.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for readers' feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Center
Dept. 471, Building Building 070B
5600 Cottle Road
San Jose, California 95193-0001

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Abstract

This document describes the downsizing of a DATABASE 2 business application from an IBM mainframe running MVS to the RISC System/6000 and AIX/6000. The project included a minimal effort migration of the COBOL application and its database to AIX, replacement of its ISPF panels with a graphical user interface using AIXwindows Interface Composer/6000 (AIC/6000), maintenance and testing of both AIX and MVS versions of the application on AIX, and reimplementation of the application in C^{++} on AIX. Software baselines for all AIX and MVS versions of the application were controlled on AIX. The purpose of the project was to evaluate and exercise the several application development products available for the AIX platform in the context of a realistic application development effort. These products were SDE WorkBench/6000, SDE Integrator/6000 and several AIX application development products which are integrated with SDE WorkBench/6000 including: AIXwindows Interface Composer/6000 (AIC/6000), Configuration Management Version Control/6000 (CMVC/6000), XL C, XL C^{++}, Micro Focus COBOL and Micro Focus COBOL Toolbox for AIX. This volume focuses on the programming issues involved in the migration and modernization of the application, and on the use and tailoring of the AIX products supporting the edit, compile, and debug activities.

AD AX (266 pages)

Contents

Figures

Tables

Special Notices

AIXwindows Interface Composer/6000	AIX
AIX/6000	AIXwindows
Configuration Management Version Control/6000 (CMVC/6000)	COBOL/370
Common User Access	CUA
DATABASE 2	DB2
DB2/2	DB2/6000

Distributed Relational Database Architecture	DRDA
IBM	OS/2
OS/400	POWER Architecture
PowerPC	POWERserver
POWERstation	Presentation Manager
RISC System/6000	SAA
Scalable POWERparallel Systems	SDE WorkBench/6000
SDE Integrator/6000	Systems Application Architecture
VS COBOL II	SP1

The following terms, which are denoted by a double asterisk (**) in this publication, are trademarks of other companies:

AdaWorld	Alsys
ANSI	American National Standards Institute
AT&T	American Telephone and Telegraph Company
NetLS	Apollo Computer, Inc., a subsidiary of Hewlett-Packard Co.
Network Licensing System	Apollo Computer, Inc., a subsidiary of Hewlett-Packard Co.
NCS	Apollo Computer, Inc., a subsidiary of Hewlett-Packard Co.
Network Computing System	Apollo Computer, Inc., a subsidiary of Hewlett-Packard Co.
TRITON Tools	Baan International B.V.
SEDIT	BENAROYA
Yellow Pages	British Telecommunications, PLC
SEDIT	BENAROYA
Prolog by BIM	BIM
Teamwork	Cadre
Process WEAVER	CAP Gemini Sogeti (CGS)
ProMod-PLUS	CAP debis GEI
CaseWare/CM	CaseWare
Software Backplane	CRI (Atherton)
DECADE	Delaware Computing
APPLIDUAL	DUAL
ENFIN/3	Easel
FrameMaker	Frame Technology
GNU	Free Software Foundation
Hewlett-Packard	Hewlett-Packard Company
HP	Hewlett-Packard Company
HP-UX	Hewlett-Packard Company
SoftBench	Hewlett-Packard Company
INFORMIX	Informix Software, Inc.
INFORMIX/4GL for ToolBus	Informix
StP-ISE	IDE
IEEE	Institute of Electrical and Electronics Engineers
POSIX	Institute of Electrical and Electronics Engineers
Intel	Intel Corporation
Pentium	Intel Corporation
Interleaf 5	Interleaf
ISO	International Organization for Standardization
Internet	Internet, Inc.
PVCS CB and VM	INTERSOLV
PVCS Version Manager	INTERSOLV, Inc.

KeyOne	LPS
X Window System	Massachusetts Institute of Technology
MIT	Massachusetts Institute of Technology
XRunner	Mercury Interactive Corp., Israel
Micro Focus Dialog System	Micro Focus
COBOL, COBOL with Toolbox	Micro Focus
Micro Focus	Micro Focus Limited
Micro Focus COBOL	Micro Focus Limited
Micro Focus COBOL Toolbox	Micro Focus Limited
Micro Focus Run Time Environment, RTE	Micro Focus Limited
RTE, Run Time Environment	Micro Focus Limited
Operating System Extensions	Micro Focus Limited
Micro Focus Animator	Micro Focus Limited
MS-DOS	Microsoft Corporation
Microsoft	Microsoft Corporation
Microsoft Windows	Microsoft Corporation
Innovator	MID
Motorola	Motorola, Inc.
Open Interface	Neuron Data
OBJECTORY	Objective Systems (OS)
ORACLE	Oracle Corporation, Inc
Motif	Open Software Foundation, Inc.
OSF	Open Software Foundation, Inc.
OSF/Motif	Open Software Foundation, Inc.
OSF/DCE	Open Software Foundation, Inc.
HyperWork	PBS
SMARTsystem	PROCASE
ROSE	Rational
REFINE/FORTRAN	Reasoning Systems
Objecteering	Softeam
Software TestWorks (STW)	Software Research
Sun	Sun Microsystems Incorporated
SunOS	Sun Microsystems Incorporated
SPARCstation	Sun Microsystems Incorporated
SPARCserver	Sun Microsystems Incorporated
Solaris	Sun Microsystems Incorporated
NFS	Sun Microsystems Incorporated
Network File System	Sun Microsystems Incorporated
SYBASE	Sybase, Inc.
Systemator	Sysdeco
SDT	TeleLOGIC
Emacs editors	Unipress Software
ASE/ASA, AGE/GEODE, LOGISCOPE	VERILOG
VADS	Verdix
ViSTA	VERITAS Software
View, Vutil	Versant
VIEWS/VSF	Virtual Software Factory
UIM/X	Visual Edge Limited
I-CASE	Westmount
Uniface WB	UnifAce
UNIX	X/Open Company Limited
X/Open	X/Open Company Limited

Preface

This book should be read by application developers, their managers, and software configuration managers. It describes a project that downsized and modernized a COBOL DATABASE 2 business application from MVS to AIX as a way of exploring the use and advantages of several AIX application development products. By supporting and automating many aspects of software development on the RISC System/6000, these products enhance the productivity of UNIX programmers and ease the transition to UNIX for mainframe application developers.

This book looks closely at IBM's SDE WorkBench/6000 and the set of integrated developer tools that are bundled with SDE WorkBench/6000. It also examines the use of optional integrated AIX application development tools, such as: IBM's AIXwindows Interface Composer/6000; an OSF/Motif** compliant graphical user interface "builder"; IBM's Configuration Management Version Control/6000; and SDE Integrator/6000, a product enabling the integration of locally developed utilities with SDE WorkBench/6000.

This book looks at several technical issues related to migrating a COBOL application and its DB2 database from MVS to AIX. It shows how AIC is used to generate GUI code, and how that code is linked with application code written in COBOL, C, and C^{++}. This book also looks at how a COBOL legacy application, which accesses DB2, can rewritten in C^{++} for DB2/6000.

A companion volume, *Did You Say, CMVC?*, GG24-4178, tells how to plan for and use CMVC for application development and also describes how CMVC was used to support this specific project.

Though it is almost two years since the first lines of this book were written, it still holds a lot of valuable information and points to interesting techniques of *how to* migrate and reengineer legacy applications. We, the authors of this book, hope you enjoy it and find it valuable in your efforts to capitalize and protect your company's investments.

[illegible]

Leif Trulsson
ITSO - San Jose, California
May 1995.

- Better late than never -

How This Book Is Organized

Readers should focus on the sections that are most relevant to them and skip those that are not. There are two themes to this book. The first is the use and benefits of the application development products and the environment they create, and the second theme is the migration and modernization of a COBOL DB2 legacy application to AIX.

The reader should note that some chapters include details of the first theme interleaved with details of the second. Thus, the reader who is primarily interested in the programming issues for COBOL or C^{++} may want to skim lightly over the details of how to use the various SDE WorkBench/6000 windows and menus. Likewise, the reader who wants to get a feel for how developers with diverse backgrounds, using unique development tools, and programming in different languages worked in the same application development environment, may not want to get bogged down in the details of how an ISPF character-based user interface is approximated with a GUI.

The book is organized into chapters that help the reader identify specific areas of particular interest. The three core chapters describing the sequence of programming activities are organized into subsections focusing on design, implementation, integration, and tool usage.

- Chapter 1, “Why AIX Application Development Products?”

  This chapter looks at why open systems application development products are needed in the 1990s and discusses the types and characteristics of those products. It is for managers and can be skipped by any reader familiar with these concepts.

- Chapter 2, “Introducing AIX Application Development Products and DB2/6000”

  This chapter provides an overview of the AIX application development products used on this project and describes DATABASE 2/6000 and its related products. It can be skipped by any reader familiar with the names and purposes of the products.

- Chapter 3, “The Legacy Application”

  This chapter describes the legacy application and discusses the strategy we chose to employ in its migration. It is a prerequisite to understanding the next four chapters.

- Chapter 4, “Defining the Project and Setting Its Goals”

  This chapter describes the engineering goals, constraints, and decisions governing the downsizing project itself. It should be of interest to both managers and developers.

- Chapter 5, “Our Application Development Environment”

  This chapter describes the hardware, software, and file system topologies created to support and enable the AIX application development environment and tool configuration used in this project. It can be of more interest to someone with a system or network administration focus, and is a prerequisite to understanding the distributed development aspects of the AIX application development environment, which is discussed in the next three chapters.

- Chapter 6, “Migrating the Legacy Application to AIX”

This chapter describes the application's migration to AIX, generation of its ISPF-like GUI, and changes made to the MVS COBOL code that enabled code running on both platforms to share a common subset of the COBOL code. It should be of interest to developers focusing on COBOL, DB2/6000, C, and GUI issues.

- Chapter 7, “Improving the Application's GUI”

  This chapter details the improvement of the GUI according to CUA guidelines. It should be of particular interest to C and GUI programmers.

- Chapter 8, “Object-Orientation of Our Legacy Application”

  This chapter describes the reimplementation of the legacy application and the regeneration of the improved GUI in C^{++}. It should be most useful to C^{++} developers.

- Chapter 10, “Integrating User-Developed Utilities into SDE WorkBench/6000”

  This chapter describes integrating a user-developed utility with SDE WorkBench/6000 using SDE Integrator/6000. Developers who are creating or supporting the AIX application development environment should find this chapter useful.

- Chapter 9, “Tailoring SDE WorkBench/6000, Integrated Tools, and DB2/6000”

  This chapter details the specific AIX application development environment created for this project's use, including how products were installed, configured, and tailored for the specific languages, purposes, and requirements of this project. System administrators should find this chapter useful.

Prerequisite Publications

The following publications would be needed to install, customize or utilize the products described in this book:

AIX Interface Composer Publications

- *Installing and Configuring AIXwindows Interface Composer, Version 1.2*, SC23-2570
- *Getting Started with AIXwindows Interface Composer*, SC23-2557
- *AIXwindows Interface Composer Developer's Guide, Version 1.2*, SC23-2558
- *Extending and Customizing AIXwindows Interface Composer*, SC23-2559
- *User Interface Programming Concepts: AIXwindows Interface Composer, Volume 2*, SC23-2405

CMVC Publications

- *Configuration Management Version Control Server Administration and Installation, Version 2 Release 2*, SC09-1631
- *Configuration Management Version Control Concepts, Version 2 Release 1*, SC09-1633
- *Configuration Management Version Control User's Guide, Version 1 Release 2*, SC09-1634

- *Configuration Management Version Control Commands Reference, Version 2 Release 2*, SC09-1635
- *Configuration Management Version Control Client Installation and Configuration, Version 2 Release 1*, SC09-1596
- *Configuration Management Version Control User's Reference, Version 2 Release 2*, SC09-1597

C^{++} and Object-Oriented Analysis, Design, and Programming Publications

- *IBM AIX XL C^{++} Compiler/6000 User's Guide, Version 1.1.1*, SC09-1472
- *Object-Oriented Analysis and Design with Applications*, SR28-5338, by Grady Booch, published by Benjamin/Cummings Publishing Company, Inc., Redwood City, CA (available from IBM or at book stores),
- *The C^{++} Programming Language, Second Edition*, SR28-5340, by Bjarne Stroustrup, published by Addison-Wesley Publishing Company, Reading, MA (available from IBM or at bookstores),
- *The Annotated C^{++} Reference Manual*, SR28-5342, by Bjarne Stroustrup and Margaret A. Ellis, published by Addison-Wesley Publishing Company, Reading, MA (available from IBM or at bookstores),
- *Object-Oriented Programming Using C^{++}*, SR28-5339, by Ira Pohl, published by Benjamin/Cummings Publishing Company, Inc., Redwood City, CA (available from IBM and at bookstores),
- *Object-Oriented Programming*, by Peter Coad and Jill Nicola, published by PTR Prentice Hall, Inc., Englewood Cliffs, NJ,

DATABASE 2/6000 Publications

- *IBM DATABASE 2 AIX/6000 Call Level Interface Guide and Reference*, SC09-1626
- *IBM DATABASE 2 AIX/6000 Information and Planning Guide*, GC09-1569

NetLS and NCS Publications

- *NetLS Quick Start Guide*, SC09-1661
- *Managing Software Products with the Network License System*, SC09-1660
- *Managing NCS Software*, SC09-1834

SDE WorkBench/6000 and SDE Integrator/6000 Publications

- *IBM AIX SDE WorkBench/6000 User's Guide and Reference, Version 1.2*, SC09-1453
- *Installing IBM AIX SDE WorkBench/6000 and SDE Integrator/6000*, GC09-1452
- *IBM AIX SDE WorkBench/6000 Development Manager: Managing Files and Directories, Version 1.2*, SC09-1455
- *IBM AIX SDE WorkBench/6000 Program Editor, Version 1.2*, SC09-1456
- *IBM AIX SDE WorkBench/6000 Static Analyzer, Version 1.2*, SC09-1457

- *IBM AIX SDE WorkBench/6000 Program Debugger, Version 1.2*, SC09-1458
- *IBM AIX SDE WorkBench/6000 Program Builder: Managing Program Construction, Version 1.2*, SC09-1459
- *IBM AIX SDE WorkBench/6000 General Tools, Version 1.2*, SC09-1460
- *IBM AIX SDE Integrator/6000 Portability Guide, Version 1.2*, SC09-1461
- *IBM AIX SDE Integrator/6000 Programmer's Guide, Version 1.2*, SC09-1462
- *IBM AIX SDE Integrator/6000 Programmer's Reference, Version 1.2*, SC09-1463
- *IBM AIX SDE Integrator/6000 Distributing and Encapsulation, Version 1.2*, SC09-1464

Related Product Publications

The following publications should be valuable if the reader uses products detailed in this document.

ORACLE Publications

- *ORACLE RDBMS Database Administrator's Guide, Version 6.0*,3601-v6.0 1090
- *ORACLE for IBM RISC System/6000 Installation and User's Guide Version 6.0*, 5687-60-0792

International Technical Support Organization Publications

A complete list of International Technical Support Organization publications, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070

To get a catalog of ITSO technical publications (known as “redbooks”), VNET users may type:

```
TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
```

How to Order ITSO Redbooks

IBM employees in the USA may order ITSO books and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-284-4721. Visa and Master Cards are accepted. Outside the USA, customers should contact their local IBM office.

Customers may order hardcopy ITSO books individually or in customized sets, called GBOFs, which relate to specific functions of interest. IBM employees and customers may also order ITSO books in online format on CD-ROM collections, which contain redbooks on a variety of products.

Acknowledgments

This document is the result of a residency project conducted at the International Technical Support Organization, San Jose Center, during 1993.

The advisor for this project was:

Lorna R. Conas,
International Technical Support Organization,
San Jose Center, California, USA

The authors of this document are:

Richard Kortmann, Advisory Programmer, IBM Germany Development GmbH, Hanover, Germany

Branko Peteh, AIX Development Support, IBM Bergen Environmental Center, Bergen, Norway

Leif Trulsson, AIX Consultant Systems Engineer,
IBM Services South, Malmö, Sweden

Thanks to the following people for the invaluable advice and guidance provided in the production of this document:

Kalman Bohus, Toronto Labs, Toronto, Canada

Joe DeCarlo, International Technical Support Organization,
San Jose Center, USA

Laura Nystrom, Editor
San Jose, USA

Marvin Harrison, Toronto Labs, Toronto, Canada

Sam Lahire, Toronto Labs, Toronto, Canada

Sarah McNamara, Toronto Labs, Toronto, Canada

Bob Minns, International Technical Support Organization,
Austin Center, USA

Shankar Sanyal, Toronto Labs, Toronto, Canada

Leigh Williamson, Austin, USA

Thanks to the following people whose provision of technical support and products enabled the success of this project:

David Satterwhite, ORACLE Corp., Redwood Shores, CA

Robert Garnero, CASEbridge Corp., Sunnyvale, CA

Ken Ausich, CASEbridge Corp., Sunnyvale, CA

Steven Kornberg, Micro Focus Limited, Palo Alto, CA

Part 1. AIX Application Development

Chapter 1. Why AIX Application Development Products?

Increasingly, companies are looking to open systems platforms to help meet the cost, schedule, and quality challenges they face in application development (AD). For some, migrating to the IBM* RISC System/6000* as a first open systems platform is a particularly attractive choice because of the Advanced Interactive Executive (AIX*) operating system's affinity for the IBM proprietary systems on which they currently execute and develop their applications. Using IBM and vendor application development products to create an application development environment can ease the transition to open systems and enhance programmer productivity on those platforms if you know what qualities and functionality to look for in those products.

1.1 Who Is Migrating to Open Systems and UNIX?

Open systems provide increased flexibility, lowered operating system and hardware costs, and greater interconnectivity options than older proprietary environments offer. Applications are migrating to open systems from a variety of platforms for several reasons that are discussed in this section.

1.1.1 Downsizers: Mainframe Developers

Many application developers want to migrate their business applications from proprietary operating systems and mainframe computers to workstation and midrange computers running operating systems based on UNIX** because they are much less expensive to purchase and maintain. These developers may also be looking for greater flexibility in their vendor relationships or a wider set of hardware and software options. They may want to bring applications and the databases that support them closer to the end users by creating a distributed, network-based computing environment. This migration from mainframes to open systems is often referred to as *downsizing*.

Developers interested in downsizing usually do not want to downsize all of their applications at once. Typically these developers want newly migrated applications to interoperate, or coexist, with their mainframe applications for an indefinite period of time as they explore the advantages and potential disadvantages of the new computing environment they are creating. Often these developers have limited in-house open systems experience and are looking to create a development environment that is friendly to the UNIX novice.

1.1.2 Upsizers: Desktop Developers

Another group of developers is migrating to open systems from a different direction. This group is faced with the requirement to upgrade their applications from networked workstations running MS-DOS**, Microsoft Windows**, or OS/2*, onto larger capacity mid-range computers or multiuser operating systems. These developers outgrew their original platforms but want to continue to work them into their new configurations. These developers are also building distributed applications often characterized by client/server architectures and database dependencies. Developers on these platforms probably worked with the C programming language, and with desktop versions of software development tools, utilities, and libraries that originated on UNIX. They make an easy transition to AIX. Because they are used to graphical user interfaces (GUIs) and point-and-click interaction with their tools, however, they may not be as quick to adapt to the traditional command-line interface epitomized by the UNIX “shells.”

C^{++} is a widely used object-oriented language. Compilers, object-oriented databases, and many of the development tools that exist for C^{++} are supported by vendors on UNIX platforms. Many object-oriented downsizers are migrating to UNIX-based open systems because of this. The variety of application development products designed specifically for object-oriented development are still limited, because this is emerging technology. However, these developers are not only looking to create an application development environment that supports their newer object-oriented development efforts. They also need support in their AD environment and products for their continuing traditional programming projects.

1.2 Why Are They Migrating to AIX?

Several factors contribute to the business application developers' desire to migrate their legacy applications to the RISC System/6000. It is appealing as an open systems platform for developers needing to move either down the price scale or up the performance and capacity scale. Since its introduction, the product family is growing at both ends of the price performance range. High-end performance doubles every 12 to 18 months while prices undercut the competition at each new product announcement.

1.2.1 Superior Price Performance

IBM defines the acronym *RISC* as *Reduced Instruction Set Computer*. The original RISC System/6000 computers were based on the Performance Optimized with Enhanced RISC (POWER) Architecture* which is sometimes referred as “RIOS 1.” This implementation was enhanced in 1993 when IBM announced computers based on the second-generation POWER Architecture chip set, sometimes called “RIOS 2.” The RIOS 1 implementation of the POWER Architecture was super-scalar. It used parallel floating-point, integer, and instruction dispatch units which, along with zero-cycle branching and a multiply-add instruction, made it possible to have a peak execution rate of five operations per clock cycle. By doubling the number of floating-point and integer units in the chip complex, the RIOS 2 computers doubled the number of floating-point and integer operations that could be performed in a single clock cycle. Thus, RISC System/6000 computers built with the RIOS 2 chip set have double the integer and floating point performance of comparable RIOS 1 models using approximately the same clock speed.

The PowerPC* is a somewhat simplified single chip implementation of the POWER Architecture. It is manufactured by IBM and Motorola**. The PowerPC is the basis of computer lines being produced by IBM, Apple, and several other companies. This chip has two to three times greater price performance than traditional PC chip architectures, such as the Intel** X86 family. There will be several PowerPC chips targeted for laptop, desktop and multiprocessor uses; the 601 is just the first to be announced. The 601's performance was equal to or better than Intel's Pentium** at less than half the cost when first announced. Parallel shared-memory and symmetrical multiprocessor (SMP) computers built on either the PowerPC or the RIOS chip sets present the opportunity to increase this performance tremendously. IBM and others are pursuing several initiatives along these lines.

The RISC System/6000 family of POWERservers* and POWERstations* price performance range extends from stand-alone desktops and small local area network (LAN) servers. POWERserver models range from specialized single-user scientific or technical workstations to rack-mounted, thousands-of-transactions-per-second, multiuser commercial processors. The PowerPC complements these models with high-volume, very low-end laptops and desktops, while the Scalable POWERparallel Systems* (SP1* and SP2) offers a scalable, parallel, general purpose supercomputer at the very high end. Every price performance point on the range in between these two extremes is represented in this binary compatible family, including the RISC/6000 X station. The X station a cost-effective network-attached X server. The RISC System/6000 family includes high-availability configurations, high-capacity, high-performance Network File System** (NFS**) data servers, and specialized niche products, such as network routers and gateways. RISC System/6000 computers support high speed connectivity options to IBM mainframes and the full variety of disk, input/output, and communication devices typically associated with open systems computers.

1.2.2 AIX and Open Systems Software

AIX is the ideal open systems platform from the portability point of view. AIX implements not just one flavor of UNIX, but also the two original generic versions (AT&T** and BSD), plus the emerging open systems standard interfaces such as IEEE* POSIX** and OSF/AES**. IBM has made a public commitment that AIX will continue to conform to every relevant industry standard that promotes the open systems environment:

- AIX conforms to *de facto* industry standards, such as: TCP/IP, ATT SVID Issue 2, and BSD 4.3.
- AIX conforms to *de jure* standards, such as POSIX 1003.1-2, ISO**/IEC 9945-1 and ISO/IEC 9945-2, and FIPS 1541-1.
- IBM markets AIX products that are based on specifications defined by open systems industry consortia, such as the OSF/Motif** window manager and the OSF/Distributed Computing Environment (OSF/DCE**).

IBM is active with standards promoting consortia and provides AIX technology to such organizations as the Open Software Foundation (OSF**), X/Open**, and the Common Open Software Environment (COSE) group. In the language arena, AIX also supports standards strongly, having implemented the current existing and proposed American National Standards Institute (ANSI**) standards in all the compilers.

All major open systems software vendors support their products on AIX. IBM remarkets and sells many best-of-breed open systems applications on AIX under its own logos. All major UNIX relational database vendors support their products on the RISC System/6000. IBM is offering its own AIX software solutions, such as CMVC and DB2/6000 on AIX and on other vendors' open systems platforms.

1.2.3 DATABASE 2/6000 and Mainframe Affinity

Whether downsizing or upgrading, developers leaving proprietary platforms face the special challenge of how to facilitate the maximum reuse of existing software, design, or key application programming interfaces (APIs) during their migration to the open systems platform. Salvaging as much as possible of the existing application is advisable not only because of development cost savings but also because of the expense of retraining technical and end user staff. IBM's RISC System/6000 is an attractive target platform for open systems migration with the appearance of many products and APIs on it that were formerly available only on IBM mainframes.

The recent availability of DATABASE 2/6000 (DB2/6000*), for instance, supports the migration of DATABASE 2* (DB2*) applications to the RISC System/6000 from IBM operating systems, such as MVS, VSE, OS/400*, and OS/2. DB2/6000 enables the existing DB2 database design and code to remain largely intact as the application is migrated to AIX. Because of AIX's strong support for IBM's Systems Network Architecture (SNA), it offers several options for distribution and connectivity of applications with IBM proprietary platforms as well.

1.2.4 MS-DOS Affinity

Developers moving from desktop environments may want some MS-DOS affinity on their UNIX platform. MS-DOS simulators that execute programmers' favorite MS-DOS applications on the AIX workstation are available. There are also products that enable X Window System** and ASCII access to AIX from remote MS-DOS and Microsoft Windows workstations on the LAN. In addition, there are UNIX versions of many popular MS-DOS and Microsoft Windows applications. AIX communications software, such as Portable Netware/6000, enables AIX to participate on PC LANs.

1.2.5 AIX Application Development Products

Another reason many developers are choosing AIX as their first open systems platform is the array of application development (AD) products available for AIX. These AIX AD products ensure that AIX is a fine development platform for applications targeted to run on IBM proprietary platforms. AIX is also an excellent development platform for applications targeted to run on multiple open systems platforms. Many of these AIX AD products interoperate with appropriate counterpart products executing on other UNIX platforms. Developers can install a wide variety of language- and technology-specific tools on AIX. With AIX AD products, developers can create a consistent, integrated, cross-UNIX open systems application development environment.

1.3 What Should Open Systems Application Development Products Provide?

Open systems offer many advantages to the business application developer, but they also introduce complexity and diversity into the application development environment. To meet the needs of application developers, open systems AD products must improve programmer productivity, automate repetitive procedures, enforce consistency in processes associated with software development, and improve the quality of the software produced.

1.3.1 The Application Development Framework

The most fundamental requirement is to provide a development environment or an AD framework that enables programmers to become rapidly productive with minimum formal training. A framework provides a consistent environment for both programmers and the AD tools they use, and facilitates the movement of engineering data between the tools that generate and manipulate this data.

The AD framework can provide a significant advantage if it presents a consistent user interface for the various tools it contains. Users spend more time productively using their tools if the conventions used to enter and display data are well-known and predictable. GUIs that use point-and-click window-based mechanics are particularly well-suited for rapid mastery by the novice. Because there is always a range of skills and experience in the technical staff employed on any application development effort, the user interface must have mechanisms appropriate for both the casual or novice user and for the experienced or specialized user.

An AD framework should be rich in fundamental tools yet permit the integration of additional tools procured from a wide variety of sources to meet the unique needs of any given development effort. Just as the user interface to tools should be consistent, the tools should have a consistent interface with each other indirectly, through the framework. This framework should provide uniform mechanisms to start, stop, and control execution of all tools.

The AD environment must be immediately usable with a minimal site-specific configuration, yet be sophisticated enough enable development organizations to tailor it on a network, system, or individual user basis. This configurability should support a wide range of software engineering process models and practices. The framework should present essentially the same model to users and AD tools whether it is installed stand-alone on a single machine or distributed across several systems in a network.

1.3.2 Application Development Tools for Edit, Compile, and Debug

An AD environment must provide a complete set of tools to fully support the edit, compile, and debug cycle. It must provide for a number of high performance compilers, as well as other tools, such as a link editor and loader. This environment must also support the exercise of the code in its various states, providing tools such as:

- Debuggers
- Interpreters
- Static analyzers
- Graphical browsers.

The AD environment must support development in several languages equally well, because the programmer who works in COBOL today will be working with C or C^{++} tomorrow, and will work with several other programming languages, such as database fourth-generation languages (4GLs), along the way. As an application development organization migrates and modernizes its applications, it supports development in several languages simultaneously. The development tools must therefore support multiple languages, recognizing that the programmer's needs vary with the language. Editors should, therefore, be optimized for use with specific languages, as well as for general-purpose editing. However, all programmers ought to be able to use the same AD tools and environment for fundamental programming tasks, regardless of their language focus.

1.3.3 Application Development Tools for Software Engineering

The AD environment must support the integration of tools that automate the discipline and science of engineering software. Such tools are critical for software development on a large scale or for the development of mission-critical or commercial software on any scale. The AD environment should support tools to facilitate:

- Requirements analysis

- System and software design
- Rapid prototyping
- Code translation
- Code generation
- Formal test and verification
- Reverse engineering and reengineering.

1.3.4 Full Life Cycle Support

No AD environment would be sufficient without tools that support the full life cycle of application development. These tools are typically used by all members of the development team: technical staff, support staff, and even project management. The AD environment must support integration of tools for:

- Configuration management
- Version control
- Requirements traceability
- Documentation generation and publication support
- Process management
- Project management.

1.3.5 Cross-Platform Support and Commonality

Tools in the AD environment must provide the same benefits across platforms, to support development activities on multiple open systems platforms. The tools must also support development of code that runs under multiple operating systems on hardware platforms from a number of vendors. Another aspect of this requirement is that these tools must be accessible to some extent from workstations that are not UNIX-based or are lower-cost alternatives to a full-function UNIX desktop computer. Of more importance is that the developers' interaction with their tools be consistent across platforms.

1.3.6 ISO 9000 and Application Development

No application development organization in the 1990s can afford to ignore the importance of software quality as a factor in software development and maintenance cost management. Companies of all sizes and their customers are increasingly focused on the quality management guidelines that are presented in the International Standard Organization (ISO) 9000 Series of Quality Standards. ISO 9000 certification is a business necessity in Europe, and will become equally necessary in North America and the Asia Pacific region in the near future.

IS0 9001, ISO 9002, and ISO 9003 are of particular importance to companies in the business of providing software or services. ISO 9001 provides a comprehensive set of requirements for a software quality system where a contractual agreement between two parties must demonstrate the supplier's capability to design and supply a product or service. Recognizing the peculiarities of the software industry, ISO 9000-3 provides guidelines for the application of ISO 9001 to the development, distribution, and maintenance of software.

Several of the guidelines set forth in ISO 9000-3 can be met by the appropriate application of automated configuration management tools and many elements of ISO 9001 depend on configuration management mechanisms. The ISO 9001 topics, which depend to some degree or wholly on configuration management include:

- Document control
- Design control
- Product identification and traceability
- Inspection and test status
- Control of nonconforming product
- Internal quality audits.

While many companies that perform application development may not market software products, they face the same issues as companies that do. Business application developers can benefit equally from applying the ISO 9000-3 principles to their application development process; therefore, an AD environment should provide automated configuration management, and the better its mechanisms are integrated with all processes the more likely they are to be successfully applied.

1.3.7 Software Engineering Institute Capability Maturity Model

The Software Engineering Institute (SEI) also addresses the problem of software quality through the process by which the software is developed and the degree to which that process is overtly managed by the organization. SEI defines a classification of software engineering process maturity levels that are used to characterize a software development organization. Organizations can be formally assessed or perform a self-assessment to find which level they have achieved. The higher the level an organization achieves, the more likely it is to produce a better quality of software. In addition, it can expect to achieve greater software development and maintenance cost efficiencies. These levels are defined as:

- Initial
- Repeatable
- Defined
- Managed
- Optimized.

SEI ratings are increasingly important to providers of software to United States Government customers but are designed to be relevant to any organization that develops and maintains software. Business application developers should find that they can also benefit from increasing their process management from one maturity level to the next.

As an organization attempts to bring itself up to each new level of process maturity, it should find automated software configuration management and change control mechanisms to be invaluable. Business application developers should ensure that automated software configuration management and change control mechanisms can be thoroughly integrated into any AD environment they choose to implement.

1.3.8 Open Systems Standards and Application Development Products

When an organization considers an AD environment and its tools, it should also consider the relationship between those products and any relevant industry, government, or *de facto* standards. While some AD solutions are entirely self-contained, they can also be proprietary and may not conform to open systems. An AD environment and its tools should be solidly grounded in standards and should facilitate development of standards-conforming applications.

One very relevant AD standard is the Reference Model for frameworks that the National Institute of Science and Technology (NIST) and the European Computer Manufacturers Association (ECMA) have published. This model defines the services and tools that a Software Engineering Environment (SEE) framework should provide. The reference model defines the relationships between these tools in terms of three types of tool integration:

Visual	The user interface
Control	The mechanism by which tools agree to work cooperatively
Data	How tools agree to exchange data among themselves.

When choosing a framework product, developers should find one that implements the ECMA SEE framework Reference Model.

An AD environment should not only implement a standards-based model, but should be crafted of parts and pieces that are themselves standards-based. Compilers and code generators should implement either ANSI or ISO language standards, or both. GUIs should be built on standard protocols, such as X, and conform to style guidelines, such as OSF/Motif. Communications should use *de facto* industry standard technologies, such as: TCP/IP, Network Computing System** (NCS**) and NFS.

Finally, if the applications are to be easily ported from one open systems platform to another, they should use only library and system calls that are specified in portable operating system standards, such as the POSIX 1003 Series or the OSF/Application Environment Specification (OSF/AES). They should also confine themselves to the use of other APIs, such as Structured Query Language (SQL), which are widely implemented or formally standardized.

1.3.9 AIX Application Development Products

The above requirements are met by the AIX AD products described in Chapter 2, “Introducing AIX Application Development Products and DB2/6000” on page 11. The use of these products in the context of a real project is described in subsequent chapters.

Chapter 2. Introducing AIX Application Development Products and DB2/6000

This chapter identifies and describes the AIX application development products used in the migration and modernization effort. Because DB2/6000 was used, it is described here, too.

2.1 POWERbench/6000 Product Family

IBM offers its AIX application development products collectively in the POWERbench family of products. Each POWERbench product provides a comprehensive programming environment on AIX that supports the construction, test, and maintenance phases of software development. POWERbench products are available for C, C^{++}, FORTRAN, and COBOL development.

The POWERbench products combine SDE WorkBench/6000 with one of the AIX compilers and other appropriate development tools. The POWERbench products come with a single installation process, a single point of contact for defect support and a simplified ordering process. The AIX compilers that are described in this book (or their follow-on products) are components of the POWERbench products. POWERbench products announced at the time of publication of this volume are:

- C^{++} POWERbench
- FORTRAN POWERbench
- COBOL POWERbench.

The AIX application development tools are also offered separately.

2.2 Software Development Environment WorkBench/6000

The IBM AIX Software Development Environment (SDE) WorkBench/6000 is a task-oriented application development environment that provides three key characteristics:

- A common graphical user interface (GUI) for development tools
- A common mechanism for controlling application development tools
- A tool-to-tool communication and coordination by means of a common message service and tool class-based communication protocol.

The SDE WorkBench/6000 is based on the SoftBench** integration framework technology, which IBM licenses from Hewlett-Packard** (HP**). The core components of the SDE Workbench/6000 are:

- Broadcast Message Server (BMS)
- Tool Manager (TM)
- Execution Manager (EM)
- Development Manager (DM).

2.2.1 Broadcast Message Server

The core of SDE WorkBench/6000 is Broadcast Message Server (BMS), which enables tool-to-tool communication and tool control through Tool Manager. Each tool sends out predefined messages in common formats to indicate what it has done and what it requires of other tools.

2.2.2 Tool Manager and Execution Manager

Tool Manager provides the developer with a uniform mechanism for starting, stopping, viewing, and controlling the execution of integrated development tools. Execution Manager fields messages on BMS and automatically starts up the tools for the developer if they are not already running. Tool Manager enables the developer to save the state of tools at any time and restores them automatically on the next invocation.

2.2.3 Development Manager

Development Manager provides the developer with an object-oriented, graphical directory and file management tool. Development Manager provides a view of all, or a subset, of a directory's contents. It enables the definition of file and directory classes based on file name patterns and provides a mechanism for triggering a default action when members of any class are selected. It also enables pull-down menus to be associated with the selection of members of each defined class. Development Manager, by means of its pull-down menus, provides access to menus tailored for any of these user-selectable version control products:

- AIX's Source Code Control System (SCCS)
- AT&T's Revision Control System (RCS)
- IBM's Configuration Management Version Control/6000 (CMVC).

2.2.4 Distributed Computing Support

SDE WorkBench/6000 provides a significant advantage for the developer working in a heterogeneous network environment. SDE WorkBench/6000 incorporates support for:

Distributed Execution All integrated tools run in an *execution context* that includes the *execution host* and *tool scope*. Execution Manager and Tool Manager employ the concept of tool *scope* to determine where and whether to start a tool. Tools have network, host, directory, or file scope depending on how the tool is designed. SDE WorkBench/6000 employs a **spcd** (SubProcess Control daemon) on remote hosts, including hosts running SoftBench. The SubProcess Control daemon performs remote execution and maintains tool-to-BMS communications.

Distributed Data All integrated tools operate in a *data context*. On invocation they are supplied with the name of a host, a directory path name, and a file name (or set of file names) as input parameters. These specify the input data the tool is to operate against. SDE WorkBench/6000 tools that are able to deal with remote data know to look for that data in NFS file systems

	mounted at an algorithmically-determined file system location.
Distributed Display	All integrated tools run as an X Windows client application. SDE WorkBench/6000 attaches to the user's X server and ensures that all integrated tools controlled by it appear on the appropriate screen, regardless of the execution or data context host.

2.3 SDE WorkBench/6000 Bundled Tools

SDE WorkBench/6000 comes bundled with several integrated application development tools that support the compile, edit, and debug cycle, as well as developer-to-developer communications. In addition, SDE WorkBench/6000 supports integrated access to a variety of IBM AIX and vendor compilers, including XL C, XL C^{++}, XL FORTRAN, and Micro Focus COBOL**.

Some of the tools are:

- Program Builder
- Program Editor
- Program Debugger
- Static Analyzer
- File Transfer Program
- Mail Tool
- Message Monitor.

2.3.1 Program Builder

Program Builder employs the AIX **make** utility to compile and link programs. It automatically invokes Program Editor when build errors occur, indicating which lines should be displayed. Program Builder provides the developer with local control over remote compiler and link execution, and provides a mechanism for automatically generating *makefile* files.

2.3.2 Program Editor

Program Editor provides a multiwindow language-sensitive editor based on the OS/2 Live Parsing Editor (LPEX). Program Editor is a highly tailorable, programmable editor that is sensitive to the syntax of C, C^{++}, COBOL, and FORTRAN source code files. It can be made sensitive to the syntax of other languages also. Program Editor can display keywords, special symbols, and other syntactical elements of a file in unique fonts and colors and can detect certain classes of syntax errors.

Program Editor enables simultaneous editing of multiple views of multiple files found on multiple hosts in the network. Tailorable pull-down menus, programmable key combinations, and definable commands (macros) enable the user to perform many tasks. These include: position character, mark and unmark text, copy, delete, and search for text. The user can also issue noneditor commands from Program Editor, interfacing with both the operating system and BMS.

2.3.3 Other Editors

SDE WorkBench/6000 also provides integration support for GNU** Emacs 5.5 or AIX's **vi**. Integration support of the **vi** and GNU Emacs 5.5 editors includes a GUI and BMS control integration; so both editors can send and respond to BMS messages.

2.3.4 Program Debugger

Program Debugger provides a sophisticated graphical user interface to the AIX **dbx** debugger. It provides for continuous display of source code and variables, debugging of processes already running, multiprocess debugging, and full source and machine-instruction level debugging for X LC, X LC^{++}, XL FORTRAN, and COBOL.

2.3.5 Static Analyzer

Static Analyzer helps the developer understand static characteristics of the application. This includes ready identification of all variables, functions, declarations of and references to an identifier, and all source and include files. Like other bundled tools, Static Analyzer receives and responds to BMS messages. Static Analyzer also provides pull-down menu support for automatic invocation of other integrated tools that might operate on the same files.

2.3.6 File Transfer Program

File Transfer Program provides a BMS integration of the TCP/IP **ftp** command. Its GUI simplifies use of **ftp** which sends and receives files across the network.

2.3.7 Mail Tool

Mail Tool provides a BMS integration of AIX's **mailx** program. This tool enables a user to send and receive mail. It also can be programmed to trigger the automatic sending of predefined mail messages to specific users when it detects particular BMS messages.

2.3.8 Message Monitor

The user can send, view, and log BMS messages using Message Monitor.

2.4 Integrated Optional Program Products

IBM provides a variety of Optional Program Products (OPPs) that are integrated with SDE WorkBench/6000. These include application development tools and cross-life cycle development support tools. This section describes the OPPs we used during this project.

2.4.1 AIXwindows Interface Composer/6000

AIXwindows Interface Composer/6000 (AIC) is a software development product that helps the developer design and create OSF/Motif GUIs for applications. The developer manipulates user interface components such as: windows, menus, text fields, scrolled areas, and push buttons to design the GUI, using a drag-and-drop editor to choose from any of the OSF/Motif *widgets*. In addition, the *resources* and *behavior* of the widgets can be edited and modified at design time, and the effect of these modifications can be seen immediately on the screen.

AIC contains a built-in interpreter so developers can create the code link from the user interface to their application logic without having to leave AIC. They can access their own compiled functions and an optional library of AIC *convenience functions* in the C interpreter.

AIC 1.1.1 generates C code that is compatible with X Version 11 Release 4 (X11R4), while AIC 1.2 generates C or C^{++} code that is compatible with X Version 11 Release 5 (X11R5). Both releases of AIC generate a customizable main program and a *makefile* file. In addition, AIC 1.2 accepts User Interface Language (UIL) as input and generates UIL as output, making it useful for preexisting GUIs implemented in UIL. The generated code does not depend on any AIC run-time library; so the code can be run on systems that do not have AIC installed.

Starting with AIC 1.2 provides a control integration with SDE WorkBench/6000. This means AIC automatically invokes Program Editor to edit, Program Builder to run *makefile* files, and CMVC to perform library functions. The generated code does not introduce additional software layers; so the performance, structure, and readability of the code is comparable to self-written programs.

AIC/6000 is IBM's implementation of the UIM/X** product from Visual Edge Limited of Canada. AIC 1.1.1 performs the same functions as UIM/X 2.0, but also implements several X11R5 features that are required to support the level of internationalization achieved with AIX 3.2. AIC 1.2 performs the same functions as UIM/X 2.5. Both releases of AIC differ from UIM/X in that they enable dynamic loading of object code into the interpreter, through a pull-down menu. UIM/X requires the use to exit AIC and relink the UIM/X executable image statically.

2.4.2 Micro Focus COBOL and Micro Focus COBOL Toolbox

Micro Focus COBOL provides a system for developing and running programs written in COBOL. It supports COBOL as defined in ANSI X3.23-1985, as well as other dialects, including:

- IBM OS/VS COBOL
- IBM VS COBOL II
- COBOL/370*
- IBM SAA* COBOL
- X/Open COBOL.

This means that Micro Focus COBOL can be used for both new program development and COBOL programs running in other environments.

2.4.2.1 Micro Focus COBOL

With Micro Focus COBOL, you can easily migrate COBOL applications from many other systems and dialects, including MS-DOS, OS/2, Microsoft Windows, and IBM mainframe COBOL. The system is made up of several parts, most importantly:

- The **cob** command that invokes all stages of compiling and linking a COBOL application. This includes COBOL programs, C routines, assembler routines, and system libraries.
- The *compiler* that determines if programs are valid COBOL programs. It produces intermediate code suitable for execution or debugging.
- The Micro Focus Animator** that enables the programmer to test and debug COBOL source code while the interpreter steps through it

- The Native Code Generator that is used by the **cob** command to produce optimized object code from intermediate code
- The **cobrun** command that dynamically loads and runs the intermediate code
- The Micro Focus Run Time Environment** (RTE**) that is a run-time support library.

Micro Focus COBOL provides facilities to write programs, making full use of:

- Screen displays
- Custom file handlers
- Mixed-language programming.

2.4.2.2 Micro Focus COBOL Toolbox

Micro Focus COBOL Toolbox** includes everything that is in Micro Focus COBOL. It also provides an integrated environment for handling source code and running object code. The most important components of the Micro Focus COBOL Toolbox are:

- The Micro Focus COBOL Toolbox Development Environment, which provides a set of menus that invokes other components of the Micro Focus COBOL Toolbox. The AIX link editor is invoked using function keys.
- The Micro Focus COBOL Editor, which invokes the Compiler and Animator
- The Micro Focus Compiler, which generates intermediate files
- The Micro Focus Animator, which interprets intermediate files for test and debug
- The Micro Focus Generator, which produces optimized object code
- The Micro Focus RTE, which enables the Animator to interpret intermediate files.

2.4.3 Configuration Management Version Control

CMVC has a client-server architecture. CMVC products that run on UNIX-based operating systems are available from IBM, HP, Sun Microsystems (Sun**), including both SunOS** and Solaris**. Any CMVC Client interoperates with any CMVC server in the same version. There are:

- A command line client
- A stand-alone GUI client
- A GUI client that is integrated with SDE WorkBench/6000.

The CMVC server controls source code and other product data in its host's file system, but stores other *meta-data* in a relational database. Meta-data is data about the files being controlled, and data representing CMVC objects. The CMVC server works with any of these Relational Database Management System (RDBMS) products: DB2/6000, ORACLE**, INFORMIX**, and SYBASE**.

CMVC provides the range of functions described in this section.

2.4.3.1 Configuration Management

Software configuration management is the process of identifying, monitoring, and managing changes made to a software baseline. The baseline can include documentation, design and specification data, build and compile control information, and the source code itself. CMVC supports files containing these types of data by associating them with CMVC *components*.

2.4.3.2 Version Control

Version control is provided by means of either SCCS, a product supplied with AIX or SDE WorkBench/6000 or both, or PVCS Version Manager 5.1**, a product from INTERSOLV Inc. Version control ensures that any given version of a file from the current version back to its initial version can be identified and retrieved. Version control applies to both ASCII and binary data files for any of the products.

2.4.3.3 Integrated Problem Tracking

Problem tracking is provided for *feature* and *defect* changes. Features and defects are associated with a CMVC component and identify specific versions of all controlled files that implement the feature or defect.

2.4.3.4 Change Control

An audit trail is maintained by identifying a version of a file, who modified the file, and why it was modified, and by relating it to a defect or feature, if appropriate.

2.4.3.5 Release Management

Release management is implemented by identifying a *release* with a given component and identifying the version of every file comprising that release. Files in a release are *extracted* into specific build directories that are associated with the files.

2.4.3.6 Access Control

Access to files can be controlled by CMVC. A variety of *access authorities* are defined for all files associated with a given component. Users acquire some access authorities implicitly for components they own and inherit some authorities over components because they have those authorities for parent components. Users also can explicitly grant or deny access authorities for components that they own.

2.4.3.7 Automatic Notification

Appropriate users are automatically notified of CMVC actions taken against components and their files. Notification is provided by electronic mail so it can be received without logging in to CMVC. A user's *interest* in notification is configurable by component and type of action.

2.4.3.8 Configurability

The new *configurable* fields can be added to records supporting features, defects, files, and users. CMVC also provides configurability in the *processes* that it enforces as CMVC files, features, and releases move through the various states that CMVC recognizes and controls.

2.5 SDE Integrator/6000

SDE Integrator/6000 integrates application development tools with the SDE WorkBench/6000. SDE Integrator/6000 enables programming a GUI for a tool that is consistent with SDE WorkBench/6000. It also enables programming a tool to communicate with other tools through BMS and to be controlled through Tool Manager and Execution Manager. If source code is available the tool can be integrated by writing a C, C^{++} or Encapsulation Description Language (EDL) program and linking it with the tool's object code.

If source code is not available, a *wrapper* program is written to trap the tool's standard in, standard out, and standard error messages. The wrapper program also produces a GUI for the tool, communicates for the tool sending and receiving BMS messages, and enables tool control through Execution Manager and Tool Manager.

The SDE Integrator/6000 is compatible with HP SoftBench Encapsulator so that tool encapsulations in the HP environment are compatible with the SDE WorkBench/6000 environment.

2.6 DB2/6000 and Related Products

IBM sells a family of DB2 products for its various operating systems. These DB2 products can communicate among themselves by means of SNA or TCP/IP communications mechanisms. The database server on a given platform can interact directly with clients on the same platform, or with clients on remote platforms connected by a network. The database server can also interact indirectly with remote clients on a distant network through a *request server*, which connects to both the database server and the remote client's network.

2.6.1 DB2/6000 Server

DB2/6000 Server is the DB2 server for the RISC System/6000. It enables you to store, retrieve, and update your data on the RISC System/6000. Additional AIX DB2 products described in the paragraphs that follow provide for interoperability and distributed development capabilities. In some cases, the function provided in one of these DB2 products is also included in the database server product.

2.6.2 DB2 Client Support/6000

DB2 Client Support/6000 enables the DB2/6000 Server to be accessed by a number of client applications running remotely on MS-DOS, Microsoft Windows, OS/2, and AIX workstations using TCP/IP communications protocol. If these clients require SNA Advanced Program-to-Program Communication (APPC) protocol access, the AIX database server requires SNA Services/6000 and the SNA support feature of the DB2 Client Support/6000 to be installed, as well. SNA access for clients is supported only on OS/2 and AIX. DB2 Client Support/6000 is not necessary if the client applications are running on the same host as DB2/6000.

2.6.3 DB2 Client Application Enabler/6000

The remote systems on which client code runs require that a Client Application Enabler product be installed on them. DB2 Client Application Enabling/6000 (DB2 CAE/6000), for instance, enables the client code running on one RISC System/6000 to connect to the database server installed on another RISC System/6000. There are also Client Application Enabler/DOS (DB2 CAE/DOS) for installation on Microsoft DOS or Windows hosts and Client Applicaton Enabler/2 (DB2 CAE/2) for installation on OS/2.

2.6.4 DB2 Software Developers Kit/6000

DB2 Software Developers Kit/6000 (DB2 SDK/6000) can be installed on RISC System/6000s that do not also have DB2/6000 Server installed. This enables development on these remote platforms of database applications that are targeted to run on the DB2/6000 Server host.

DB2 SDK/6000 includes a precompiler for C and FORTRAN. It also supports an embedded SQL COBOL API through appropriate Micro Focus products. This API and the precompilers support both dynamic and static SQL statements. DB2 SDK/6000 also provides a DB2 Call Level Interface (DB2 CLI) for C that eliminates the need for precompilation of embedded SQL statements. The DB2 CLI for C can be used from C^{++} applications directly or from the other XL languages by means of interlanguage calls to these C language routines.

The functions of DB2 SDK/6000 are bundled with DB2/6000 Server, to enable development of database applications on the DB2/6000 Server host.

2.6.5 Distributed Database Connection Services/6000

Distributed Database Connection Services/6000 (DDCS/6000) enables the DB2/6000 server to access any Distributed Relational Database Architecture* (DRDA*) database directly. DRDA implements database SQL across a variety of different operating systems, including DB2 on MVS, SQL/DS on VM and VSE, DB2/2* on OS/2, and OS/400 on AS/400 systems. This includes the read, write, and modify capability. While DRDA/6000 was not used in this project, it would have enabled the migrated application to be run on a RISC System/6000 while the database and server remained on MVS. DDCS/6000 also provides a gateway service for AIX, OS/2, MS-DOS, or Windows client applications that need to access an MVS DB2 server. In this case, the DB2/6000 Server need not also be installed on the RISC System/6000 host; the host needs only DDCS/6000.

2.6.6 DB2 Configurations

A variety of combinations of servers, clients, and requesters running on different hardware platforms is supported by this family of products. A complex configuration, such as that illustrated in Figure 1 on page 20, might include all of the DB2 products mentioned above, in addition to having DB2 on the MVS mainframe host.

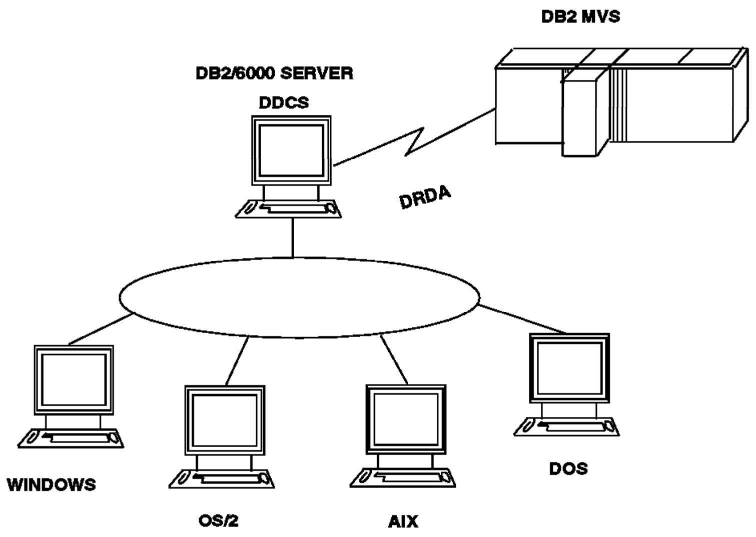


Figure 1. Multiplatform DB2 Configuration Options

In this figure, DB2/6000 Server is installed on the AIX host shown at the top of the LAN diagram. DB2/6000 Server can service client applications running on the same AIX host on which it is installed. If DB2 Client Support/6000 is also installed on this AIX host, DB2/6000 Server can service client applications running on the LAN-attached workstations. However, those hosts must also have CAE/DOS, CAE/2, or CAE/6000 installed on them. A client application running on any of the LAN hosts shown in Figure 1 has DRDA access to the MVS DB2 server through DDCS/6000, which is installed on the DB2/6000 Server host. This DB2/6000 Server host acts as a requester to the MVS DB2 host, passing the DRDA commands and resulting data back and forth between the remote clients and the MVS DB2 Server. Database client applications running on the MVS host cannot access DB2/6000 Server, nor can MVS DB2 act as a requester to DB2/6000. To develop database client applications on the LAN-attached workstations, DB2 SDK/6000, DB2 SDK/DOS, or DB2 SDK/2 must be installed on them. Development is supported on the DB2/6000 server host, however, without installing DB2 SDK/6000.

2.6.7 DB2 Products for Hewlett-Packard and Sun

While they were not used on this project, the DB2 family of products includes products available on other vendor's UNIX platforms that are functionally equivalent to the DB2/6000 products. DB2 for HP-UX** DB2 for Solaris Operating Environment are DB2 Server products available for HP and Sun platforms. Distributed Database Connection Services, Client Application Enabling, Software Developers Kit, and Client Support products are offered for each platform also.

Part 2. Our Project

Chapter 3. The Legacy Application

The legacy application, which we selected to migrate from MVS to AIX, is a typical older COBOL business application. It has some batch components and an interactive component and maintains a database. It is not terribly complex or very large, yet it presents a relevant vehicle for exploring downsizing issues. In this chapter we describe the following characteristics of the application:

- Programs
- Database design
- User interface.

3.1 What the Application Does

The legacy application is used by a company that deals with collectors' items. The customers are organized as club members and every month members get an offer to buy a collectors' item. For example, one offering could be a limited series of a special porcelain plate or some specially designed commemorative coin. This offer is sent out by mail to each member and if the club members want the offered item, they either fill in a request form and send it back, or make a telephone call to confirm their order.

The application maintains a database in which it records information about customers, and the amounts and dates of payments received. Enrollments of new members, address changes, and deletion of members are handled by the application. Some of the input is received during phone conversations with customers and is entered online; other input is processed from mail sent by customers. Standard reports, based on queries of the database, are also supported.

3.2 Programs

The application is written in VS COBOL II* and runs on MVS under TSO, accessing a DB2 database. The user interface is based on Interactive System Productivity Facility (ISPF). The application consists of:

- One interactive program
- Four batch programs
- Two assembler language modules
- Six ISPF panels.

The application consists of the following files:

IBMOUPD The main module of the ONLINE UPDATE (interactive) program. This module, along with the assembler language modules it calls, are collectively referred to as the ONLINE UPDATE program. IBMOUPD handles all tasks related to maintaining the customer database and is started from a small CLIST (Command LIST) file. It:

Enrolls new customers

Makes address changes

Updates payments

Deletes customers.

IBMBUPD The batch update program. This program receives input from a file created by a routine that scans the order confirmation forms sent to the company. It is started from a JCL (job control language) file. This form reflects one of or a combination of the following:

An address change

A payment

A delete request.

IBMBUENR and

IBMBUINS They are two halves of one logical program, which was split into two executable files because of performance considerations. The input is a file that was created by the scanning routine mentioned earlier. These two programs do a batch enrollment of new customers. They are started from the same JCL as IBMBUPD.

IBMBSEL A batch program that produces a report according to given criteria. It is started from a JCL file.

IBMCUST An assembler language module that ensures the parameter passed to it (a customer number) is numeric. It is called by the IBMOUPD program.

IBMDATE An assembler language module that ensures the parameter passed to it is a valid date. It is called by the IBMOUPD program.

When an older application is migrated, certain implementation details that are a result of circumstances on the old platform may not be necessary on the new platform. For instance, the two assembler language modules replaced functions that could have been done by DB2. They were implemented for performance considerations that may not exist in the new program environment.

Because the ONLINE UPDATE program demonstrates a superset of the issues we would encounter in migrating all the programs, we decided to concentrate on migrating it as a proof-of-concept exercise. The migration issues are discussed in Chapter 4, “Defining the Project and Setting Its Goals” on page 29.

3.3 Database Design

The original MVS application used DB2 Version 2.0 for MVS as the database server. The database is organized into four tables:

- CUST
- NAME
- PAYMENT
- ZIP.

The tables have the following column definitions:

CUST	CUSTNO	NUMERIC(10) NOT NULL - (key column)
	REFNO	NUMERIC(10) NOT NULL
	ACTDATE	DECIMAL(6) NOT NULL WITH DEFAULT - (key column)
	ADDRCHG	DECIMAL(6) NOT NULL WITH DEFAULT
	PROFIT	NUMERIC(7,2)
	MAILID	NUMERIC(2) NOT NULL

	SOURCECODE	NUMERIC(3) NOT NULL
	COLLECTCODE	SMALLINT
	DUNNCODE	SMALLINT
NAME	CUSTNO	NUMERIC(10) NOT NULL - (key column)
	FIRSTNAME	CHARACTER(13) NOT NULL
	LASTNAME	CHARACTER(18) NOT NULL
	STREET	CHARACTER(26) NOT NULL
	ZIPCODE	INTEGER NOT NULL
PAYMENT	REFNO	NUMERIC(10) NOT NULL - (key column)
	PAYDATE	DECIMAL(6) NOT NULL
	AMOUNT	NUMERIC(7,2) NOT NULL
ZIP	ZIPCODE	INTEGER NOT NULL - (key column)
	CITY	CHARACTER(20) NOT NULL

Figure 2 shows the relationships between the tables. The NAME table contains data relevant to customers. The PAYMENT table contains data relevant to payments. The CUST table creates a correspondence between the NAME and PAYMENT table records. The ZIP table creates a correspondence between the customer's postal ZIP code and the city name. The CUSTOMER and ZIP tables were created to remove redundancy from the database (that is, they were created to effect normalization of the database to the third normal form.)

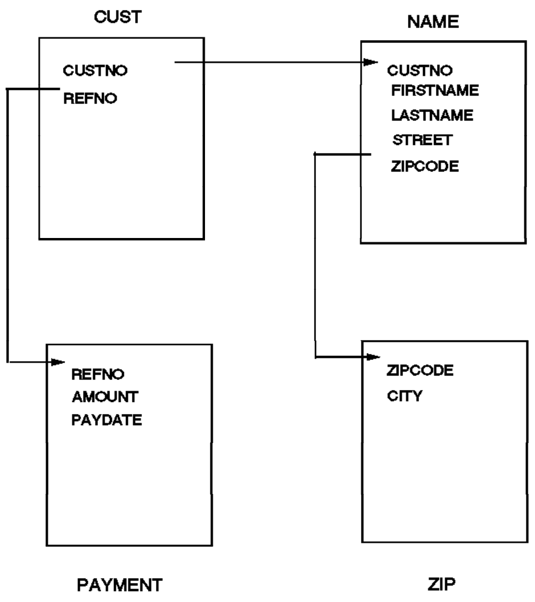
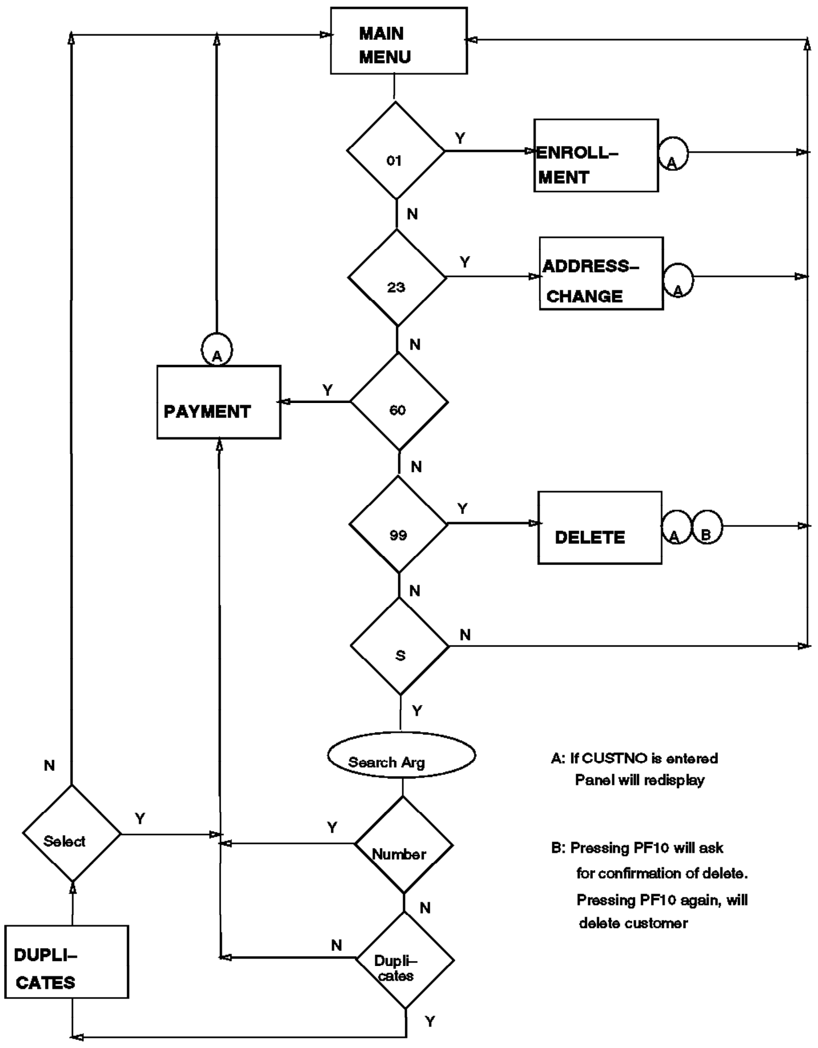


Figure 2. The Database Table Relationships

3.4 The User Interface

The user interface for the ONLINE UPDATE program utilizes ISPF panels. The panels were designed for 3270 color terminals and they use various color schemes to display input and output fields. Users initiate actions by pressing function keys.

The application consists of six panels. The relationships between the panels are shown in Figure 3.



MAIN MENU
01
Y
N
ENROLL–MENT
A
23
ADDRESS–CHANGE
PAYMENT
60
99
DELETE
B
S
Search Arg
Select
Number
Dupli–cates
DUPLI–CATES
A: If CUSTNO is entered Panel will redisplay
B: Pressing PF10 will ask for confirmation of delete. Pressing PF10 again, will delete customer

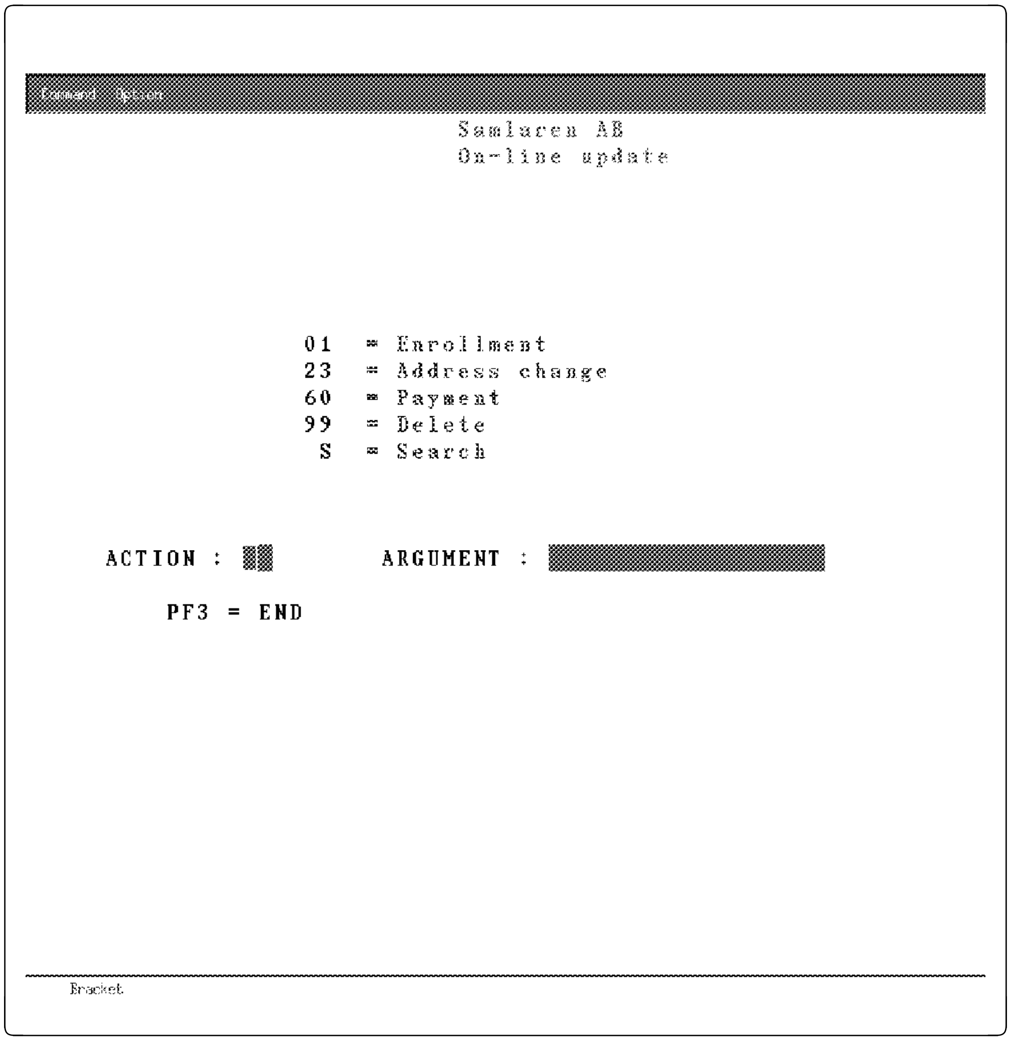
Figure 3. Relationships between ISPF Panels

Figure 4 on page 27 shows the main ISPF panel definitions.

```
)ATTR DEFAULT(%+_)
/* head menu for online updates           */
     %   TYPE(TEXT) INTENS(HIGH) COLOR(WHITE) SKIP(ON)
     +   TYPE(TEXT) INTENS(LOW) COLOR(BLUE)
     _   TYPE(INPUT) HILITE(REVERSE) CAPS(OFF) JUST(LEFT) COLOR(GREEN)
         TYPE(TEXT) COLOR(GREEN) INTENS(LOW)
     "   TYPE(TEXT) COLOR(YELLOW) INTENS(HIGH)
     O   TYPE(TEXT) COLOR(GREEN) INTENS(HIGH)
     ;   TYPE(TEXT) COLOR(RED) INTENS(LOW)
     !   TYPE(OUTPUT) COLOR(RED) INTENS(HIGH) CAPS(ON)
     u   TYPE(TEXT) COLOR(PINK) INTENS(HIGH)
     ?   TYPE(TEXT) COLOR(TURQ) INTENS(HIGH)
)BODY
O                           Samlaren AB
                            On-line update
%
%
%
%
%
%
%                 01 += Enrollment
%                 23 += Address change
%                 60 += Payment
%                 99 += Delete
%                  S += Search
%
%
%
%    ACTION :_Z %      ARGUMENT :_Z                %
!ERRMSG
%       PF3 = END
)INIT
 .ZVARS = '(ACT,ARG)'
 .CURSOR = ACT
)PROC
 VER (&ACT,NB,LIST,01,23,60,99,S,s)
 IF (&ACT = 'S')
    VER (&ARG,NB)
 IF (&ACT = 's')
    VER (&ARG,NB)
    &ACT = TRANS(&ACT s,S)
 VPUT (ACT,ARG)
)END
```

Figure 4. Main ISPF Panel Definitions

On the terminal it looks like Figure 5 on page 28.



```
Command  Option
                                    Samlaren AB
                                    On-line update




                    01   = Enrollment
                    23   = Address change
                    60   = Payment
                    99   = Delete
                     S   = Search


      ACTION :              ARGUMENT :

          PF3 = END
```

Figure 5. The Main Panel as It Appears on a 3270 Terminal

Chapter 4. Defining the Project and Setting Its Goals

Our primary goal in conducting this project was to familiarize the reader with AIX application development products. We quickly realized that the best way to achieve this goal was to illustrate the use of these tools to perform meaningful and typical application development tasks. We decided to migrate and modernize an MVS DB2 COBOL application to AIX as a sample development project. This led to the formulation of a second goal, to explore and document issues related to downsizing and modernizing legacy applications. This chapter describes the project and how we defined its parameters to meet these two goals.

4.1 Getting to Know the AIX Application Development Products

The first purpose of the project was to use IBM's AIX AD products in a realistic application development effort and see how they met the needs of the developers using AIX as a development platform. To make the experience valuable to the widest number of application developers, we defined a project that dealt with several issues. We then chose products to build an AIX AD environment, which would address those issues. The key issues were:

- Mixing multiple vendors' UNIX target and development platforms in a common environment
- Distributed application development tools in a network workstation environment
- Multiple programming languages, language-specific tools, interlanguage programming
- Developers with and without UNIX experience
- Software configuration management requirements
- Team programming
- Degrees and methods of tool integration.

4.1.1 Mixing Multiple Vendors' UNIX Target and Development Platforms in a Common Environment

Any company developing applications on UNIX might want to mix UNIX platforms from multiple vendors in its development environment or to target applications to execute on UNIX platforms from multiple vendors. Often the company is interested in doing both. Companies choosing UNIX want the freedom to migrate their applications to the most cost-effective platform easily, and to distribute client and server portions of their applications across mixed platforms. The tools these companies use to build applications must also interoperate in a standard network environment with both client and server components being able to execute on any combination of the major vendors' computers. With this in mind, we selected certain AIX AD products because they are available not only on AIX for IBM's RISC System/6000, but also on UNIX-based computer systems from HP and Sun.

At the time we conducted this project, cross-platform availability of the AD products we selected was as follows:

- HP's SoftBench was available from HP for both Sun and HP platforms, and was licensed to IBM, which sells it under the name SDE WorkBench/6000 for the RISC System/6000.

- UIM/X was available on a variety of UNIX platforms including Sun and HP, and was sold by IBM under the name AIC/6000 for the RISC System/6000.
- Micro Focus COBOL and Micro Focus COBOL ToolBox were available on Sun Solaris/1, Solaris/2, and IBM RISC System/6000; Micro Focus COBOL was also available on HP 9000, 600, and 800 series platforms. Both products could be obtained from Micro Focus directly or from the platform vendors.
- CMVC for Sun was available from IBM—server for SPARCserver** 10 or any binary compatible SPARCserver and client for any SPARCstation**, and CMVC for HP was available from IBM—server for any HP 9000 Series 700 or 800 workstation and client for any HP 9000 Series 400, 700, or 800 workstation; CMVC/6000 was available for the RISC System/6000—server on any model 320 or larger and client on any model.
- DB2 family of products was available from IBM for any RISC System/6000 with AIX Version 3.2, for any HP 9000 Series 700 or 800 workstation supported by HP-UX Version 9, and for any Sun processor supported by Solaris Version 2.3.

4.1.2 Distributed Application Development Environment

Companies looking at open systems development want to create a distributed application development environment, which has the flexibility to grow and change with their needs. A goal of this project was to set up a typical distributed open systems development environment. Ours consisted of several RISC System/6000 compute and file servers with developer workstations being either direct-attached high function terminals (HFTs) or RISC System/6000 X stations. Compute resources, such as specific language compilers and debuggers, were allocated to some hosts, while library services and database managers were located on other hosts.

We did not create a multiple vendor development environment, however. We used only RISC System/6000 equipment and AIX. Had we substituted other vendor equipment for any workstation or server in our configuration, we could have explored additional issues, such as:

- Remote and concurrent build management on heterogeneous platforms
- Advantages and disadvantages of platform-specific tools
- Installation and system administration on multiple platforms
- Use of non-UNIX desktop developer workstations such as OS/2 platforms running Presentation Manager* X Windows (PMX) or MS-DOS platforms running an X server implementation.

But on the whole, the configuration was designed to model the distribution of AD resources, such as compiler compute servers or database servers that might be found in a multiple vendor open systems development environment. The hardware and software topologies are described in detail in Chapter 5, “Our Application Development Environment” on page 39.

4.1.3 Multiple Programming Languages

An important goal was to determine if the AIX AD environment supports the traditional language of business application development, which is COBOL, as well as it supports the native UNIX language, which is C. We also wanted to explore how the environment supports the C^{++} programmer's needs. Another goal was to document where the AIX AD products could be tailored for use with each language. We also wanted to explore interlanguage calling issues, and testing and integrating mixed-language applications.

4.1.4 Choosing the Developers

To examine how appropriate AIX AD products were for a typical IBM AD customer, we chose the developers for this project to mimic the skills and developer background we anticipated an IBM customer application development team might have. Our developers had strong skills in their respective languages: C, COBOL, and C^{++}. One had knowledge of IBM mainframe systems and possessed experience in application development for those environments. This team member also possessed expertise with the legacy application and programming skills with COBOL. Another of our developers had OS/2 experience, GUI software development expertise, and programmed primarily in C. A third member was the team's C^{++} expert and had extensive UNIX experience. However, none of our team members had experience with the SDE WorkBench/6000 or the integrated versions the AD products.

4.1.5 Software Configuration Management Requirements

While we wanted a relatively small scale development effort, we also wanted to examine how the AIX AD environment supported realistic software configuration management requirements. Therefore, we decided to control the multiple software baselines for both MVS and AIX on our AIX development platform. We also decided to use CMVC's change control functions. Because our focus is on the developer, and not on the configuration manager, this book only mentions how CMVC is accessed from SDE WorkBench/6000. It does not go into any depth on how we used CMVC for software configuration management on this project. Refer to the companion volume *Did You Say, CMVC?* for that perspective.

4.1.6 Team Programming

We wanted to examine how well the AIX AD environment we created supported typical developer interaction patterns. Developers customarily spend some time together in cooperative efforts as a team. The rest of the time they work relatively independently, but in parallel. The AD environment becomes the glue that holds their work together and serves to help coordinate their efforts. We designed the project so that the three developers worked in separate areas initially (COBOL porting, GUI development, C^{++} implementation), but later worked together to integrate and test their code.

4.1.7 Degrees and Methods of Tool Integration

No matter how rich the set of tools that are bundled with an AD framework product, the framework must be flexible enough to allow the integration of additional tools. For SDE WorkBench/6000, achieving *visual integration* means adding a GUI that looks and acts like the GUIs presented by the bundled tools. Achieving *control integration* means making it possible for the tool to send and receive BMS messages and to respond to Tool Manager and Execution Manager controls.

Source code for tools is often unavailable. Therefore, it is necessary to have a mechanism for *wrapping* a layer of software around an existing tool that implements visual and control integration, but does not require recompiling or relinking the tool. A framework product should also provide an integration API to use when source code for a tool is available. One goal of this project was to explore these two mechanisms provided by SDE Integrator/6000 for integrating new tools with SDE WorkBench/6000.

At a minimum, control integration enables the user to start and stop a tool using Tool Manager. A greater degree of integration would enable the tool to respond to additional BMS messages appropriate for its tool class, to recognize its execution and data contexts, and to trigger the start of other appropriate tools indirectly, and to provide access to other integrated tools from pull-down menu options. Development Manager can also be tailored for a new tool and made aware of the types of file objects on which it operates. The combined effect of tight integration of a tool with the framework should improve developer efficiency and productivity. Therefore, a final goal of the project was to compare and contrast the impact on the developers effectiveness of different degrees of tool integration with SDE WorkBench/6000.

4.2 Downsizing and Modernizing an MVS Application

Refining our goals and defining the development effort required us to look at the topics of downsizing and object-orientation from several perspectives. We defined a specific plan of action to cover many issues and examined the feasibility of its success.

4.2.1 Defining a Downsizing and Modernizing Strategy

We knew that not every company desiring to downsize would have the same needs, follow the same timetables, or use the same approaches. We analyzed various migration strategies and defined the goals for our project, hoping to touch on the most representative issues. The project plan included the following steps:

- Identification of a representative subset of the application for migration
- Minimal effort migration of the application and database
- Exploration of the issues involved in creating and running parallel MVS and AIX versions with a common base of COBOL source
- Replacment of the 3270-based user interface with a C language GUI having an appearance similar to the 3270-based user interface
- Improvement of the C language GUI to make it object-oriented and consistent with modern standards
- Replacement of the COBOL source code with a C^{++} implementation integrated with a C^{++} implementation of the improved GUI.

4.2.1.1 Proof of the Concept

The fundamental principle we assumed all companies would apply was that of a clearly planned series of activities and verification of the success of one step before proceeding to the next. Most companies would probably do a feasibility study to identify and evaluate issues associated with their applications. They would probably choose an appropriate subset of a typical application and migrate it as proof of the concept. We incorporated this principle into the planning of our effort. We identified the IBMUPD program as our sample to migrate.

4.2.1.2 Database Migration Options

We assumed that some companies would use AIX only as a development platform for their MVS-targeted applications. Others would probably move only their client applications to AIX, leaving the database server on MVS. Finally, some would migrate both the application and its database server to AIX.

By using DB2/6000, we were assured of support for these various options. Since we had limited time, we migrated both our application and its database server to AIX. We also explored the option of modifying the MVS version of COBOL to produce a common subset of COBOL to use in both AIX and MVS versions of the application. This would facilitate running both versions of the application in parallel against a common database server and assist in validating the migrated version against the original. It also gave us insight into software maintenance on AIX for applications targeted for execution on MVS.

4.2.1.3 User Interface Options

Some companies would modernize their applications by putting a GUI on them, while others would maintain their traditional IBM 3270 character-based user interface and execute it by means of emulation software. There also might be companies that would implement a user interface free of software dependencies on the old environment but that would still be very similar to the MVS user interface. This would have significant savings in terms of end user training costs. It also might be interim strategy to prevent the incremental deployment of the newer end user workstations and/or displays from becoming a source of friction or confusion in the end user community. Our project explored all these options.

4.2.1.4 Object-Orientation Options

Finally, we gave thought to the companies that would reimplement their existing applications in object-oriented technologies. We determined that they would be equally inclined to approach this in stages. At first they might simply give the applications an object-oriented user interface. Later they might implement significant portions of the application in C^{++}, integrating it with their object-oriented GUI.

Adopting object-oriented technologies does not imply that you must abandon your investment in relational database technology. However, most database vendors provide only non-object-oriented APIs with their database products. Therefore these companies also need an object-oriented programming API to provide their applications with access to the database. This API provides an intermediate interface compatible with the new source code and the old database API.

4.2.2 Examining the Feasibility of the Project

In examining the feasibility of migrating and modernizing our application to AIX, we posed the following questions:

Code	In what language is the application written and is that supported on the target platform?
Data	How is the data stored? Is it in a relational database, hierarchical database, flat files, or some other type of files? Is there vendor software or built-in support for this type of data storage and retrieval on the new host? How can data be extracted, physically relocated, and imported into the new

	database? How can the database tables and table relationships be recreated?
User Interface	Can the user interface be ported or emulated on the new host? If the user interface is replaced, what are our options on the new host? Should the user interface remain character-based or be made graphical?
Program Structure	Is the application well structured? Can it be easily modularized so changes necessitated by migrating to the new host can be localized in separate modules? Is the user interface separated from the application logic? Does the application have a client/server architecture?
API Dependencies	Does the application depend on APIs other than the database API? Are these APIs also supported on the new host? Does it have operating system dependencies? Does the new operating system provide equivalent function?

4.2.2.1 COBOL Code Issues

Our goal was to rewrite as little as possible of the COBOL code, performing a minimal effort migration. There were two reasons for this. The first was to show how a minimal effort migration could be done. The second was to show how two versions of the application might be developed, one for MVS and one for AIX, which would share a substantial portion of common COBOL code. This meant that language compatibility was essential.

Our application was written primarily in IBM VS COBOL II. By using Micro Focus COBOL for AIX, we were assured of reaching this goal because the compiler supports numerous dialects, including IBM VS COBOL II. This was the most significant code issue to be resolved, but others remained.

Small portions of the application were written in assembler language. Because this code would not migrate, it would have to be rewritten. Options were:

- Rewrite the modules in some other language.
- Rewrite the modules in COBOL by incorporating their function into the COBOL modules calling them.
- Replace the modules with function provided by DB2.

This assembler language code had been written to replace existing DB2 function as a performance enhancement on MVS. While this was not necessary on the RISC System/6000, we decided to rewrite the assembler language modules in C. This was better than modifying the COBOL code because we wanted to maximize the common COBOL between the AIX and MVS versions of our application. We chose C for portability. Because most UNIX systems are implemented in C, there would be a C compiler on any UNIX platform we might use in a later migration. Additionally, C is widespread on both small systems and mainframes.

Because we chose to implement the assembler language routines in C, we had to deal with the interlanguage calling conventions of COBOL calling C. Because we decided to implement the GUI in C, we had to deal with the interlanguage calling conventions of C calling COBOL also. The COBOL documentation told us how to call COBOL from C, but we could not find documentation describing how C calls COBOL. We mastered these by experimentation.

4.2.2.2 C^{++} Code Issues

AIX provides a C^{++} compiler based on the proposed ANSI standard for that language. Compilers from many vendors conform to this proposed standard already; so using this compiler ensured portability. The only interlanguage calling that might arise would be C^{++} calls to C language subroutines, which is fully supported by the C^{++} language.

4.2.2.3 Data Issues

The goal of this project in resolving any data issues was to migrate the database to AIX, automate any necessary data transformations, and create the exact same database tables on AIX as on MVS. As mentioned earlier, we abandoned the idea of examining database and application distribution issues. The HP and Sun DB2 products were announced, but not yet available, so we could not explore issues of migrating data between UNIX systems.

The following data issues needed to be resolved:

- The database was located in Sweden and contained Swedish language data, which included some special symbols. National Language Support (NLS) issues, such as collation sequence, also had to be explored.
- The data was stored in EBCDIC on MVS and would be in ASCII on AIX. Some mechanism had to be determined to extract the data, convert it, and insert it into the AIX database.
- The new database tables to be created had to be identical to the old ones. Any changes to the database design would imply significant changes to the COBOL source code and make maintenance of common baselines impossible.

4.2.2.4 User Interface

The MVS application user interface was based on ISPF. ISPF is an IBM 3270 terminal-based mechanism available on IBM mainframes that supports character-oriented display and input screens and function keys. At the time of this project, ISPF was not available on the RISC System/6000; so the user interface had to be replaced or accommodated.

We examined five possibilities and implemented two GUIs, one for the COBOL version and one for the C^{++} implementation. These possibilities were:

1. Emulating ISPF on AIX, using a utility available for IBM internal only use
2. Creating a character-oriented user interface using the Micro Focus Dialog System, which is a Micro Focus COBOL Toolbox add-on product that separates screen and keyboard input/output from the COBOL application program
3. Creating a character-based user interface in COBOL using ACCEPT and DISPLAY statements
4. Replacing the ISPF panels with a GUI that closely mimics the ISPF panels
5. Replacing the ISPF panels with an object-oriented GUI that reflects Common User Access* (CUA*) guidelines. CUA is a member specification in IBM's Systems Application Architecture* (SAA).

We used AIC/6000 to generate C code for the ISPF-like and the CUA-based user interfaces. We regenerated the CUA-based user interface in C^{++} for use with the C^{++} implementation of the application.

4.2.2.5 COBOL Program Structure Issues

We had already decided on the goal of deriving an MVS and an AIX version of the application that shared common COBOL code. We now needed to determine what impact this goal had on the original MVS COBOL source code, which implemented as a single large module, except for the ISPF panels which were broken apart into separate modules. There was no other obviously MVS-only code or data embedded in the main program module. The remainder of the application appeared to be well structured and could be broken apart into separate modules, some of which could be shared between the AIX and MVS implementations. Figure 6 illustrates generically how a large, well structured application can be broken apart into separate modules.

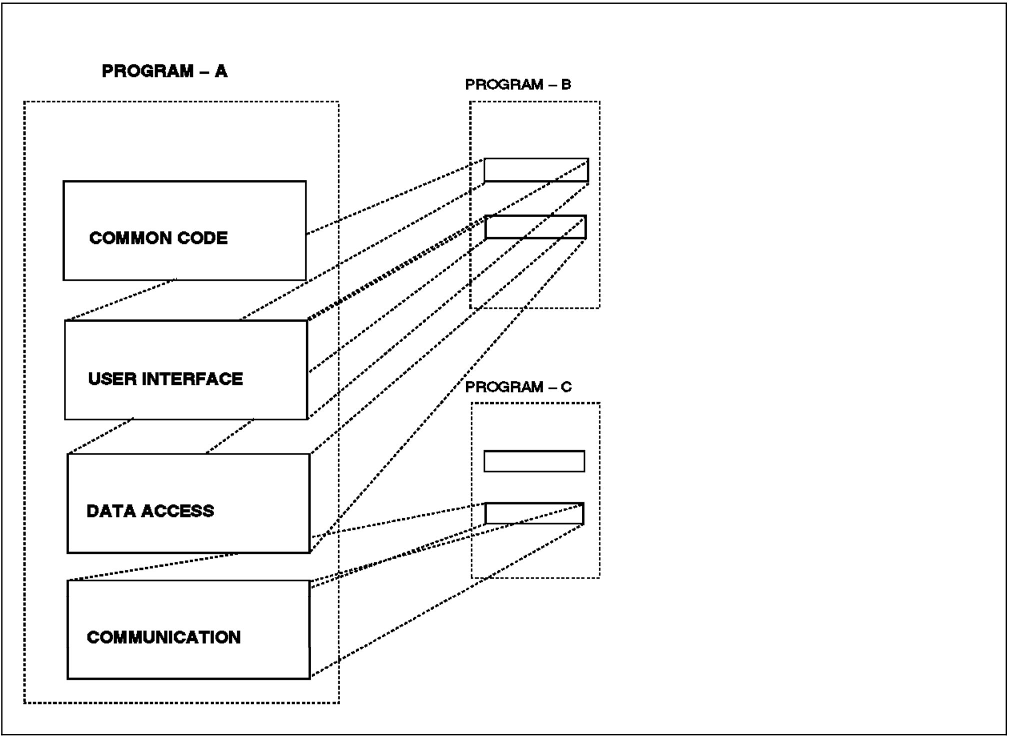


Figure 6. Modularization of a Well-Structured Program

We then examined options for maintaining the common COBOL code, and determined that the COBOL mechanism of COPY code would serve our purposes. We set the goal of creating a body of COBOL code, which during the compilation process would bring in platform-specific modules in some cases, and platform-common modules in others. Each version would also have some non-COBOL modules that were unique to the platform linked with the resulting COBOL object code.

On MVS, the original code was contained in a COBOL PROGRAM, and the user interface was implemented by calls to ISPF. The program logic itself was modular, having one section for each display panel that contained the code to process the panel and the related database logic. On AIX, the COBOL code was more like a subroutine library. The main body of code was a simple event-driven C program, which generated GUI displays. OSF/Motif *callbacks*, triggered by the GUI process, called the COBOL subroutines that implemented the database logic.

Figure 7 on page 37 shows how we planned to transform the application from the original MVS code structure, to the new MVS structure, and finally, to the AIX structure.

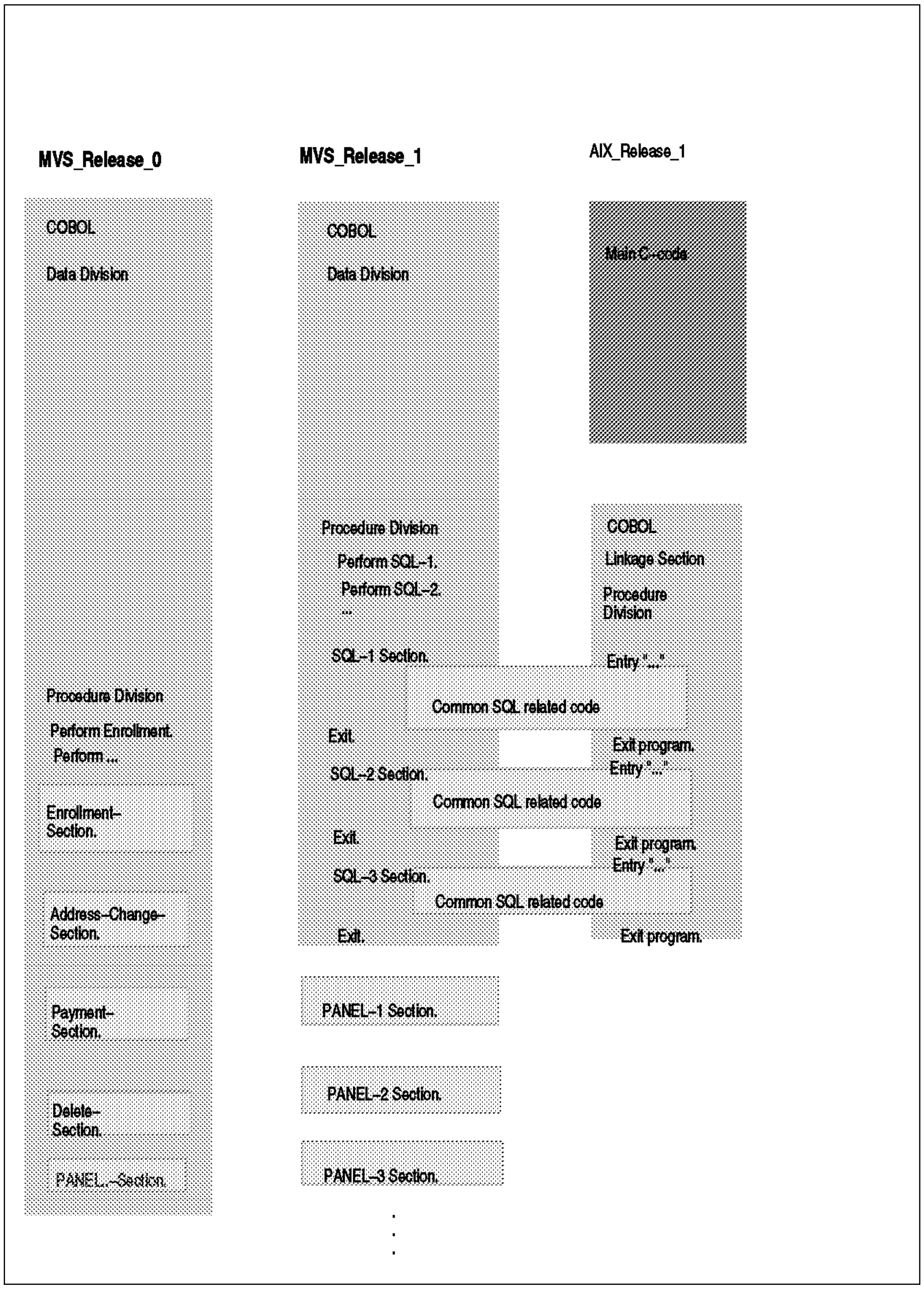


Figure 7. Evolution of the COBOL Code. NOTE: The box sizes do not represent relative code sizes.

4.2.2.6 C++ Program Structure Issues

We wanted to reuse the improved GUI design when we reimplemented the application in C++. This was feasible because AIC 1.2 could generate either C or C++ from the same design files. When AIC generates C++ it defines a C++ class for each *interface* or window. We decided therefore to derive an application class jointly from the GUI window class and from a newly written application class that implemented the database logic. We also created component classes to encapsulate the DB2/6000 CLI for C.

When planning this, however, there were many unknowns, for example:

- Would classes generated by AIC enable overriding of their methods to be overridden in derived classes?
- Would classes generated by AIC use component classes that our application classes could also manipulate?
- Would we need to create component classes to deal with the database?

During the project, we arrived at answers to these questions by experimentation and examination of the AIC 1.2 generated code.

4.2.2.7 API Dependencies

Our biggest API dependency was on the embedded SQL in the COBOL source code. As we expected, Micro Focus COBOL provided full support for this API. When we replaced the COBOL code with C++ code, we decided to replace the SQL calls embedded in the COBOL source code with calls to the DB2 CLI for C, because C subroutines are directly callable from C++ code.

Chapter 5. Our Application Development Environment

A wide range of hardware, software, and file system configurations that can provide the foundation for the application development environment and tools are described in Chapter 4, “Defining the Project and Setting Its Goals” on page 29. In this chapter we describe the specific hardware, software, and file system topologies we created for our project. The sections that follow describe why we placed our users at, and the various compute services on, the specific computers in our distributed application development environment.

5.1 Fundamental Guidelines

We chose a specific number of computers so our project could model a typical distributed application development environment. In such an environment:

- Different makes, models, and brands of computers are acquired over time and do not always support identical operating system releases, peripherals, and software products.
- Different application development tools are installed exclusively on a few computers and made available to all users by remote execution. These computers function as *compute resource servers*.
- Some software resources might be universally installed, such as Xservers or development environment framework products.
- The workstation consoles and network-attached graphical displays are dedicated to specific users, but users can log in to various computers on the network at different times.
- Disk capacity is concentrated on a few hosts and made available to all hosts. These hosts function as *file servers*.

5.2 Hardware and Network Topology

Our network topology, as illustrated in Figure 8 on page 40, consisted of four IBM RISC System/6000 computers and a RISC System/6000 X station connected on a token-ring LAN. Table 1 on page 40 identifies the host names, IP (Internet** Protocol) addresses, and physical characteristics of these computers. We refer to these RISC System/6000s by their host names in this book. The computers are named after seas: *yellow*, *bering*, *bengal*, and *sargasso*. The X station was named *zorin1*.

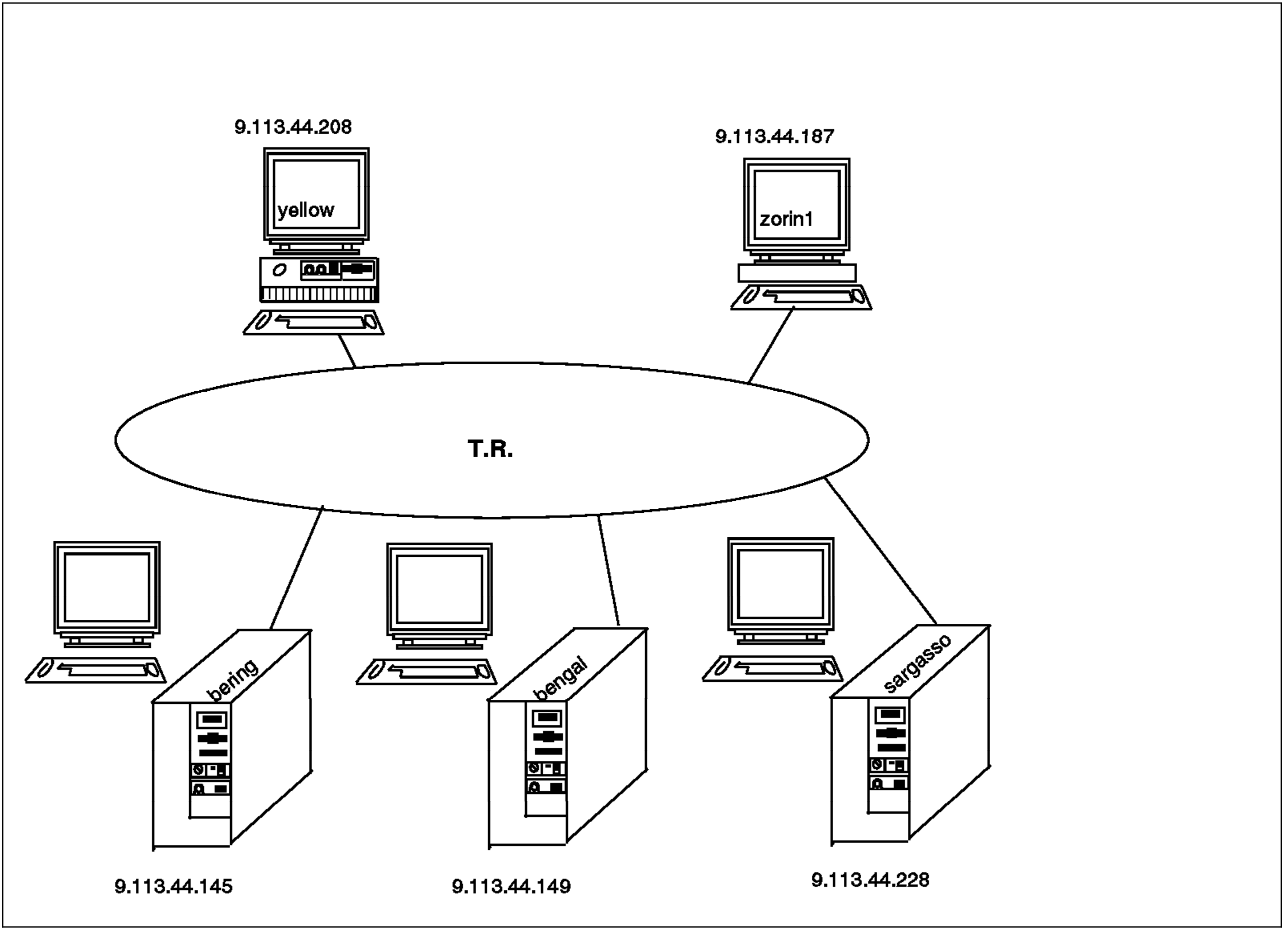


Figure 8. Our Network Topology

Table 1 (Page 1 of 2). Our Hosts, Internet Addresses, and Hardware Configurations

Host Name / IP Address	**Make / Model**	**Hardware Configuration**
yellow / 9.113.44.208	7013 / 340	2 GB disk, 32 MB memory, 6091-19 display, 500 GB tape, 3.25 inch floppy, 4/16 MB Token Ring, Colorgraphics Display Adapter
zorin1 / 9.113.44.187	7010 / 130 (Xstation)	6091-19 display, 4/16 MB Token Ring
bering / 9.113.44.145	7013 / 52H	2 GB disk, 32 MB memory, 6091-19 display, 8 mm tape, 3.25 inch floppy
bengal / 9.113.44.149	7013 / 52H	2 GB disk, 32 MB memory, 6091-19 display, 2.3 GB tape, 3.25 inch floppy, 4/16 MB Token Ring, Colorgraphics Display Adapter

Table 1 (Page 2 of 2). Our Hosts, Internet Addresses, and Hardware Configurations

Host Name / IP Address	Make / Model	Hardware Configuration
sargasso / 9.113.44.228	7013 / 520	400 MB disk, 32 MB memory, 6091-19 display, 3.25 inch diskette drive, 4/16 MB Token Ring, Colorgraphics Display Adapter

5.2.1 Display and X Server Requirements

Each developer requires a high function terminal or X station because SDE WorkBench/6000, which has an OSF/Motif GUI, requires a high resolution (1024x768 or greater) monochrome or color bit-mapped display.

We could have used OS/2 workstations with a Presentation Manager X Window System emulation program, such as PMX. We could have also used MS-DOS workstations with any of several third-party X Window System products. There would have been font and color issues that we did not have, but those workstations could have served effectively in this topology.

5.2.2 Memory and Disk Requirements

We found suggestions for memory and fixed disk storage presented in each product's installation manual. For example, *Installing IBM AIX SDE WorkBench/6000 and IBM AIX SDE Integrator/6000* indicates that SDE WorkBench/6000 requires at least 16 Megabytes (MB) of RAM, but it also advises that it works much better with 32MB. *Configuration Management Version Control Server Administration and Installation, Version 2 Release 1*, states that the CMVC server requires 16MB over and above that required for the database product. The *ORACLE for IBM RISC System/6000 Installation and User's Guide Version 6.0* indicates that ORACLE requires 16MB. AIX and AIXwindows require at least 16MB.

The reader should note that these memory requirements are not necessarily additive. Because AIX memory is backed by page space on disk, AIX can create the illusion of much greater virtual memory than is actually present in the physical chips. We found that 32MB of memory performed adequately on all our systems. This included the system that supported multiple SDE WorkBench/6000 users, the CMVC server, and multiple CMVC clients.

The amount of disk space required to serve as page space for a Licensed Program Product is usually much larger than the disk space requirement for installing the product. Recommendations for page space are additive, and increasing page space can significantly improve performance problems related to memory constraints. For example, *Installing IBM AIX SDE WorkBench/6000 and IBM AIX SDE Integrator/6000* advises that you need at least 16MB of free page space for the first user and 13MB more page space for each additional SDE WorkBench/6000 user on the system. ORACLE not only uses significant pageable memory, but also pins memory thereby reducing the amount of memory left over for other applications. AIXwindows also requires considerable memory, and therefore page space. Typically, we allocated 100MB of page space on these systems.

Disk space for AIX Journaled File Systems and database-managed disk partitions can be significant for a development project, although it need not be available on every system in the network. ORACLE reserves a large disk partition and manages that space itself. On installation, ORACLE required 250MB of disk space for this partition. CMVC, which was installed on this same system, stored all the versioning data for the development source code and related files in AIX Journaled File Systems space. The disk space required for this can vary depending on the size of the project and the number of separate files involved, and it can be significant.

5.3 Software Topology

Our software topology illustrated in Figure 9 shows on which hosts the various client and server portions of the tools and SDE WorkBench/6000 were run. It also shows which hosts acted as file servers. We decided how to distribute software and data across the various hosts by balancing the following goals:

- Clients should run on the host at which the end user runs the X server as much as possible, to minimize display update delay and network traffic.
- Servers should run on the host with appropriate memory and disk resources.
- Software should run where prerequisite and corequisite software is available.

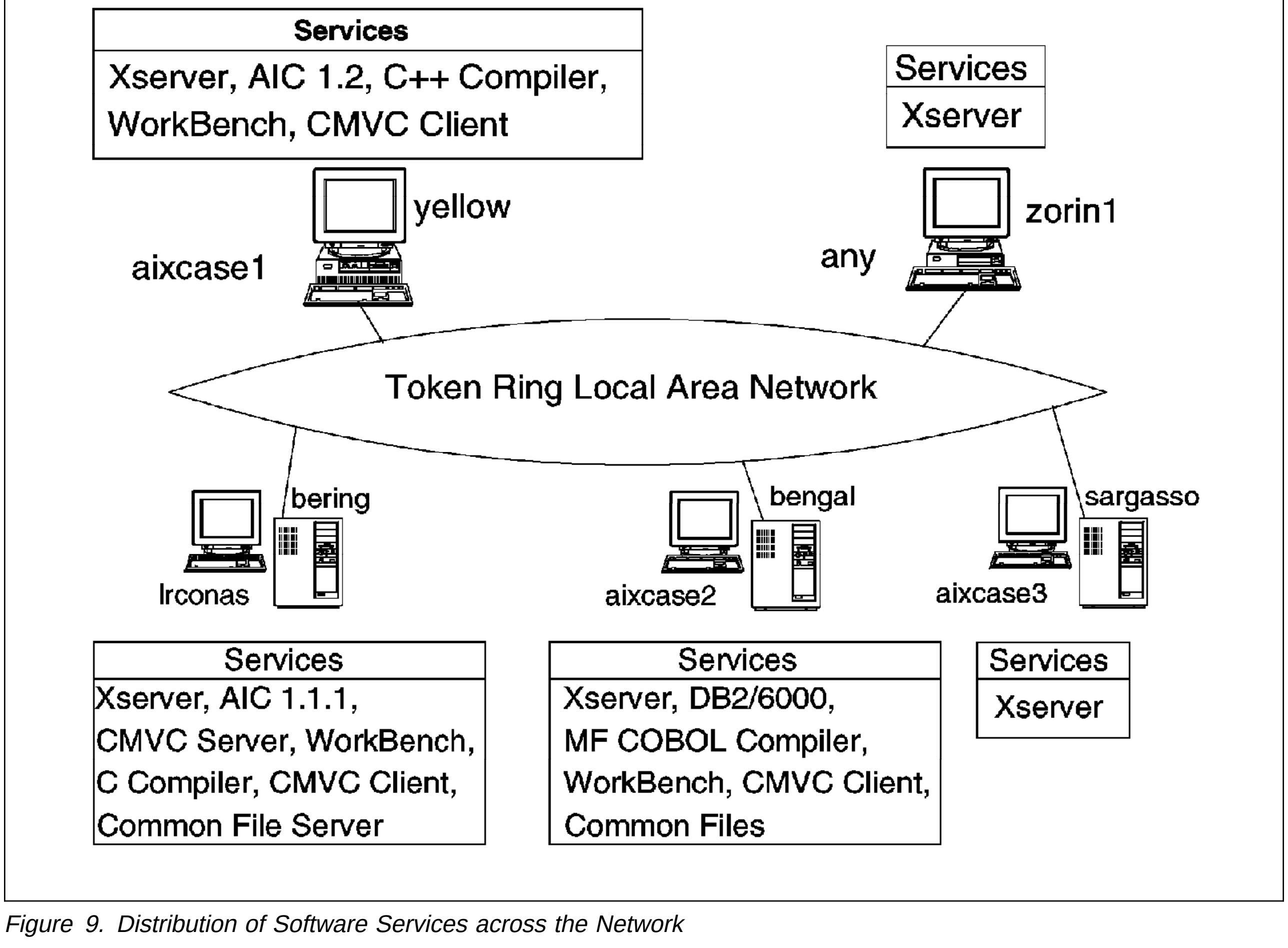

Services
Xserver, AIC 1.2, C++ Compiler, WorkBench, CMVC Client
yellow
aixcase1
Services
Xserver
zorin1
any
Token Ring Local Area Network
bering
lrconas
bengal
aixcase2
sargasso
aixcase3
Services
Xserver, AIC 1.1.1, CMVC Server, WorkBench, C Compiler, CMVC Client, Common File Server
Services
Xserver, DB2/6000, MF COBOL Compiler, WorkBench, CMVC Client, Common Files
Services
Xserver

Figure 9. Distribution of Software Services across the Network

5.3.1 AIX and LPP Versions, Releases and Levels

All hosts had AIX/6000* Version 3.2.3-extended installed, except for *yellow*, which had AIX/6000 Version 3.2.4 installed. We needed one system at this operating system service level because AIC Version 1.2 generates C or C^{++} code that is compatible only with X Windows Version 11 Release 5 (X11R5) and OSF/Motif 1.2. AIX 3.2.4 is a prerequisite to AIXwindows 1.2 (IBM's implementation of X11R5 and OSF/Motif 1.2).

Other systems remained at the lower service levels because the versions of SDE WorkBench/6000 and CMVC, which we intended to use, were not yet certified for either the newer operating system or AIXwindows level. Furthermore, we wanted to use AIC 1.1.1, so we could compare the differences between the two versions of AIC in their degree of integration with SDE WorkBench/6000. AIC 1.1.1 generates C code that is compatible only with X Windows Version 11 Release 4 (X11R4) and OSF/Motif 1.1, which are compatible only with AIX 3.2.3. IBM provides X11R4 compatibility libraries on later AIX releases. Applications suitable for X11R4 environments can be run if the environment variable `LIBPATH` is set up so the compatibility libraries precede the X11R5 libraries. (This was relevant to applications such as SDE WorkBench/6000 and CMVC, which were X11R4 compatible during the project).

The *sargasso* host was of an indeterminate AIX 3.2 level and had limited disk space, so neither SDE WorkBench/6000 nor developer tools were installed on it. We only logged-in, run the X server, and remotely run SDE WorkBench/6000, CMVC, and the other AD tools on other hosts.

Some of the standard Licensed Program Products (LPPs) installed on each system were:

Application Development Toolkit (ADT), which included **make**, **SCCS**, **dbx**, and other standard UNIX developer tools

Basic Operating System Extensions (BOSext1,BOSext2), which included a variety of basic services such as C shell and mail handler

Network Services, which included TCP/IP device drivers and network interface software services, such as Network File System (NFS), Network Information Services (NIS), Domain Name Service (DNS), and Network Computing Services (NCS).

Table 2 shows the software configurations installed on the various hosts.

Table 2 (Page 1 of 2). Software Configurations

Host Name	AIX & LPPs	AIXWindows	AD Environment & Tools
bering	AIX 3.2.3.e, BOSext1, BOSext2, NetLS 2.1, ADT, TCP/IP, NFS, NCS, Xstation Mgr. 1.3, X LC 1.2.1	X11R4, OSF/Motif 1.1	SDE WorkBench/6000 1.2.2, Oracle 6.0.36, CMVC 2.1 Server & Client, AIC 1.1.1, SDE Integrator/6000 1.2

Table 2 (Page 2 of 2). Software Configurations

Host Name	AIX & LPPs	AIXWindows	AD Environment & Tools
bengal	AIX 3.2.3.e, BOSext1, BOSext2, NetLS 2.1, ADT, TCP/IP, NFS, NCS, Xstation Mgr 1.3, XL C 1.2.1	X11R4, OSF/Motif 1.1	SDE WorkBench/6000 1.2.2, CMVC 2.1 Client, Micro Focus COBOL 3.1, DB2 CAE/6000 DB2/6000
sargasso	AIX 3.2.3, BOSext1, BOSext2, TCP/IP, NFS, NCS	X11R4, OSF/Motif 1.1	none
yellow	AIX 3.2.4, BOSext1, BOSext2, NetLS 2.1, ADT, TCP/IP, NFS, NCS, Xstation Mgr 1.3,XL C^{++} 1.1.2	X11R5, OSF/Motif 1.2	SDE WorkBench/6000 1.2.2, CMVC 2.1 Client, AIC 1.2, DB2 CAE 1.1, DB2/6000

5.3.2 X Windows Services

The terms client and server in the X Windows paradigm often confuse people. The X client is an application that makes use of the input/output services provided by an X server associated with a given display. The X client can run on one computer while the X server it accesses runs on another computer, but both X server and client can run on the same computer. An X client is often also the client portion of a separate distributed application. For example, CMVC is a client/server application whose client and server portions may run on separate computers on the network. The CMVC client makes use of the services provided by the CMVC server. It also uses the services provided by an X server, and therefore is an X client application. A CMVC client might be running on one host, making input/output requests through an X server that runs on a second host while accessing the CMVC server running on a third host. The X server always runs on the host where the display is physically attached. An X station is a host that is capable of running only the X server; it is not a full function computer.

We assigned developers to hosts by matching the developers' areas of responsibility with the hosts on which the AD products supporting those responsibilities were installed. Developers who could not sit at the console of the host supporting the tools they needed, could sit at an X station or another system console where they could run at least the X server. They could then remotely run the tools they needed. Each developer also had occasion to remotely run some AD products. In subsequent chapters we refer to specific users by their UNIX login names and to specific hosts that are running applications on their behalf, to illustrate the users' interaction with their tools and SDE WorkBench/6000. Table 3 on page 45 identifies the developers' login user IDs, areas of responsibility, and hosts where their displays were attached.

Table 3. Developer Workstation Assignments

Host Name	**User ID**	**Development Responsibilities**
bering	*lrconas*	Project management, software configuration management
bengal	*aixcase2*	COBOL DB2 development
saragasso	*aixcase3*	GUI development.
yellow	*aixcase1*	C^{++} development
zorin1	any user	Alternative workstation

5.3.3 Network Software

DNS was also configured, but none of our systems were configured as a name server. We did not use Network Information System (NIS), formerly known as *Yellow Pages***; instead, we kept our `/etc/passwd`, `/etc/security/passwd`, and `/etc/groups` files up-to-date manually.

5.3.4 CMVC Server and Clients

The host, *bering*, was configured as the CMVC server on the network. CMVC client software was installed and configured on *bengal*, *yellow*, and *bering*.

5.3.5 DB2/6000 Server and Clients

DB2/6000 Client and DB2/6000 Server were installed on *bengal* to support the initial application migration to AIX. Access to DB2/6000 on *bengal* was through the COBOL API. The DB2/6000 Client and DB2/6000 Server were installed on *yellow* to support the object-oriented reimplementation of the application. Access to DB2/6000 on *yellow* was through the DB2 CLI for C.

5.3.6 Compiler Compute Servers

Micro Focus COBOL 3.1 and Micro Focus COBOL ToolBox 3.1 were installed exclusively on *bengal*. The XL C^{++} compiler was installed exclusively on *yellow* and the XL C compiler was installed on all systems, but most C source code compilation was run on *bering*. This distribution of the compiler mimicked what would be typical of a mixed-vendor environment where the compiler capable of generating executable code for the HP platform would be installed only on HP computers and the compiler capable of generating executable code for Sun platforms would be installed only on the Sun computers. Even if compilers were more widely distributed, it would not be uncommon to restrict use of certain compilers to specific hosts in network load-balancing strategies.

5.3.7 AIC Compute Servers

AIC version 1.1.1 was installed only on *bering*. AIX/6000 Version 3.2.3-extended was the prerequisite operating system level for this AIC version. AIC Version 1.2 was installed exclusively on *yellow* because AIX/6000 Version 3.2.4 was a prerequisite for that version. This would be fairly typical of a mixed environment where operating systems are upgraded gradually according to business needs.

5.3.8 SDE WorkBench/6000

SDE WorkBench/6000 was installed only on *bering*, *bengal*, and *yellow*. The developer using *sargasso* run X11R4 from that host, but run SDE WorkBench/6000 and other tools remotely on *bering* or *yellow*. Developers on each system had occasion to remotely run various integrated development tools on every system but *sargasso*. Typically, SDE WorkBench/6000 would be installed on every host in the network that could support it.

5.4 File System Topology

The file systems on each host were arranged specifically to support our application development environment and played a key role with SDE WorkBench/6000 and CMVC. A common development file tree and subsidiary file trees were established to provide a working area for application baselines under development and hold formal releases of the application. Several file systems were created specifically for CMVC and ORACLE. File systems were mounted across the network to provide the illusion of a network-wide *single system image* for our developers. We ensured that each user had a *home directory* on only one host, although the user might log in to any host in the network. (The home directory is where the user's files are placed by default, if no other location is specified when the files are created.) Various other directories and file systems that were used by the development tools on specific hosts were also made accessible to users on all hosts across the network by means of NFS mounts.

5.4.1 NFS Mounts for Distributed Data with SDE WorkBench/6000

SDE WorkBench/6000 supports the concept of *distributed data* through NFS and a path naming convention. SDE WorkBench/6000 and *network aware* integrated tools follow the convention of constructing a local path name beginning with `/nfs/` followed by a remote host name, followed by the remote file's absolute path name. When compilers, editors, and other tools are informed by SDE WorkBench/6000 that they need to access a file on a remote host, they can construct a local path name to access that file, if the proper NFS mounts are made. The user identifies the remote host and the file path name by setting the *data context* in SDE WorkBench/6000.

A common development file tree was created for this project. It was placed in a separate file system mounted at `/ad`, on *bering*. It was remotely mounted from there on every other host at the mount point `/nfs/bering/ad`. Using this mount point ensured that SDE WorkBench/6000, which was executing on remote hosts, could access the */ad* file system on *bering*. This way, we were sure that all developers could work on common files from any host on which they run SDE WorkBench/6000.

Refer to *Installing IBM AIX SDE WorkBench/6000 and IBM AIX SDE Integrator/6000* for instructions on supporting distributed data. Figure 10 on page 47 illustrates the NFS mounts used to support distributed data with SDE WorkBench/6000 for this project.

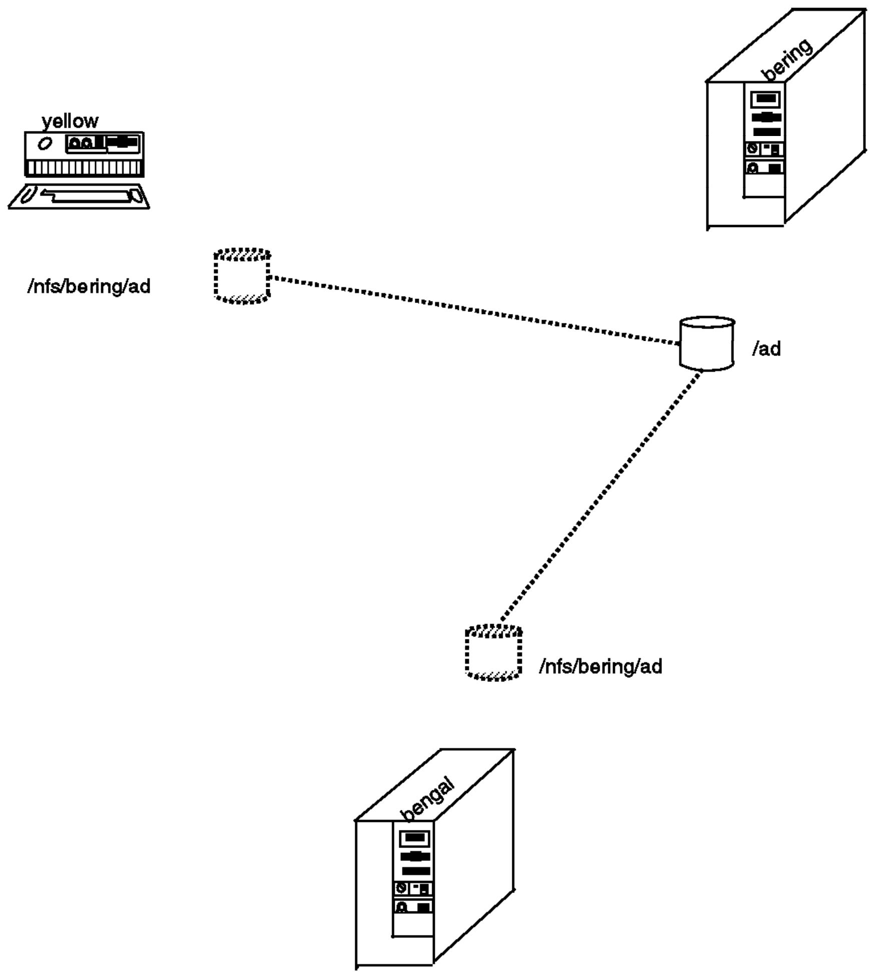


Figure 10. NFS Mounts to Support Distributed Data with SDE WorkBench/6000

5.4.2 NFS Mounts for Distributed Execution with SDE WorkBench/6000

SDE WorkBench/6000 defines the concept of *distributed execution* as what happens when a locally executing SDE WorkBench/6000 is requested by a user to start a tool's execution on a remote host. If the remote system supports distributed execution, it must export the `/tmp` directory to the local system. The local host must then mount it following the distributed data path naming convention for the mount point. For example, if a user on the *bering* host wants to run the C^{++} compiler on *yellow* from SDE WorkBench/6000, the `/tmp` file system from *yellow* must be mounted on *bering* at `/nfs/yellow/tmp`.

Refer to *Installing IBM AIX SDE WorkBench/6000 and IBM AIX SDE Integrator/6000* for instructions on supporting distributed execution. Figure 11 on page 48 illustrates the NFS mounts we made to enable every host that supported SDE WorkBench/6000 to initiate remote execution on any other host that also supported SDE WorkBench/6000.

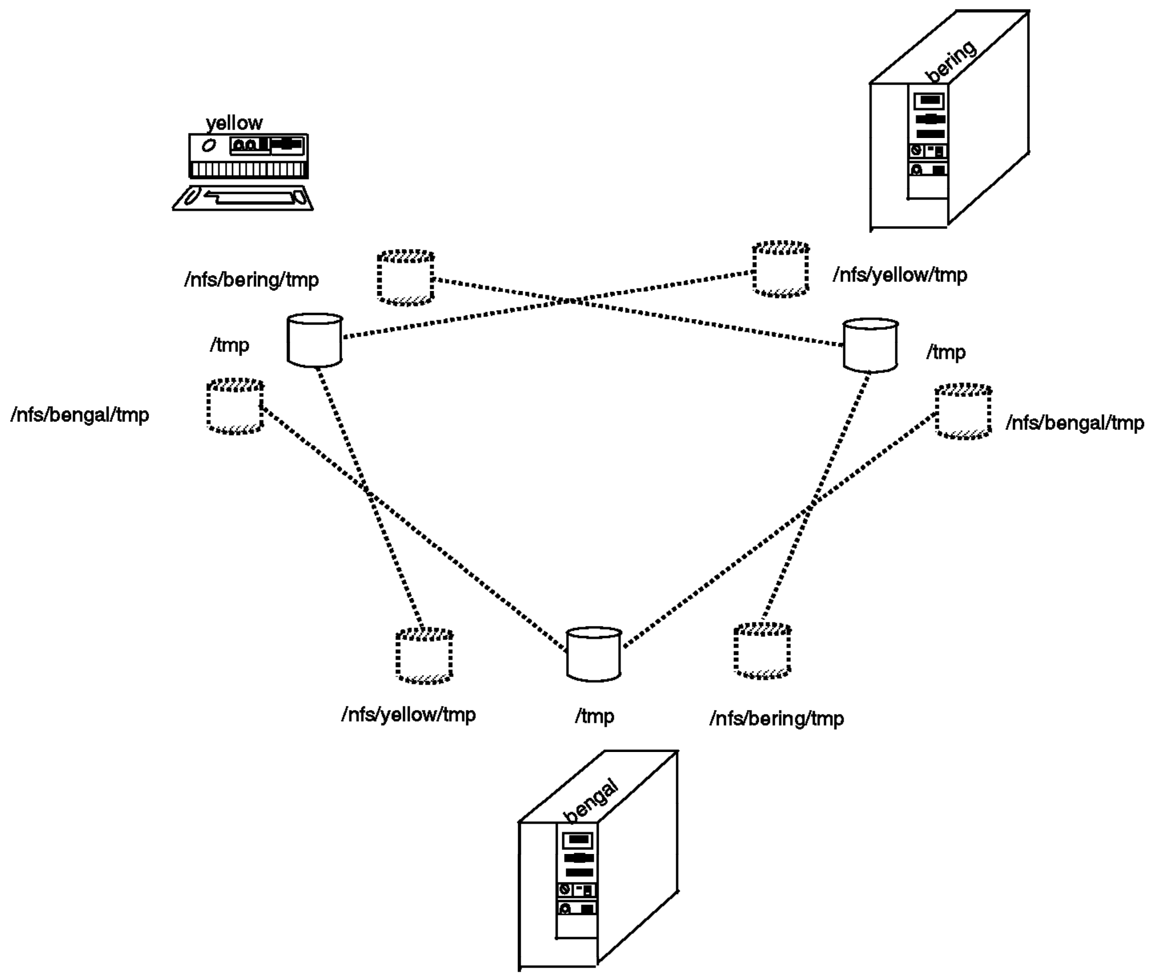


Figure 11. NFS Mounts to Support Distributed Execution with SDE WorkBench/6000

5.4.3 NFS Mounts for Single System Image

For every user able to log in to a host, that host records a specific home directory. Typically, that directory is in a local file system named `/home` or `/u/`. When a user can log in to multiple hosts in a network, the user can have multiple home directories, one on each host. This can cause confusion because the user can forget which home directory contains specific files or can inadvertently create multiple versions of files when there should be only one. Also, because many tools and utilities require users to tailor specific files located in their home directory, maintaining identical copies of those files on multiple hosts can be a lot of work.

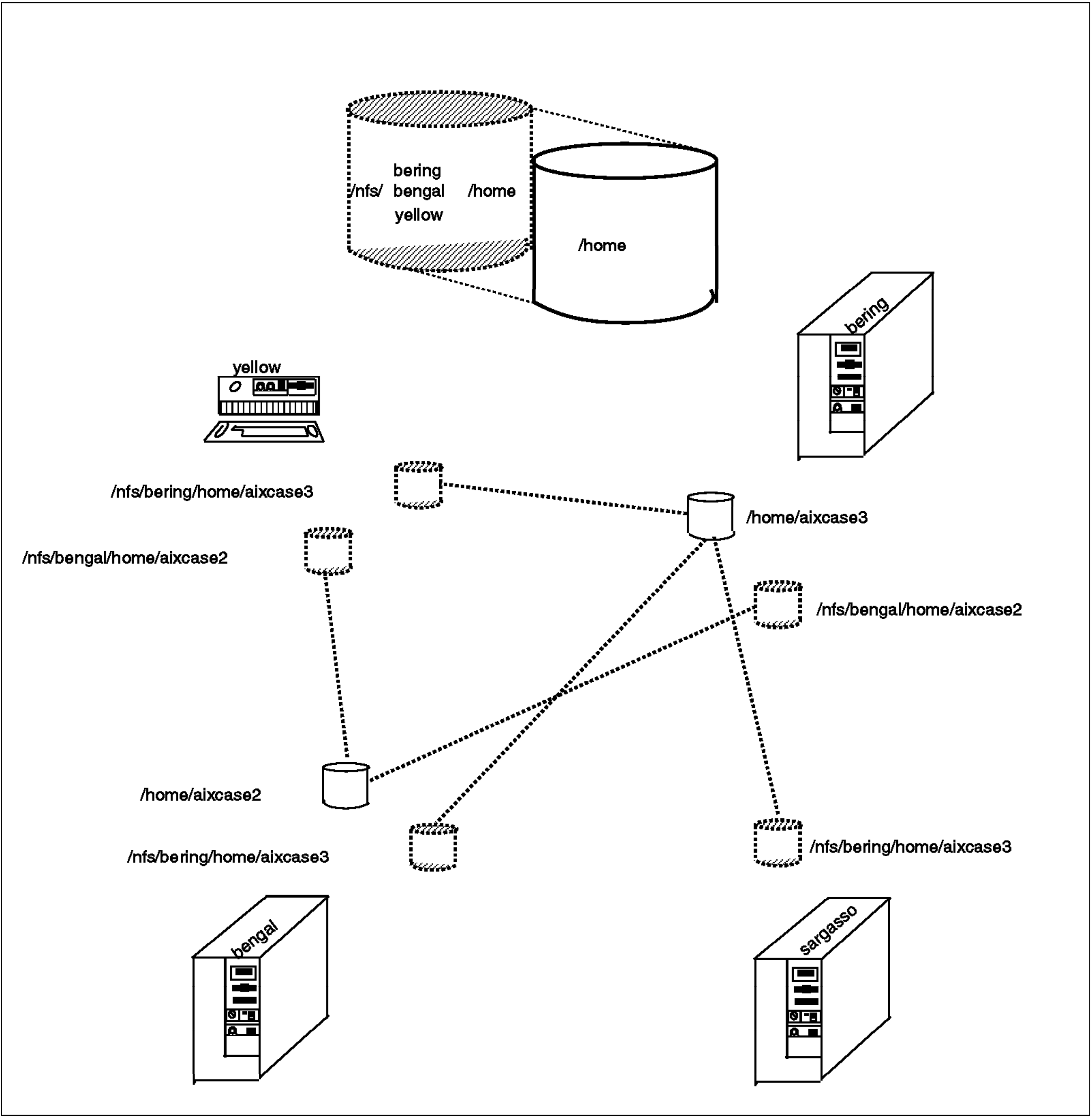
While there may be other ways to deal with this problem, the concept of a *single system image* is quite popular as a solution. This ensures that no matter where users are logged in, they have only one home directory located in a file system on one host. To accomplish this, the */home* file system from a single host is mounted using NFS on all the other remote hosts in the network. The mount point for the */home* file system is not the same on the remote hosts as it is on the local host. For example, it might be `/mnt/hostX/u` on one remote system and `/nfs/hostX/home` on another remote host. So that the home directory is properly configured when users log in to remote hosts, they must record the correct path name of their remotely mounted home directory in the `/etc/passwd` file on each remote host. In the simplest scenario, all users on all hosts would store their files on the */home* file system of a single host. This can cause unwarranted network traffic, if some users are likely to log in to one host frequently, while others are likely to log in to another host most often. A more complex scenario, therefore, is to have users establish their single home directory on the host they use most often.

We created a single system image by following the SDE WorkBench/6000 convention for naming mount points of remote file systems. For the host at which each user normally logged in, the default home directory was set to `/home/username`, and that is where the home directory was really located. On remote hosts, the default home directory was set to `/nfs/hostname/home/username`. The local */home* file system was then mounted on all remote systems at that mount point.

For example, *aixcase2* normally logged in at *bengal* and had a default home directory of `/home/aixcase2` on that host, but on *bering* or *yellow* had a default home directory of `/nfs/bengal/home/aixcase2`. We mounted the */home* file system from *bengal* at `/nfs/bengal/home` on *yellow* and *bering*.

We also had another situation to support, which was similar to having a diskless workstation. Disk space on *sargasso* was so limited that we felt it could not support the home directory for *aixcase3*, the user who normally logged in at that host. To resolve this situation, we decided that *aixcase3* should have a home directory in the */home* file system on *bering*. We mounted the */home* file system from *bering* on *sargasso* at `/nfs/bering/home` and set the default home directory to that path for *aixcase3* on *sargasso*. We mounted the `/home` file system from *bering* on each of these hosts and set the default home directory path identically on those hosts so *aixcase3* could have a common home directory *yellow* and *bengal*, The default home directory for *aixcase3* on *bering* was set to `/home/aixcase3`.

An illustration showing all the NFS mounts supporting all the users would be too complex, so the NFS mounts supporting a single system image for only the users *aixcase2* and *aixcase3* are illustrated in Figure 12 on page 50.



bering
/nfs/
bengal
/home
yellow
/home
bering
yellow
/nfs/bering/home/aixcase3
/home/aixcase3
/nfs/bengal/home/aixcase2
/nfs/bengal/home/aixcase2
/home/aixcase2
/nfs/bering/home/aixcase3
/nfs/bering/home/aixcase3
bengal
sargasso

Figure 12. NFS Mounts to Support a Single System Image

5.4.4 NFS Mounts for AIC 1.2

We found that AIC 1.2 had a restriction with respect to file path names when it interacted with other SDE WorkBench/6000 tools and Execution Manager. Although it was network aware, it was not *network scope*; so it was unable to interpret remote file path names or path names that began with the SDE WorkBench/6000 convention, `/nfs/hostname`. This is discussed in detail in 9.4.6, "Integration Restrictions and Their Circumventions" on page 220.

The files of concern were all in the common development file tree that was mounted at `/ad` and in the home directory of *aixcase3* on *bering*. To circumvent the problems caused by this restriction, path names that did not begin with /nfs were created on *yellow*, using the AIX command **link**.

Specifically, a link was created on *yellow* from /home/aixcase3 to /nfs/bering/home/aixcase3 so the user could refer to files on the remote system by a path name that AIC 1.2 interpreted to point to a local file. Another SDE WorkBench/6000 tool, such as Program Editor, might have created that same file using the local path name on /nfs/bering/home/aixcase3, but the user was able to tell AIC 1.2 to look for it with the path /home/aixcase3.

In addition, a link was created on *yellow* from /ad to /nfs/bering/ad. This enabled AIC 1.2 to find the files that other SDE WorkBench/6000 tools had manipulated with their data context including *bering* as a host and /ad at the start of the path name. AIC 1.2 could find these files if its current directory was set to the corresponding *local* path name beginning /ad.

Figure 13 illustrates how the local file links masked the NFS mount implied by the path names beginning with /nfs/hostname.

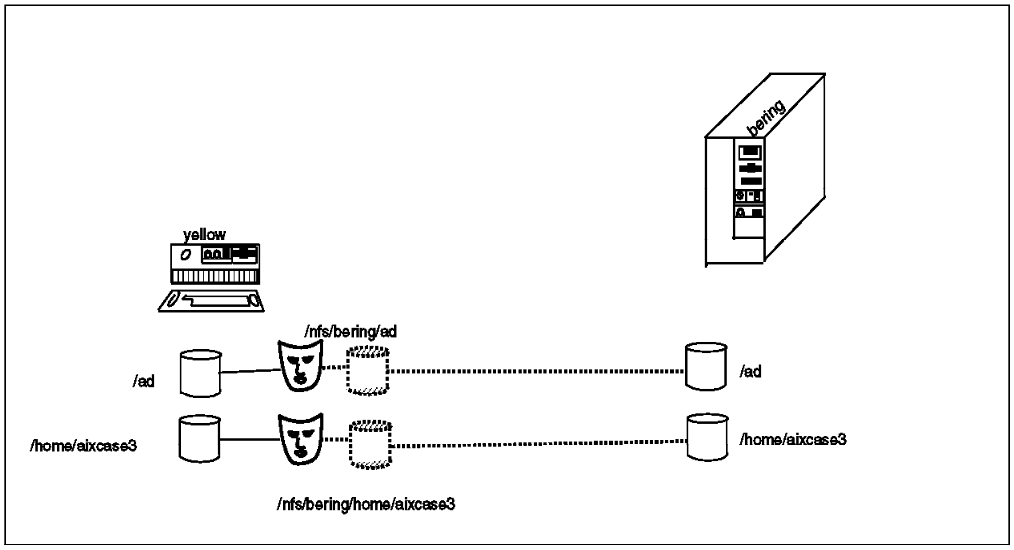


Figure 13. File Systems Cross-Mounted on Our Hosts

5.4.5 Common Development File Tree

As mentioned earlier, our common development file tree was placed in a file system that was mounted directly at /ad on *bering*, but mounted indirectly (using NFS) on every other host at /nfs/bering/ad.

The /ad directory contained a directory for each of two applications: *ProductA* (the application being downsized), and *ProductB* (an imaginary second application). These directories held file trees for the *production releases* of these two applications. Each of these production release file trees contained the source, compilation instructions, executables files, and all other files associated with the release that it represented. The production releases were named according to the target platform and numbered to identify their sequence. For example, the production release file tree springing from the /ad/ProductA/MVS_Release_0 directory contained the files comprising the original MVS release, before the migration. The production release file tree springing from the

/ad/ProductA/MVS_Release_1 directory contained a version of this application modified to support common code between the MVS and AIX releases. These production release file trees are illustrated in Figure 18 on page 59.

The /ad directory contained other directories, which contained the prototype development file trees for *ProductA* and *ProductB*. They were named /ad/projectA_proto and /ad/projectB_proto. Below /ad/projectA_proto were several layers of subdirectories that were organized according to source code languages and the types of data contained in the various files.

When a source file was checked in the CMVC library, the relative path name leading to that file in the prototype development file tree was also stored. When a particular version of that file was later extracted from the library to the production release file tree, CMVC used that same relative path name to place it in that file tree. Therefore, there are similarities in the file tree organizations below /ad/projectA_proto and the various release directories in /ad/ProductA.

5.4.5.1 ProjectA Prototype Development File Tree

The *projectA_proto* prototype development file tree is shown in Figure 14 on page 53. It contained the following directories:

Directory	**Description**
bin	Contained executable files (binary data files) of the application and any related test tools.
catalog	Contained any message catalogs used by the application. Message catalogs enable the separation of user message texts from the application code and the retrieval at execution time of message texts in the appropriate language of the user executing the application.
db2	Contained all data, command scripts, and utilities related to the DB2/6000 data base.
include	This directory contained any C language include files (filename.h) associated with the application's C language source modules.
resource	Contained *X11 resource files* for the individual OSF/Motif widgets that are generated by AIC for each AIC interface file. All *language-dependent* widget resources, such as label strings, dialog titles, and mnemonics, can be set by means of these resource files. This mechanism isolates all text strings appearing in the GUI from the GUI code itself and enables these test strings to be translated into multiple languages with our recompiling the code. Separate X11 resource files can be created containing the translation of these text strings into each target language. When our application runs, the X Windows System will identify the correct X11 resource file to use according to values in certain shell variables. These variables are set by each user.
source	Contained source code for the application in appropriate subdirectories.
test	Contained test code and data.

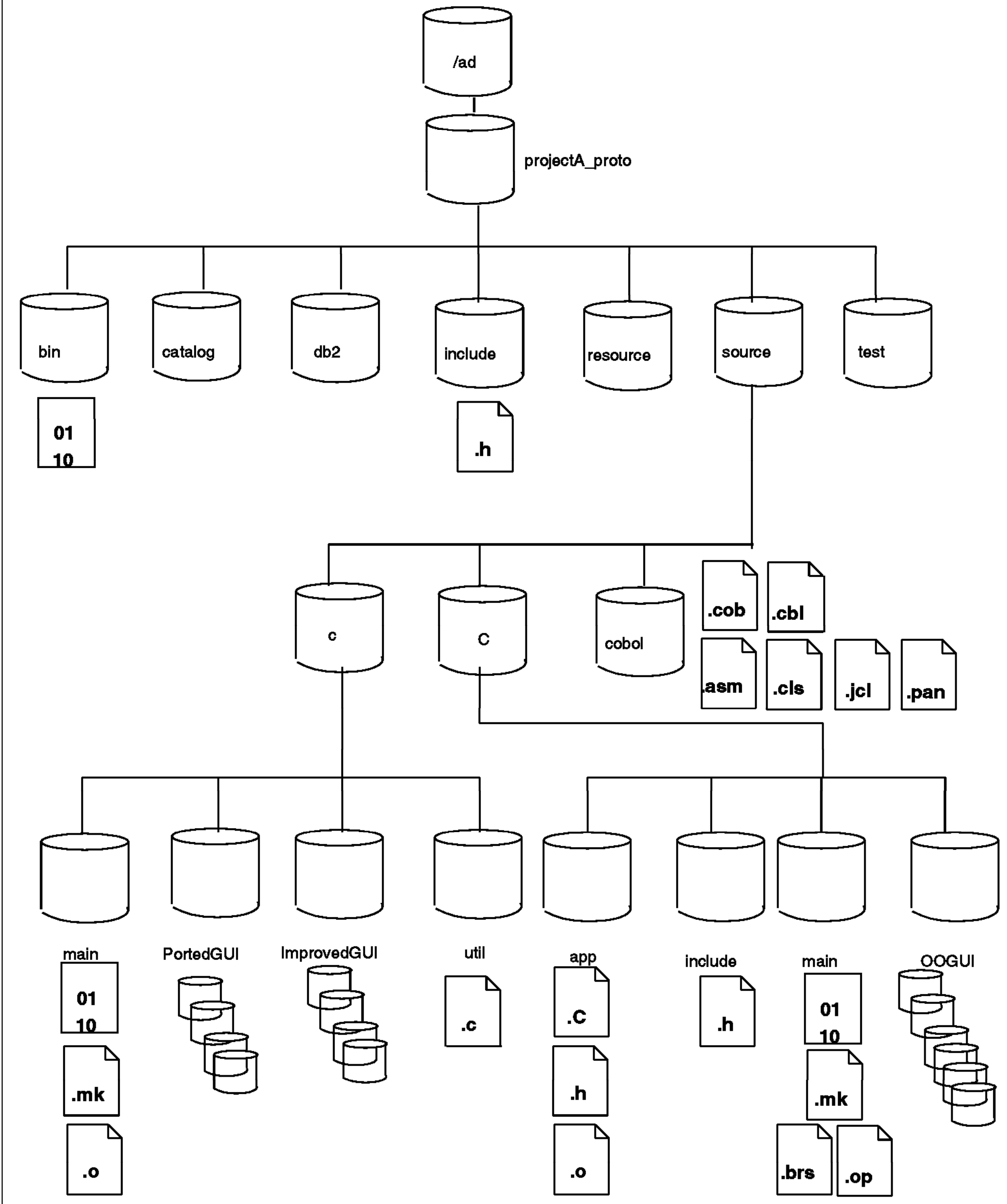


Figure 14. ProjectA Prototype Development File Tree

5.4.5.2 The Source Directory

The `source` directory contained a subdirectory for each of the three languages:

Directory	Description
`c`	Contained source modules in the C language
`C`	Contained source modules in the C^{++} language
`cobol`	Contained source modules in the COBOL language

The C Language Source Directory: This directory was named `c` because C language file names end with the *.c* extension. It contained a directory for the main routine of the application, two directories for the GUI source (*PortedGUI* and *ImprovedGUI*), and a directory for utility subroutines. Figure 15 on page 55 illustrates the *PortedGUI* directory and Figure 16 on page 56 illustrates the *ImprovedGUI* directory. The C language source directory contained:

Directory	Description
`main`	Contained the main module that invoked the GUI code.
`PortedGUI`	Contained the AIC interface and callback source modules for the AIX_Release_1 GUI (as first migrated to AIX). It also contained a subdirectory for each window of the GUI and the source for the interface file and callback subroutines.
`ImprovedGUI`	Contained the AIC interface and callback source modules for the AIX_Release_2 GUI (as made more CUA or UNIX-like). It also contained a subdirectory for each window of the GUI and the source for the interface file and callback subroutines.
`util`	Contained the utility subroutines that replaced the assembler code.

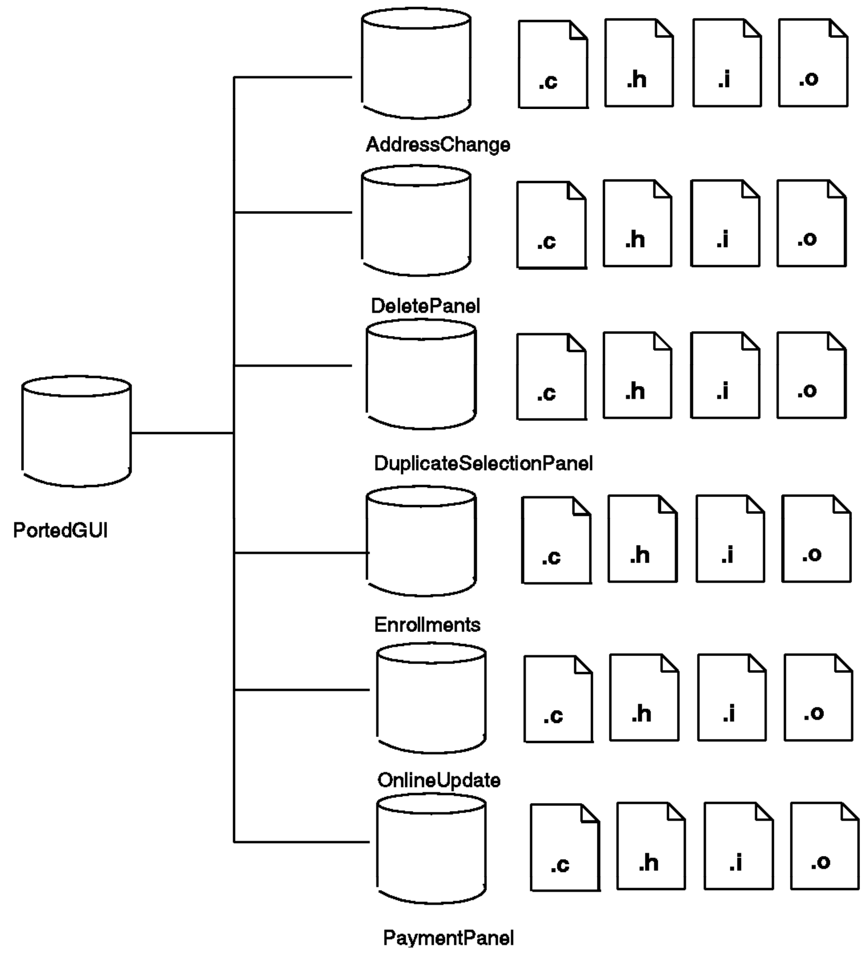


Figure 15. PortedGUI Directory

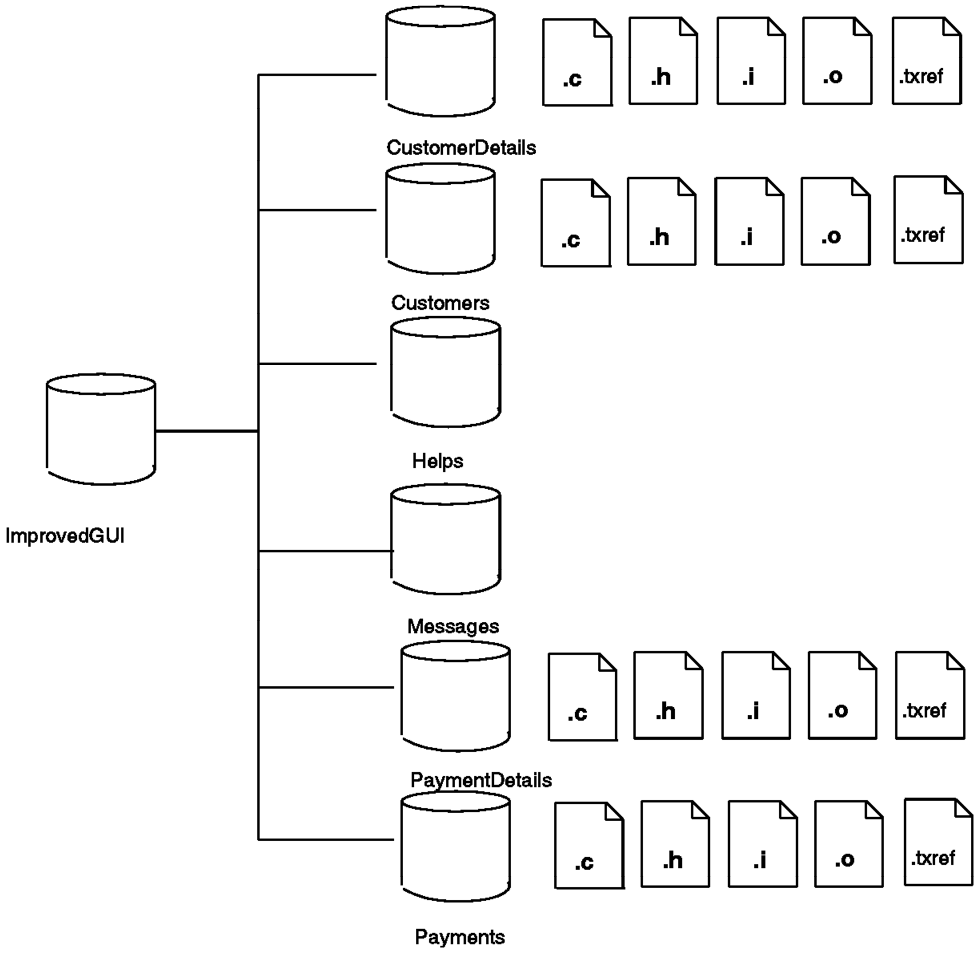


Figure 16. ImprovedGUI Directory

The C++ Source Directory: The C++ directory was named `C` because C++ source code file names end with the *.C* extension. It contained source code for the object-oriented implementation and two subdirectories, one for the user interface and one for the application itself. Figure 17 on page 57 illustrates this directory. The C++ source language directory contained these:

Directory	**Description**
`OOGUI`	Contained the AIC source modules. There were no callback source files, because the newer version of AIC required the callbacks to be incorporated in the interface file in order to generate correct C++ source files. It also contained a

subdirectory for each window of the GUI. The source code for these windows is stored in these subdirectories.

app	Contained the application C^{++} source modules that defined its classes and methods, and included C language modules that were necessary for access to the DB2 CLI for C.

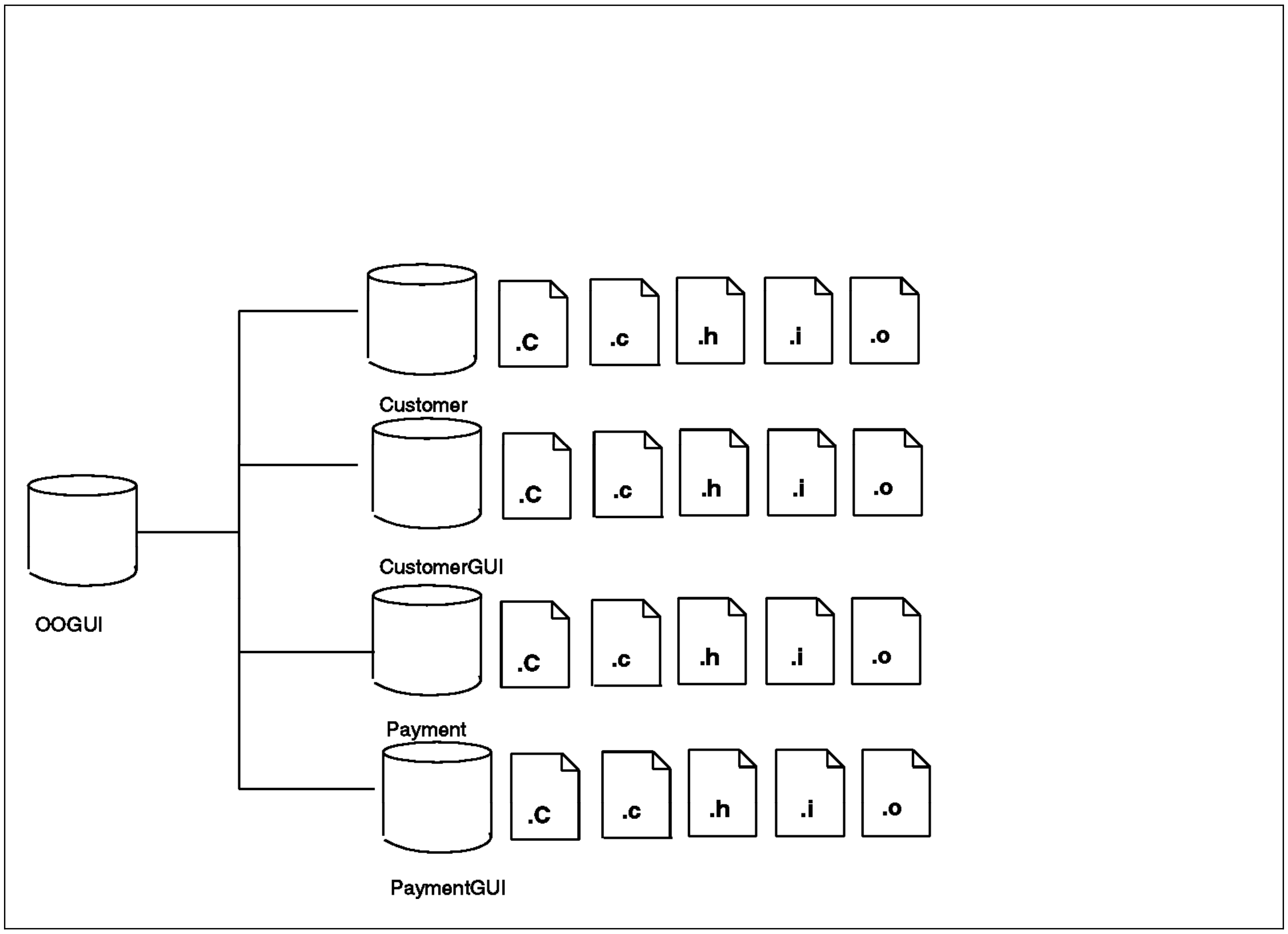


Figure 17. OOGUI Directory

The COBOL Source Directory: The *cobol* directory contained the COBOL source code and all other source code necessary for the MVS build. The MVS releases were not actually built on AIX; so there seemed little added value in placing the various files in subdirectories based on language or file type. Files placed here included MVS COBOL source and copy code modules, ISPF panel definition source modules, JCL scripts, CLIST files, and assembler language source modules.

Note that some of these subdirectory names recurred in the directories containing the production releases, but not all appeared beneath any one production release's directory. For example, the AIX and MVS releases might have had directories containing COBOL source, but the MVS releases would not have had directories containing C language source code. Likewise, the object-oriented release did not require a subdirectory for either the C or COBOL source code language modules. However, all releases had a directory for the database, message catalogs, and binaries.

5.4.5.3 The ProductA Production Release Directories

As mentioned earlier, /ad/ProductA contained a directory for each production release. Partial contents of this directory are shown in Figure 18 on page 59. It contained:

Directory	Description
MVS_Release_0	Contained all the files required to build the original MVS application. /ad/ProductA/MVS_Release_0 contained a file tree whose subdirectories were a subset of those contained in /ad/projectA_proto. For example, it contained a source directory and below that a cobol directory, which contained the original COBOL source, assembler, CLIST, and JCL files. However, it did not contain a c or C directory because no files were stored in the CMVC library with those path names.
MVS_Release_1	Contained all the files required to build the revised MVS application. /ad/ProductA/MVS_Release_1 contained a file tree similar to that of MVS_Release_0.
AIX_Release_1	Contained all the files required to build the initial AIX release; the minimal-effort migration. /ad/ProductA/AIX_Release_1 also contained a file tree whose subdirectories were a subset of those contained in /ad/projectA_proto. It contained directories for COBOL and C, but not for C++. Furthermore, it did not contain ImprovedGUI below the c directory. Instead, it contained only the PortedGUI directory.
AIX_Release_2	Contained all the files required to build the AIX release with the improved, or object-oriented, GUI. /ad/ProductA/AIX_Release_2 also contained a file tree whose subdirectories were a subset of those contained in /ad/projectA_proto. It contained directories for COBOL and C, but not for C++. Furthermore, it did not contain PortedGUI below the c directory; instead, it contained only the ImprovedGUI directory.
OO_Version_1	Contained all the files required to build the initial object-oriented AIX release. (Admittedly, it was poorly named). /ad/ProductA/OO_Version_1 contained a file tree whose subdirectories were a subset of those contained in /ad/projectA_proto. It contained a directory for C++, but not for C or COBOL.

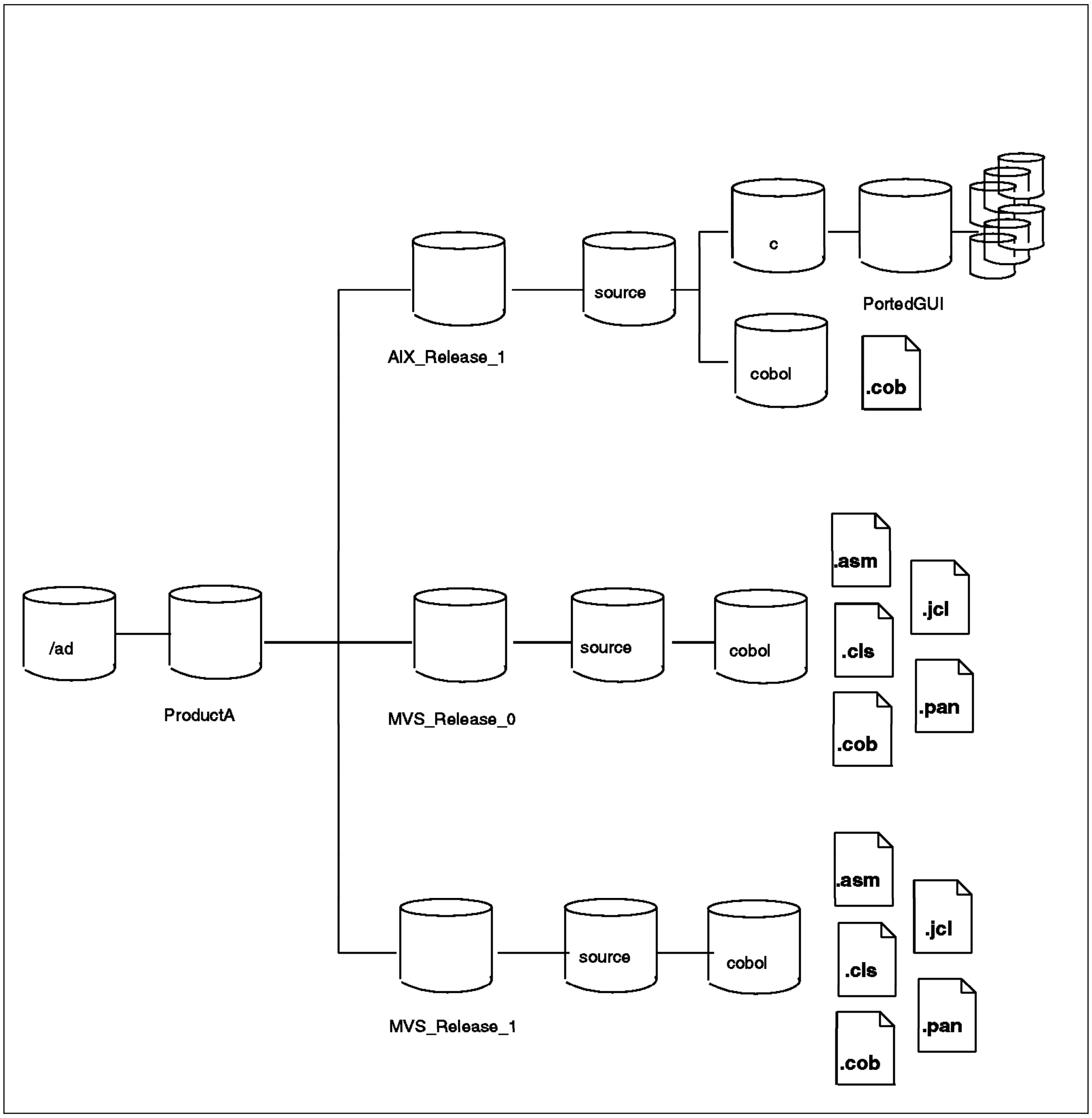


Figure 18. ProductA Production Release File Trees

Chapter 6. Migrating the Legacy Application to AIX

This chapter describes the minimal-effort migration phase of our project. It details how we used AIX AD tools to design and implement the GUI, port the COBOL to AIX, migrate the database to AIX, and integrate the COBOL and GUI source code.

In this and subsequent chapters, we show how we used the AIX AD products while designing, programming, and testing the project. We describe specific menus, buttons, and other features of the user interfaces of the various AIX AD tools we used. We also show many screen captures, and describe exact commands, menu selections, and other user interactions with these tools. The intent is that the reader should get a good feel not only for what these tools do, but also for how using these tools and SDE WorkBench/6000 can enhance the developer's productivity and efficiency. The reader should glean both a general understanding of how these tools are used, and the specifics of using the tools in our particular environment to accomplish our specific goals.

6.1 Using SDE WorkBench/6000 and Integrated Tools to Develop the GUI

The tools our GUI developer used during this initial phase of the project were:

- Tool Manager
- Program Editor
- Program Builder
- Development Manager
- Program Debugger
- AIXwindows Interface Composer/6000 v.1.1.1.

This section introduces some of these tools. How our developer used these tools is described in this and subsequent sections.

All of these tools but AIC are integral elements of SDE WorkBench/6000. We integrated AIC 1.1.1 with SDE WorkBench/6000 before we began to develop the GUI. The integration effort is described in 9.4, “Tailoring SDE WorkBench/6000 for AIC Programmers” on page 212.

6.1.1 Remote Access to SDE WorkBench/6000 and Integrated Tools

Our GUI developer did his work from the system console at the host *sargasso* as the *aixcase3* user. As mentioned earlier, we decided not to install SDE WorkBench/6000 or AIC 1.1.1 on *sargasso*. Instead, our GUI developer needed to run SDE WorkBench/6000 remotely on the host *bering*.

To invoke SDE WorkBench/6000 from the console of a system on which it is installed, the user simply starts X Windows, and types in the command **workbench** in a terminal emulation window. Access to SDE WorkBench/6000 from a remote system on the network required a few preliminary steps.

The first step was to enable the remote AIXwindows client programs to have their input and output handled by the local AIXwindows server on *sargasso*. To do this we added the remote system's host name, *bering*, to the `/etc/X0.host` file on *sargasso*. An alternative to this would be for *aixcase3* to run the **xhost** command line shown in Figure 19 every time AIXwindows was started up.

```
xhost +bering
```

Figure 19. Using the xhost Command to Authorize Remote System Access

The next step was necessary because SDE WorkBench/6000 requires that `/usr/softbench/bin` precede the normal system directories, `/usr/bin` and `/bin`, when the value of the `PATH` environment variable is set. (This requirement is explained in *Installing IBM AIX SDE WorkBench/6000 and IBM AIX SDE Integrator/6000*.) To ensure that the `PATH` variable would always be set correctly before SDE WorkBench/6000 was invoked by our developer, we modified the `.profile` file for *aixcase3*'s login name on the *bering* host. This file is processed by the Korn shell whenever the programmer logs in, either from a direct attach device or remotely over the network. Figure 20 shows the line in the `.profile` file which sets the `PATH` variable for our SDE WorkBench/6000 users. Note that this is one continuous line in the file, not two lines as shown.

```
PATH=/usr/softbench/bin:/usr/bin:/etc:/usr/sbin:usr/ucb:$HOME/bin:
    /usr/bin/X11:/sbin:/usr/lpp/cmvc/bin:.
```

Figure 20. PATH Setting in User's .profile File

From a terminal emulation window on the *sargasso* console, our programmer now remotely logged in to the *bering* host and run SDE WorkBench/6000, directing it to display output on the Xserver running on *sargasso*, with the command shown in Figure 21.

```
workbench -display sargasso:0.0
```

Figure 21. Executing SDE WorkBench/6000 for Remote Display of its Output

Since our developer intended always to log in to *bering* remotely from *sargasso*, our developer decided to update the `.profile` file so the `DISPLAY` variable was always pre-set to `sargasso:0.0`. This was done by adding to this file the statement shown in Figure 22.

```
export DISPLAY=sargasso:0.0
```

Figure 22. Setting DISPLAY Variable in .profile File

Instead of using **rlogin** our programmer could also have used the **rexec** command to remotely run SDE WorkBench/6000 on *bering*, while directing SDE WorkBench/6000 to display output at the Xserver instance executing on *sargasso*. The `PATH` and `DISPLAY` variables would still need to be set correctly, however. Since no login shell would be spawned, the `.profile` file must be explicitly processed by the same shell instance spawned to remotely run the **workbench** command. This was accomplished by forcing that shell to run the `.profile` file before executing the **workbench** command. Figure 23 on page 63 shows the **rexec** command line as it would have been entered on *sargasso*.

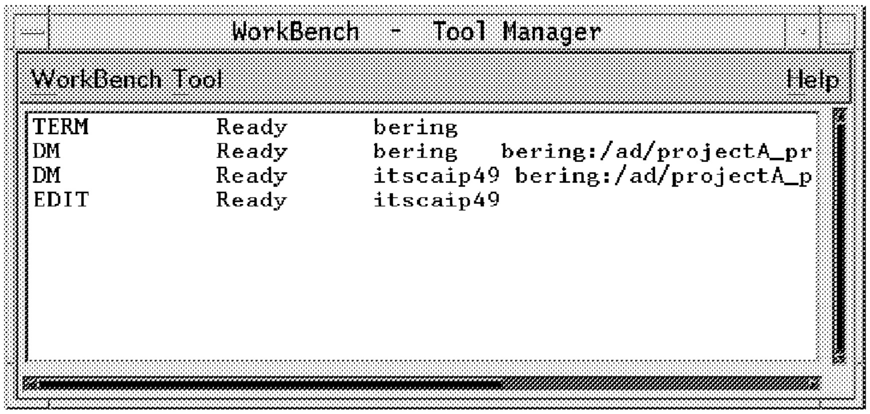
```
rexec bering ". /home/aixcase3/.profile; workbench -display sargasso:0"
```

Figure 23. Invoking SDE WorkBench/6000 on a Remote Host through rexec

Now that SDE WorkBench/6000 was running on the remote host and being displayed on the local host, any tool started up using SDE WorkBench/6000 likewise displayed its output on the local host.

6.1.2 Tool Manager

The first thing our developer saw after issuing the **workbench** command was the WorkBench Tool Manager window. Tool Manager is one of the fundamental components of SDE WorkBench/6000. It starts up automatically when the **workbench** command is issued. The WorkBench Tool Manager window lists the status and identity of all tools currently running on behalf of the user (on all hosts). Figure 24, for example, shows the WorkBench Tool Manager window listing four tool instances running on two different hosts.



```
TERM        Ready      bering
DM          Ready      bering    bering:/ad/projectA_pr
DM          Ready      itscaip49 bering:/ad/projectA_p
EDIT        Ready      itscaip49
```

Figure 24. WorkBench Tool Manager Window Lists Status of Running Tools

From Tool pull-down of the WorkBench Tool Manager window, the user starts and stops developer tools that have been integrated with SDE WorkBench/6000. The user can also set the execution host and/or data context for SDE WorkBench/6000 tools from this pull-down, and iconify or de-iconify a running tool. From the SDE WorkBench/6000 pull-down, the user can take several actions including:

- Define a set of tools to be invoked automatically when SDE WorkBench/6000 is started up.
- Start up a set of tools previously defined.
- Quit SDE WorkBench/6000 (stopping running tools).

- Iconify all tools at once.

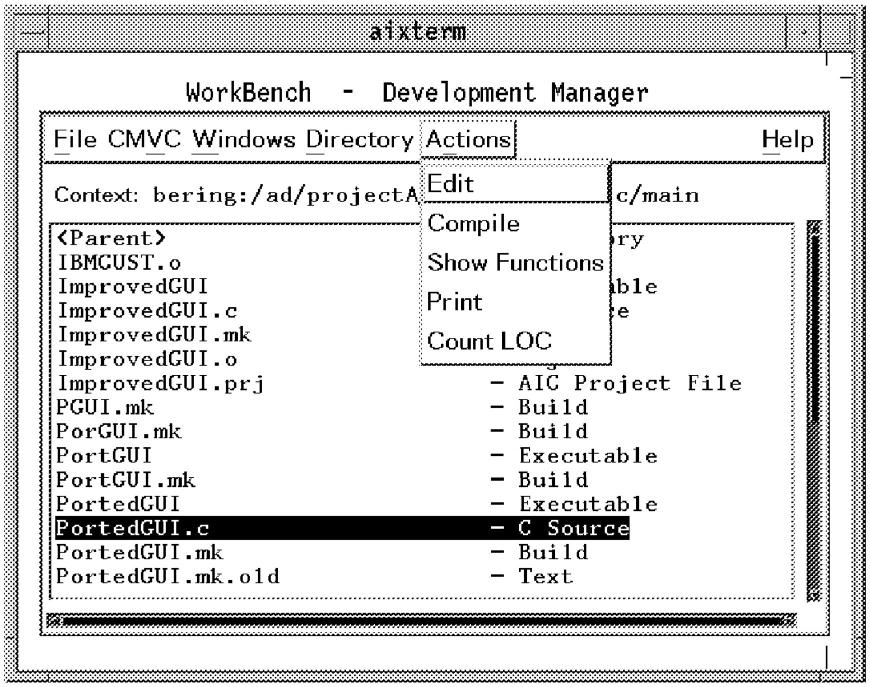
Tool Manager is discussed in greater detail in later sections as our developer uses it to start, stop, and otherwise control AIX application development tools that are integrated with SDE WorkBench/6000.

6.1.3 Using Development Manager to Manage Files and Directories

The next thing our developer saw was the WorkBench Development Manager window. Development Manager typically starts up automatically when a user issues the **workbench** command. Development Manager is a fundamental element of the SDE WorkBench/6000 product that provides the user access to files, directories, and commands. It replaces the traditional command line mode of user interaction with a graphical, point and click mechanism.

With the typical UNIX command line user interface, the user types a command followed by various flags and parameters that indicate input files, output files, and other options. To indicate the end of the command, the user presses the Enter key. The output of the command is then displayed, if there is any. Nothing else is displayed by default. For example, if the user wants to see a listing of the current directory, the command **ls** can be entered. A snapshot of the directory contents is presented. This listing would not be updated as a result of subsequent user actions. If the user wants to copy a file into another file, the user types in a command line like `cp sourcefile targetfile`, naming the source and target file names as parameters to the copy command. The user takes it on faith that this occurred, or requests that the directory listing be regenerated to confirm that faith.

Development Manager presents a very different pattern of user interaction. It always presents the user with a directory contents list in a window. Each item in the directory is shown with its name, an optional write-only symbol (-), and the file type. This window includes a menu bar. To run a command, the user typically selects one or more input file names from the contents list using a single click of the left mouse button. Next, the user selects a pull-down menu from the menu bar using the same mouse button, causing a list of command options to appear. Finally, the user selects an option from the pull-down menu and the command is run. If the action taken results in a change in the directory contents, Development Manager immediately updates the window. Figure 25 on page 65 shows the WorkBench Development Manager window with a file selected (highlighted) and the Actions pull-down menu active. If the user chose the Compile option, the directory listing would be updated automatically to show any new files created by the compiler.


aixterm
WorkBench - Development Manager
File CMVC Windows Directory Actions Help
Edit
Compile
Show Functions
Print
Count LOC

```
Context: bering:/ad/projectA                c/main
<Parent>                                   ry
IBMCUST.o
ImprovedGUI                                ble
ImprovedGUI.c                              e
ImprovedGUI.mk
ImprovedGUI.o
ImprovedGUI.prj                  - AIC Project File
PGUI.mk                          - Build
PorGUI.mk                        - Build
PortGUI                          - Executable
PortGUI.mk                       - Build
PortedGUI                        - Executable
PortedGUI.c                      - C Source
PortedGUI.mk                     - Build
PortedGUI.mk.old                 - Text
```

Figure 25. WorkBench Development Manager Window with Pull-Down Menu Active

The Actions pull-down menu displays different lists of selections depending on which type of file is highlighted before the menu is pulled down. Lists of possible actions to take are predefined for various types of files. This is a configurable aspect of Development Manager. In 9.4.3, “Modify the Action Pull-Down of Development Manager” on page 216 we discuss how we defined new file object types, and modified the Actions menu selections for them.

Some commands require more information than just the input file name(s). If additional command parameters or flags are necessary, the command presents a dialog box with fields into which the values can be entered. The user signals completion of the command by using the mouse to select the **OK** button presented in the dialog box, or presses the Enter key. Figure 26 on page 66 shows such a dialog box being presented. The user identified the file to be copied by highlighting it prior to selecting the **Copy** option from the File menu, but the copy command also needed to know the name of the file into which the copied data must be placed. This dialog box therefore prompted the user for that file name.

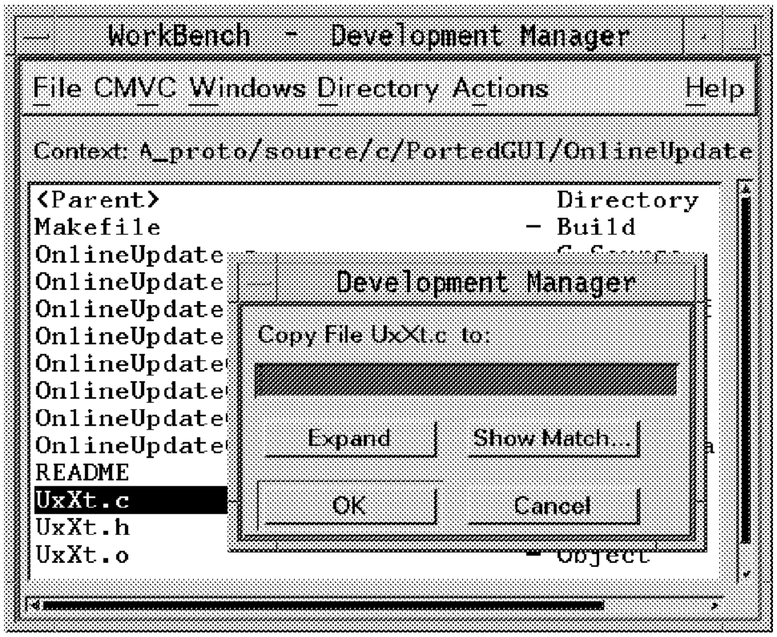


Figure 26. WorkBench Development Manager Window with Dialog Box

The user can also cause the default action for a file to be run by simply double-clicking the left mouse button while the cursor is above the file name. Development Manager uses predefined naming conventions to determine the default action for a given file. The menu bar, pull-down menus, defined naming conventions, and default actions are all tailorable features. Several ways we tailored these features are discussed in Chapter 9, “Tailoring SDE WorkBench/6000, Integrated Tools, and DB2/6000” on page 201.

Our GUI developer used Development Manager as his primary interface to the operating system. The File pull-down menu of the WorkBench Development Manager window enabled the user to set context, filter the directory contents listing, copy, delete rename and change permissions on files. The Directory pull-down enabled the developer to create, delete, rename, and list the contents of directories. The CMVC and Windows pull-downs provided access to CMVC commands and windows. These pull-downs are discussed as they are used in later sections. The *IBM SDE WorkBench/6000 Development Manager: Managing Files and Directories* details all features of this WorkBench tool.

6.1.3.1 Using Development Manager to Set Data Context

The first thing our developer did with Development Manager was to set the Development Manager's data context. All SDE WorkBench/6000 tools have a data context that defines the range of data on which the tool operates. Development Manager data context consists of a host name and a directory path name separated by a colon (:). To set the data context in Development Manager, our developer chose the **Set Context** selection from the File pull-down, and entered the information in the format shown in Figure 27 on page 67. This information represented the host name separated by a colon from a directory path name.

```
bering:/ad/projectA_proto/source/c/PortedGUI
```

Figure 27. Format for Entering a Data Context Value

An alternative method of setting the data context is to press and hold the right mouse button while the cursor is over the Context field of the WorkBench Development Manager window. A pop-up identifies all possible directories that can be changed to, from the current working directory. The first option is **Change Context**; selecting it will cause the listing of all directories in the root directory. The traditional UNIX double dot (..) is used to represent the parent directory; any subdirectories are listed individually. If there are subdirectories below any selection, a simple arrow symbol appears to the right of the directory name. The user continues to hold down the right mouse button while traversing the options presented, and new pop-ups appear when the cursor is above any option that includes subdirectories of its own. Figure 28 shows what our developer saw when using the pop-up windows to change context from `/ad/projectA_proto/source` to `/ad/projectA_proto/source/c/PortedGUI/PaymentPanel`.

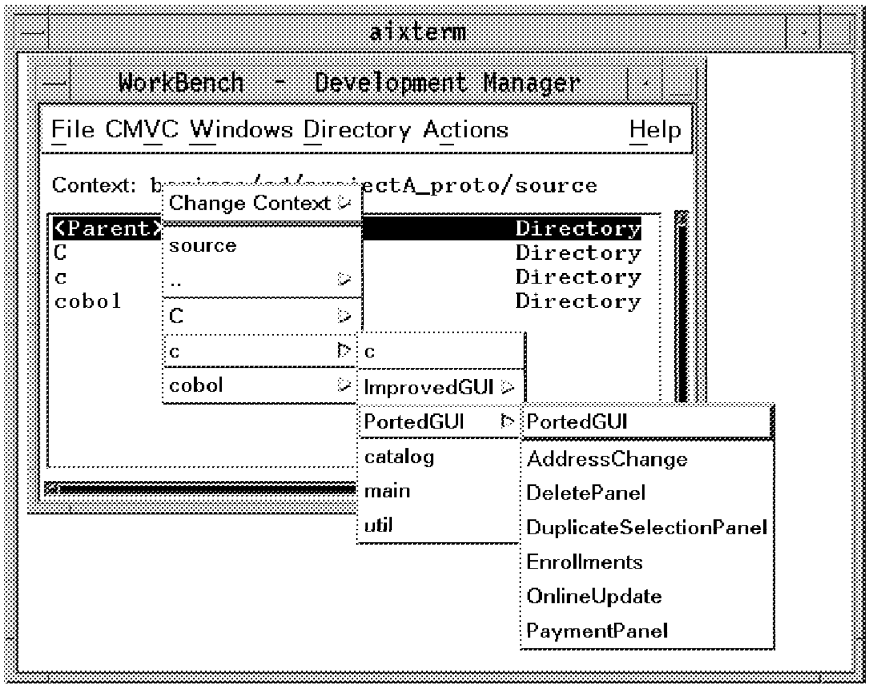
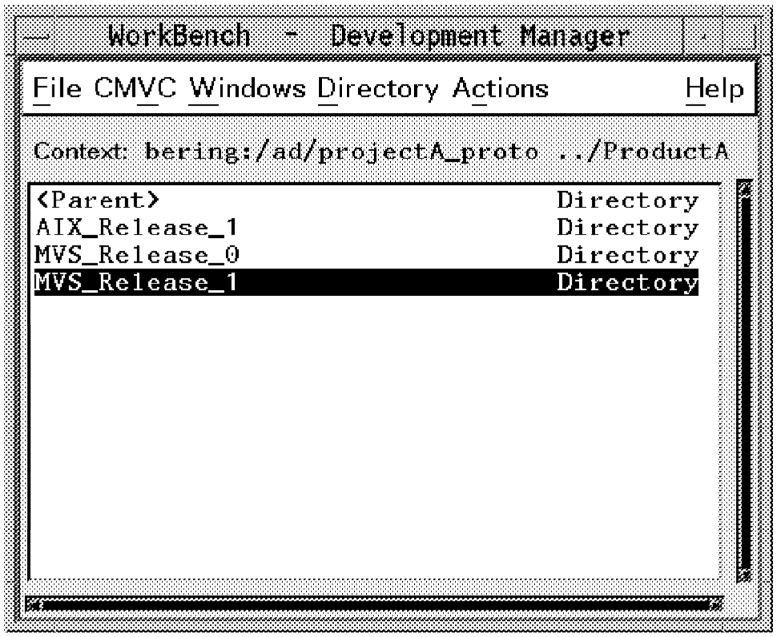


Figure 28. WorkBench Development Manager Window with Context Pop-Ups

6.1.3.2 Using Development Manager to Change the Directory

After setting the context, the contents of the context directory were listed in the window. The contents of other directories above and below that point could be listed by changing the directory. The user can change to any directory on the host specified in the data context. Changing the directory does not change the data context, however.

To change the directory, the user places the cursor over any listed directory, including the <Parent> directory, and double clicks the left mouse button. The context field displays both the context and the relative path from there to the current working directory. The window displays the new directory's contents list. The context is displayed as it was previously, followed by a space, and the relative path from the context directory to the new directory. For example, Figure 29 shows the context set to bering:/ad/projectA_proto while the current directory is set to a directory named ProductA whose parent directory is up one level on the tree from the context directory (indicated by the standard UNIX ../ notation).


WorkBench - Development Manager
File CMVC Windows Directory Actions Help

```
Context: bering:/ad/projectA_proto ../ProductA
<Parent>                          Directory
AIX_Release_1                     Directory
MVS_Release_0                     Directory
MVS_Release_1                     Directory
```

Figure 29. WorkBench Development Manager Window Context vs. Directory

After setting Development Manager context and directory, our user interface developer next created directories to hold the files that would be generated when implementing the GUI.

6.2 Design and Implementation of Our User Interface

This section describes the development approach we took in implementing our new user interface. It explains how we replaced the ISPF panels with windows generated with AIC, and how we tested the GUI with callback stub code. It describes how we incorporated the original COBOL code that accesses the DB2/6000 database with the GUI to complete the migration of the application. Throughout this section we describe how the developers interacted with and made use of SDE WorkBench/6000 tools.

Porting an application from one environment to another raises a couple of questions, especially if the target system supports an improved graphical user interface compared to the one implemented in the existing application. When you port an application from a 3270 panel-oriented dialog manager, such as ISPF, to a graphical user interface system, such as AIXwindows, you can exploit all the features that are not available in an ISPF environment. For example, the ported application could have multiple active windows, use point-and-click techniques, and be based on advanced user interface guidelines, such as common user access (CUA) or the OSF/Motif style guide.

Our goal in this initial migration was to undertake a minimal-effort migration, rather than take advantage of features that are available on AIX. We, therefore, decided to map the original ISPF panels as closely as possible to new GUI windows to reuse as much of the original COBOL code as possible. A user using the GUI would use the keyboard rather than the mouse, and the new windows would not contain advanced features like menu bars and pop-up or pull-down menus, toggle buttons, or other drag-and-drop capabilities. Instead, the new windows were designed to be exact copies of the existing ISPF panels. The only concession to the new platform was that the function keys were replaced by push buttons that could be triggered either by pressing the Enter key after the focus had been moved to the button or by clicking the mouse on the button.

6.2.1 Selecting Directory Organization and Naming Conventions

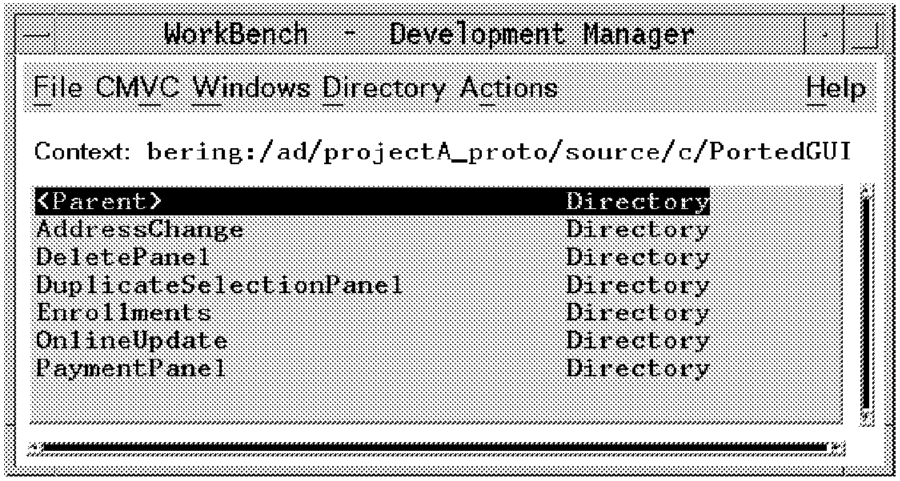
Because several developers worked on this project, it was important to determine clear and reasonable file and directory naming conventions for files created while developing the GUI. Because we were migrating an existing application from one platform to another and would be maintaining parallel versions on both platforms, we also needed naming conventions to help us relate various versions and types of AIX files to the original and modified MVS files. There would not be a one-to-one correspondence of the files on the two platforms that implemented the GUI, because the types of files used to create the ISPF user interface were necessarily different from those required to create an OSF/Motif GUI. Finally, we needed a directory hierarchy that would facilitate finding the various files in the GUI implementation.

6.2.1.1 Directories for the GUI Files

The original ISPF implementation consisted of six ISPF panels. We decided to have one directory for each of the original ISPF panels and named these directories after the ISPF panel headlines (top of screen labels). We decided to place these directories on *bering* below the directory named `/ad/projectA_proto/source/c/PortedGUI`. This enabled any developer on our project to find and access them easily. These six directories, named after the original MVS panels, would be:

- `AddressChange`
- `DeletePanel`
- `DuplicateSelectionPanel`
- `Enrollments`
- `OnlineUpdate`
- `PaymentPanel`.

We used Development Manager to create these new directories. First, we set the context to `bering:/ad/projectA_proto/source/c/PortedGUI`. Then we chose **Create** selection from the Directory pull-down, and filled in the name of the new directory in the dialog box that appeared. The newly created directory was immediately listed in the WorkBench Development Manager window directory contents list. Figure 30 shows the WorkBench Development Manager window after all these directories had been created.


WorkBench - Development Manager
File CMVC Windows Directory Actions Help

```
Context: bering:/ad/projectA_proto/source/c/PortedGUI

<Parent>                          Directory
AddressChange                     Directory
DeletePanel                       Directory
DuplicateSelectionPanel           Directory
Enrollments                       Directory
OnlineUpdate                      Directory
PaymentPanel                      Directory
```

Figure 30. Directories for the User Interface Files of the Ported GUI

6.2.1.2 Mapping ISPF Panels to AIC Interfaces and Windows

AIC introduces the notion of an *interface* (as in user interface). An AIC interface would be used during execution of the application to cause the appearance of a primary window (and, optionally, of secondary windows) on the display managed by the Xserver. Certain characteristics of a window can be changed dynamically by the application, including the title of the window, the behavior of fields, and the callback functions associated with push buttons. A given application might have several primary windows, generated by one or more interfaces. Any given interface, however, can as the application runs be used to generate multiple concurrent primary windows.

Primary windows for a single application act like the the WorkBench Tool Manager and WorkBench Development Manager windows act. They can run in parallel, and relatively independently. Secondary windows act like the dialog box shown earlier popped-up from the WorkBench Development Manager window. Secondary windows are closely related to actions taken in a primary window. They are often transitory, also.

As we mentioned earlier, we intended to have one window for each of the original six ISPF panels. This did not imply that we needed six unique AIC interfaces, though. Of the six ISPF panels in our MVS application, we decided that four would each require a separate AIC interface. However, we decided that the last two ISPF panels could be implemented using the AIC interfaces designed for two of the previously identified four panels. This was because the panel for changing a customer address looked exactly like the one for enrolling a customer and the panel for deleting a customer looked exactly like the one for showing payments. The only significant differences were:

- Some fields were input/output fields on one panel while output-only on the other panel.
- The panel labels differed between the two panels.
- The behavior of the application as a result of user actions differed. For example callbacks would be different.

In the AIXwindows environment, we could make one AIC interface meet the functional requirements of two panels by making the following changes before actually generating the window on the screen:

- Modify the input sensitivity of the appropriate fields.
- Change the window title.
- Redefine the callback functions associated with certain widgets.

We modified the input/output characteristics of a field by setting `XmNsensitive` resource of that field to either `True` (input/outpt) or `False` (output only) in the application. We managed the title with the XtVaGetValues() and XtVaSetValues() functions. We changed the callback functions associated with the push buttons by means of the XtRemoveAllCallbacks() and XtAddCallback() functions.

Therefore, we mapped the original six ISPF panels to just four AIC interfaces. Each interface file name consisted of the name of the panel it replaced plus a `.i` file name extension. We placed each interface file in the directory named for the same panel. We determined, therefore, that the names of these AIC interfaces would be:

- `AddressChange.i`
- `DuplicateSelectionPanel.i`
- `OnlineUpdate.i`
- `PaymentPanel.i`.

6.2.1.3 Source Files Generated from an AIC Interface

AIC can be made to generate C source files from the interface file. The source files consist of a program source file and an include file (sometimes called a *header file*).

Include files generally contain compile time constant declarations, structure and other template declarations, function prototypes, and preproccessor directives (such as the `#include` statement that causes an include file to be processed). The include file generated for an interface was named identically to the corresponding interface file except that the `.i` extension was replaced by a `.h` extension. The include file contained the definition of the context structure for the interface. To ensure AIC generated the include file for each interface, we set the `Aic.includeFile` resource to `True` as shown in Figure 146 on page 222. For more information on setting this and other AIC resources, refer to *User Interface Programming Concepts: AIXwindows Interface Composer, Volume 2*.

The program file is named identically to the interface file from which it is generated, except that its extension is a `.c`. The program code that generates the window (the top-level and children widgets, representing input fields, push buttons) is contained in this file. Callback code can also be generated in this file, if it is stored in the interface.

We decided that the header and program source files that AIC would generate for each interface belonged in the directory named for that interface. The names of these source files would be, not surprisingly:

- `AddressChange.c`
- `AddressChange.h`
- `DuplicateSelectionPanel.c`
- `DuplicateSelectionPanel.h`
- `OnlineUpdate.c`

- `OnlineUpdate.h`
- `PaymentPanel.c`
- `PaymentPanel.h`.

We also decided to copy all include files into a common project directory, in addition to storing them separately in the directories representing each former ISPF panel. When we referenced one of these include files inside a C source program, we enclosed the file name in double quotation marks, and omitted any path name (for example, `#include "AddressChange.h"`). Using the *-I* flag we could instruct the compiler to look for all these include files in the single directory named: `/ad/projectA_proto/include`. This proved less trouble than having to identify a unique path name for each include file in every file that referenced it.

6.2.1.4 Callback Source Files

We decided to separate the implementation of the callbacks from the implementation of the interface by putting the callback code into separate files. This was done because we preferred to use Program Editor over the internal editor provided by AIC 1.1.1. If the callbacks were stored in the interface, then changes to the callback code could only be done by invoking AIC and using its internal editor. The AIC internal editor not only lacked the features of Program Editor, but we also felt it a nuisance to have to use one editor for the callbacks source code, while we used another for all other program source files.

We decided that the names of the C source files implementing the callbacks would be identical to the names of the directories which represented the original six ISPF panels, except that we would append the string `Callbacks` to them. Because the behavior on each panel was different (even though two of them looked identical to two others) we needed six callback source files, one for each original MVS panel. Callback files that corresponded to the AIC interface files would be named:

- `AddressChangeCallbacks.c`
- `DuplicateSelectionPanelCallbacks.c`
- `OnlineUpdate.Callbacks.c`
- `PaymentPanelCallbacks.c`.

The two callback source files for the panels that were not implemented as a separate AIC interface would be named:

- `DeletePanelCallbacks.c`
- `EnrollmentsCallbacks.c`.

6.2.1.5 Callback Include Files

To enable early compiler checks and to detect typing errors as early as possible, we also decided to collect the function declarations (also known as function prototypes) of all callbacks for each interface into an include file for that interface. The base portions of these header file names would be identical to those of the C source files implementing the callbacks, but file name extension for all would be `.h`. These files would be placed in the same directories as the callback source code files. And, as mentioned earlier, all include files would also be copied into a common project directory, named `/ad/projectA_proto/include`. The callback function include files were named:

- `AddressChangeCallbacks.h`
- `DuplicateSelectionPanelCallbacks.h`
- `OnlineUpdate.Callbacks.h`

- `PaymentPanelCallbacks.h`
- `DeletePanelCallbacks.h`
- `EnrollmentsCallbacks.h`.

6.2.1.6 Widgets

Each AIC interface consists of a top-level widget together with all its descendent widgets. Widget names were chosen to be unique across all of the names interfaces. Valid widget names would begin with the name of the interface and continue with descriptive text, for example:

- `AddressChangeCustomerName`

  or

- `OnlineUpdateOkPushButton`.

There would be no unique widgets for the two windows that had no separate interface.

6.2.1.7 Callback Functions

The corresponding callback function names would be composed of the panel (or, AIX window) name, followed next by the name of the widget they were associated with, and followed lastly by the string *Callback*. For example, valid callback functions would be:

- `PaymentPanelEndPushButtonActivateCallback`

  or

- `AddressChangeOKPushButtonActivateCallback`.

Callbacks associated with the two windows that were not generated from a unique interface would simply replace the other window's name in the callback label with their own, as in:

- `EnrollmentsUpdateOkPushButtonActivateCallback`.

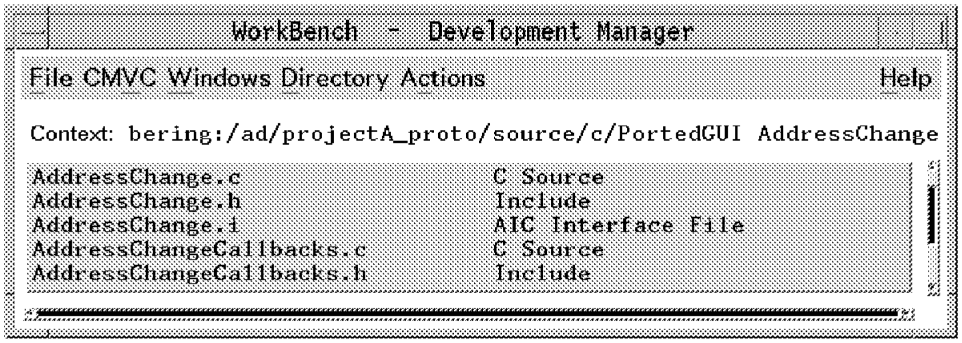
The only parameter that would be passed to each callback was the ID of the corresponding widget.

6.2.1.8 Total Files Associated with Each Window

For each MVS panel (and therefore for each AIX window) that was implemented with a unique AIC interface, we had the five source files stored in a common directory. For example, the files associated with the AddressChange window would be stored in the `/ad/projectA_proto/source/c/PortedGUI/AddressChange` directory and named:

AddressChange.i	AIC interface source file for the AddressChange window
AddressChange.h	Include file generated by AIC with a structure definition for the interface
AddressChange.c	C source file generated by AIC for the AddressChange interface
AddressChangeCallbacks.h	External declarations of the callbacks for AddressChange
AddressChangeCallbacks.c	Implementation of the callbacks for AddressChange.

Development Manager directory contents listing of the AddressChange directory is shown in Figure 31 on page 74.


WorkBench - Development Manager
File CMVC Windows Directory Actions Help

```
Context: bering:/ad/projectA_proto/source/c/PortedGUI AddressChange

AddressChange.c                    C Source
AddressChange.h                    Include
AddressChange.i                    AIC Interface File
AddressChangeCallbacks.c           C Source
AddressChangeCallbacks.h           Include
```

Figure 31. Source Files for the AddressChange Window

For the two windows that were implemented by reusing AIC interfaces, we had only two source files. The Enrollment window was built using the AIC interface designed for the Address Change window. Thus, in the `Enrollments` directory, we had only these files:

EnrollmentsCallbacks.h	External declarations of the callbacks for Enrollments
EnrollmentsCallbacks.c	Implementation of the callbacks for Enrollments.

6.2.1.9 Mapping the GUI Versions to AIC Projects

A collection of related interfaces, such as all those supporting a given application, or a given version of an application, can be collected together by means of an AIC *project*. Certain information that is necessary to generate the C code is stored in the file representing the project.

The project file's name includes the extension: `prj`. Mapping our effort to AIC projects proved easy. We decided that the AIC project would suit our needs to produce two separate GUIs for our application. We would have one project for the minimum effort port, `PortedGUI.prj`, and another for the modernized GUI, `ImprovedGUI.prj`.

AIC generates certain default names based on the project file name, unless explicitly overridden by the developer. The application class, for instance, unless specified differently, is named identically to the base portion of the project file name. The application class is the identifier that ties together all Xwindow resources belonging to the widgets, comprising the application GUI. It generates a label in the C source code generated by AIC and is reflected in the various mechanisms for setting and querying those resources. This label becomes the name of the application's resource file, and appears as the client class label in this file.

It is fine to have the project and application class be identical if you will have only one project file for a given application. In our case, we had multiple project files for different versions of essentially the same application. Had this been a production situation, where *PortedGUI* was simply a preliminary version and *ImprovedGUI* was a later version of the same application, we might have explicitly set the application class identically for both projects. For example, `ibmoupd` might have been appropriate, if it were the name of the original MVS executable file with which the users and test programs were already familiar. Instead, since we were only doing a proof of concept effort, we set the application class to either `PortedGUI` or `ImprovedGUI`, not worrying about the possible conflicts that might be caused in a more realistic environment.

The *makefile* file generated by AIC also shares the same base file name as the project file. For example: AIC generated a *makefile* file, named `PortedGUI.mk`, when the interface was loaded in from `PortedGUI.prj`, but the *makefile* file was named `ImprovedGUI` when the other project file was loaded. There was no problem with this default, as there were in fact differences between the two.

Another implication to consider when choosing the project name is that AIC would generate a *makefile* file that would make the executable file name identical to the base portion of the project file name. Again, during informal testing, this was just fine, as we wanted to distinguish between the executable files of the two versions. However, had we been doing this for production, we might have modified the *makefile* file to rename the executable file to the common file name of the application. As suggested earlier, this name might have been `ibmoupd`.

6.2.1.10 Using AIC: Prototype Tool or Development and Maintenance Tool?

Another decision we needed to make early was how to use AIC over the life of the application. AIC can be used merely to produce a prototype of the GUI. In this case, the prototype is developed interactively using AIC. AIC is used to generate the first cut of the C or C^{++} source code. That source is then edited, compiled, and debugged independently of AIC, as needed during the remainder of the development and maintenance phases of the application's lifetime. This approach can seem appropriate where large amounts of application code are hand coded independently and the GUI is retroactively fitted to the application. The approach also has reusability and maintainability implications with respect to the GUI.

AIC can also be used as a development and maintenance tool. In this scenario, the source code is never edited. AIC is used to make all changes to the GUI. Then new source code is regenerated and that code is recompiled, tested, and so on.

If a project intends to use AIC throughout the development and maintenance phases, then the files in which AIC stores the design and implementation decisions made by the user must be the subjects of configuration control. AIC accumulates this data as the user interactively creates the GUI using AIC. This data is stored—in a format understandable only to AIC—in the interface, palette, and project files that AIC creates.

The interface file contains information about the windows visual and behavioral components. The *palette* is a library of widgets or widget hierarchies that can be defined and then reused in the project. We did not use palettes, but mention them occasionally in this volume for completeness. The project file contains information relating several interfaces and data common to them. AIC generates the source code from the data it stores in these files.

Both approaches can be correct; so the developer must deteremine early which to take. We determined AIC would be used not only to develop, but also to maintain the code for the long term. This issue influenced decisions we later reached when we integrated our C++ code with that generated by AIC 1.2.

6.2.2 Design of the Windows Using AIC

After reaching these decisions and defining these conventions, we started the design and layout of the windows, primarily with AIC. AIC was accessed from SDE WorkBench/6000.

6.2.2.1 Starting AIC from the Tool Manager

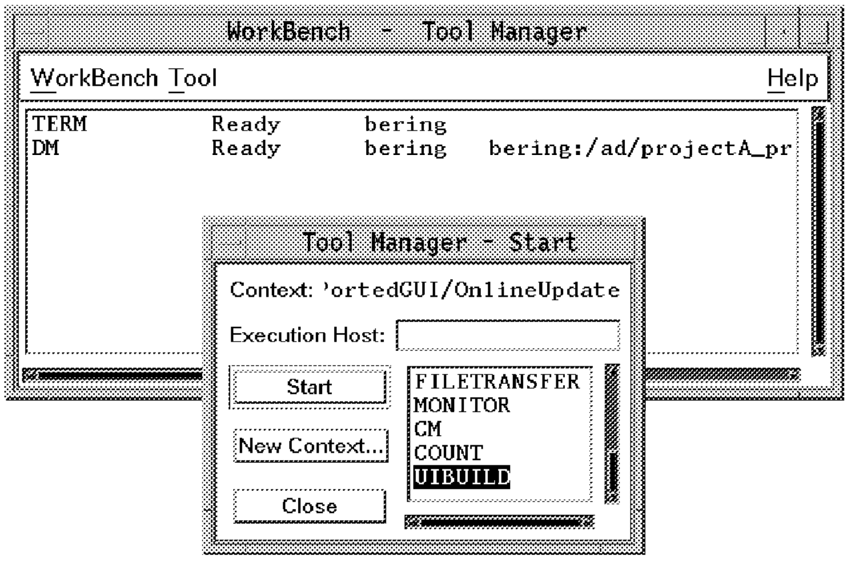
We chose **Start** from the Tool pull-down menu of the WorkBench Tool Manager window. This caused a dialog box to pop-up as shown in Figure 136 on page 214. A scrolling list appears in this dialog box identifying the classes of tools available to SDE WorkBench/6000 at the moment.

To find the class of tool we needed, we used the scroll bar to the right of the list, scrolling the list up and down. To select the class, we simply clicked the left mouse button while the cursor was over the selection. In the dialog box presented to our developer, the option `UIBUILD` identified the class of tools class used to build user interfaces. The UIBUILD class tool available on the host *bering* was AIC version 1.1.1.

The data context for this tool was preset for us by Tool Manager, based on the last context setting. This default can be changed, as it can be in Development Manager window, by pressing the right mouse button while the graphic cursor is over the context field. As in Development Manager a series of cascading pop-ups can be traversed until the correct context is identified to Tool Manager. Alternatively, the data context can be changed by clicking on the **New Context** push button and filling in the field on the resulting dialog box.

The execution host is set by clicking the left mouse button in that field and typing in the name of a valid host. The execution host defaults to that host on which Tool Manager is running. In our case, the default value *bering* was adequate.

Having selected the tool class and set the execution host and data context, we started the tool by selecting the **Start** push button. Figure 32 on page 77 shows the Tool Manager Start dialog box ready to start up the UIBUILD class tool (AIC).



```
TERM          Ready      bering
DM            Ready      bering   bering:/ad/projectA_pr
```

Context: 'ortedGUI/OnlineUpdate

Execution Host:

```
FILETRANSFER
MONITOR
CM
COUNT
UIBUILD
```

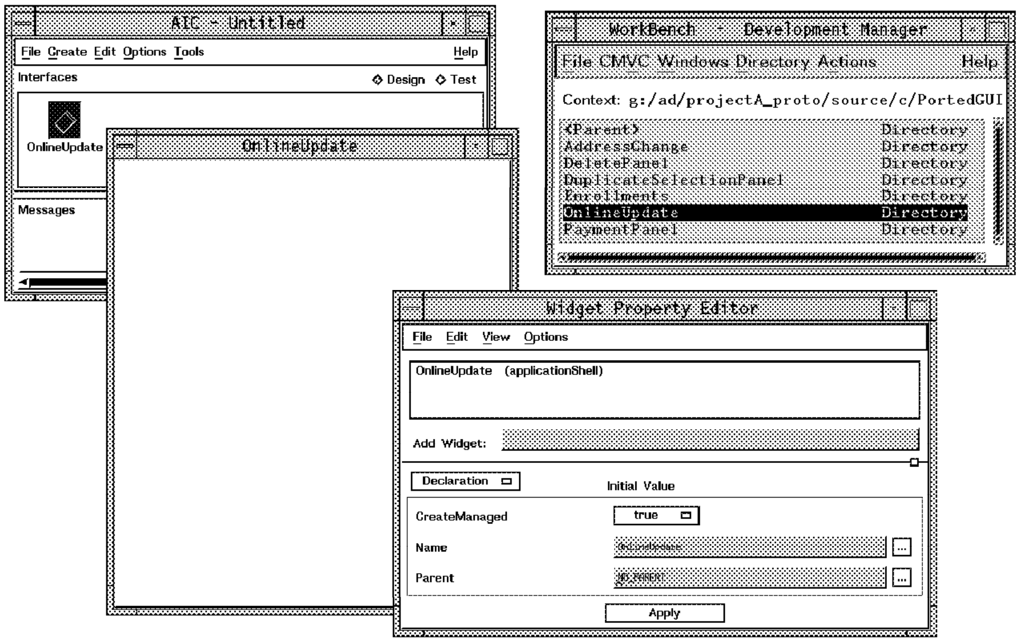
Figure 32. Tool Manager Start Dialog Box

One of the advantages of using SDE WorkBench/6000 is that the developer does not need to learn the specifics of invoking a particular tool. The user does not become familiar with flags and parameters used on the UNIX command line to start up the tools. Instead, the developer asks for a list of the classes of tools available and chooses the class that is needed for the moment. If the particular tool in a class is replaced by the system manager at some later date, the invocation of the new tool will be no different.

Another advantage is that the set of tool classes is not fixed by SDE WorkBench/6000. In our case, the class UIBUILD was created when we integrated AIC 1.1.1 with SDE WorkBench/6000. The integration effort is described in 9.4, “Tailoring SDE WorkBench/6000 for AIC Programmers” on page 212.

6.2.2.2 Developing an Interface and Laying out the Window

After AIC started, we began to prototype the layout of the `OnlineUpdate.i` interface (GUI window), which implements the main menu panel of the original application. We selected **Shells** from the Create pull-down menu and selected **Application** to create the application shell as the top widget for the interface. Using the right mouse button we resized the window and could see the top level widget. We clicked on the window, pressed the right mouse button and selected **Property Editor** from the pop-up menu. Following the naming conventions established earlier, we wanted to change the name of the widget to *OnlineUpdate* rather than the default *applicationShell1*; so we selected the **Declaration** choice from the toggle button. We entered *OnlineUpdate* as the new name for the interface, selected **Apply**, and saw the windows as shown in Figure 33 on page 78.



```
Context: g:/ad/projectA_proto/source/c/PortedGUI
<Parent>                    Directory
AddressChange               Directory
DeletePanel                 Directory
DuplicateSelectionPanel     Directory
Enrollments                 Directory
OnlineUpdate                Directory
PaymentPanel                Directory
```

OnlineUpdate (applicationShell)

CreateManaged	true
Name	[illegible]
Parent	[illegible]

Figure 33. Changing the Interface Declaration Using the Widget Property Editor

Using information in *User Interface Programming Concepts: AIXwindows Interface Composer, Volume 2*, we proceeded to prototype the interface (window). We designed the interface to match the original ISPF panel as closely as possible, and finally got to a complete layout including all the buttons, labels, and text fields that were required. From the interface browser we selected **Save as** from the File pull-down menu, and saw the four application windows shown in Figure 34 on page 79.

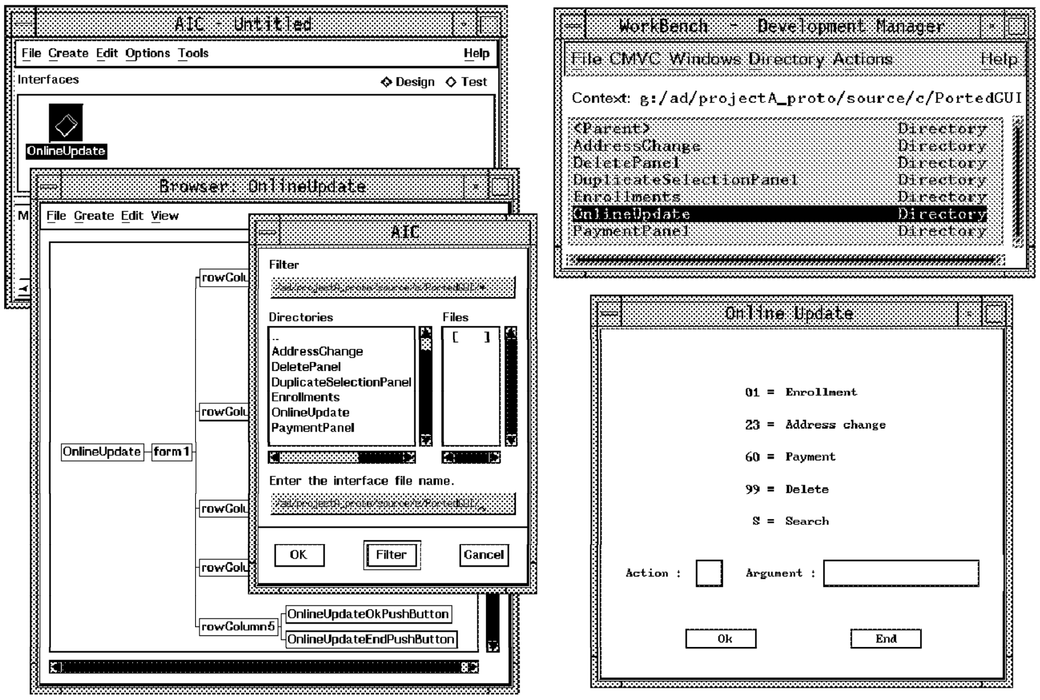


Figure 34. Saving the OnlineUpdate Interface File

We finally ended up with the interfaces shown in Figure 35.

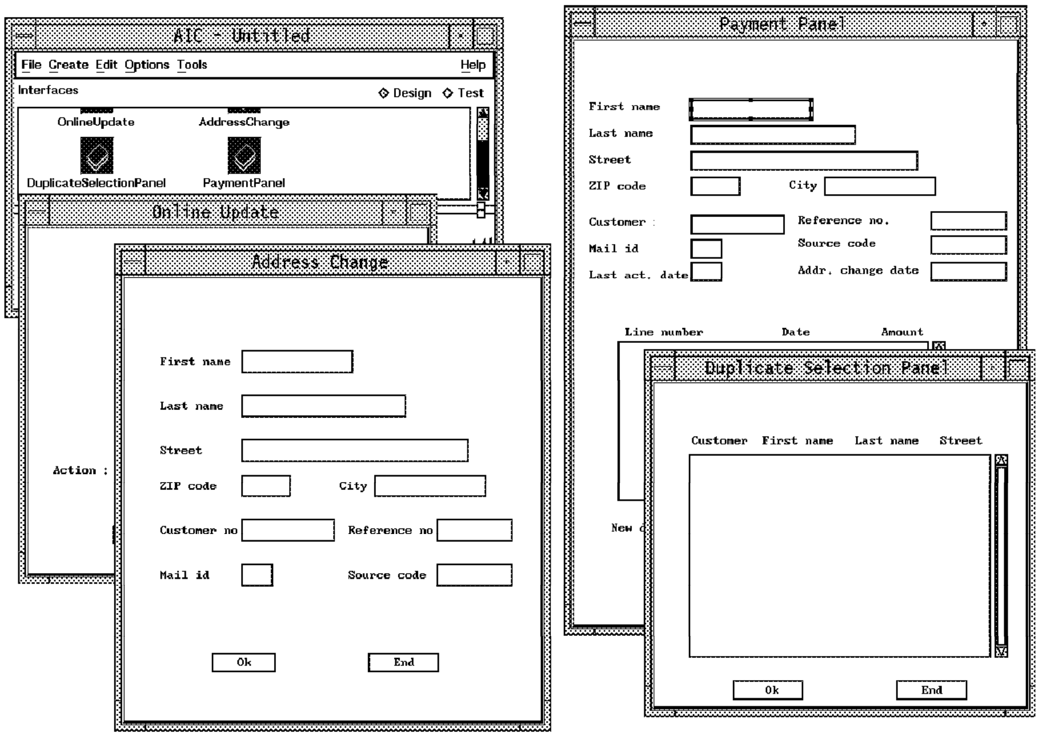


Figure 35. The Interfaces of the Ported GUI

Similarly, for each remaining GUI window, we saved an interface file in the corresponding directory. We used either the **uxcgen** command or the **Write C Code As** option on the File pull-down of AIC to make AIC generate the interface include files and the corresponding interface C source files for the remaining GUI windows: `AddressChange`, `DuplicateSelectionPanel`, and `PaymentPanel`. We moved these files into subdirectories under our `/ad/projectA_proto/include` directory. For information about generating C code for AIC projects and interfaces refer to *User Interface Programming Concepts: AIXwindows Interface Composer, Volume 2*, to learn more about this AIC feature.

6.2.3 Implementing the Callbacks

After we had finished the implementation of the user interface windows, we started to implement the callbacks. To identify the callbacks we would need we looked at the windows for push buttons. Look at the OnlineUpdate window of our application shown in Figure 36. There are two push buttons on this window. For each of these two buttons we had to provide the callback triggered when the button is pressed. These two callback functions would be called:

- `OnlineUpdateOkPushButtonActivateCallback`

  and
- `OnlineUpdateEndPushButtonActivateCallback`.

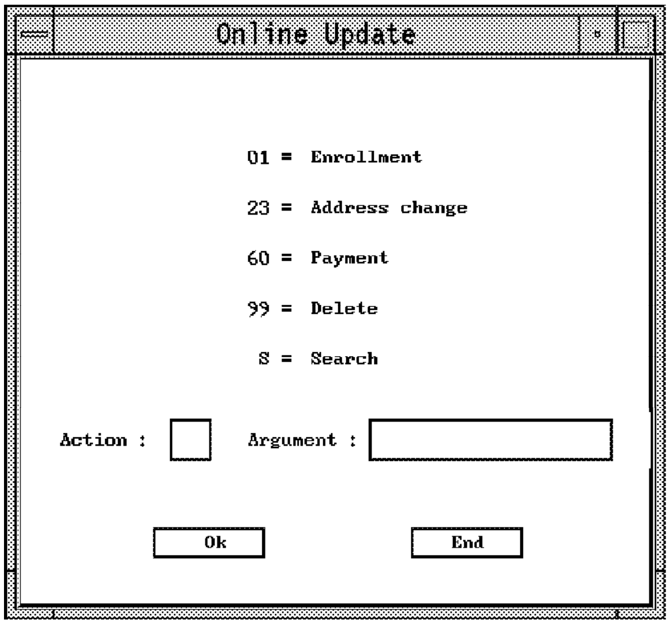


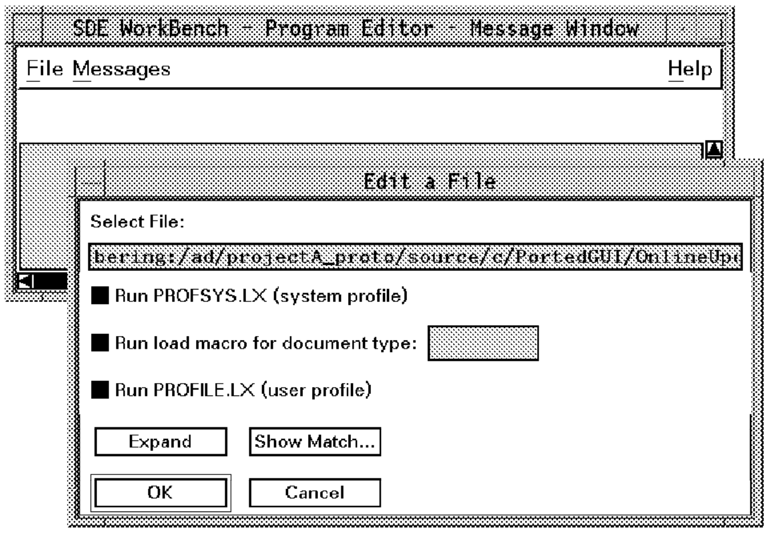
Figure 36. The Online Update Window

Implementing the callbacks required us to create two files initially: callback function prototype source files and the corresponding callback stub source files. The stubs would eventually be replaced by code which called the COBOL functions of our original application.

6.2.3.1 Using Program Editor to Create the Callback Function Prototypes

To edit the files, we invoked Program Editor. We started Program Editor just as we started AIC, using the Tool pull-down of the WorkBench Tool Manager window. Program Editor initially presents the SDE WorkBench Program Editor - Message Window. Messages from Program Editor to the user are displayed in this window. Additional windows are generated as files (called documents in Program Editor documenation) are edited. Additional windows can also be brought up to show multiple views of the same file. A view presents another section of the file, or a subset of the data in the file based on the parsing algorithms supported for that type of file.

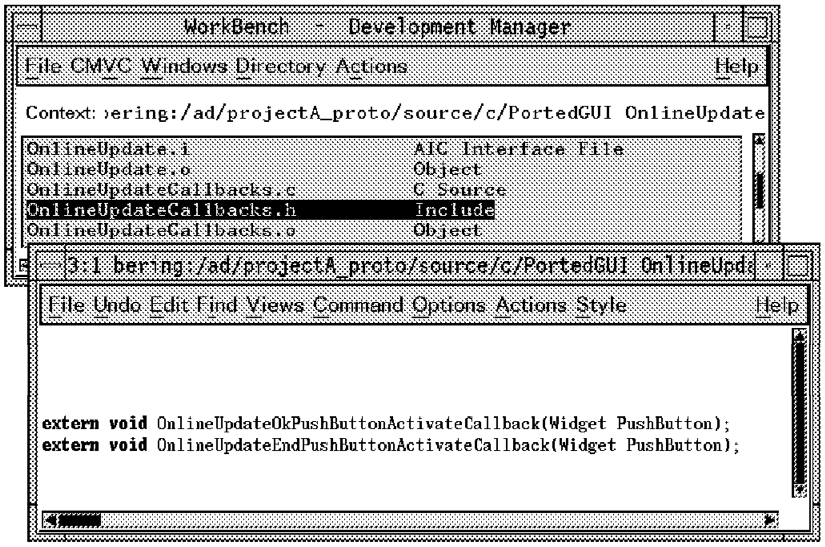
Like other AIX AD tools that are integrated with SDE WorkBench/6000, its primary windows include a menu bar with a File pull-down. Clicking on the **Edit** selection of this pull-down, we brought up the Edit a File dialog box window. We entered the host name, path name, and file name of the file we intended to create in the Select File field. This name was, as previously decided, `bering:/ad/projectA_proto/source/c/PortedGUI/OnlineUpdate OnlineUpdateCallbacks.h`. The message window and dialog box window are shown in Figure 37.



Select File:
bering:/ad/projectA_proto/source/c/PortedGUI/OnlineUpd
■ Run PROFSYS.LX (system profile)
■ Run load macro for document type:
■ Run PROFILE.LX (user profile)

Figure 37. The WorkBench Program Editor - Message Window and Edit a File Window

A document window appeared representing the new file we wanted to create. We typed in two lines of code, one prototype for each callback function. The contents of this file is shown in Figure 38 on page 82.


WorkBench - Development Manager
File CMVC Windows Directory Actions Help
Context: ›ering:/ad/projectA_proto/source/c/PortedGUI OnlineUpdate
3:1 bering:/ad/projectA_proto/source/c/PortedGUI OnlineUpd
File Undo Edit Find Views Command Options Actions Style Help

OnlineUpdate.i	AIC Interface File
OnlineUpdate.o	Object
OnlineUpdateCallbacks.c	C Source
OnlineUpdateCallbacks.h	Include
OnlineUpdateCallbacks.o	Object

```
extern void OnlineUpdateOkPushButtonActivateCallback(Widget PushButton);
extern void OnlineUpdateEndPushButtonActivateCallback(Widget PushButton);
```

Figure 38. The Include File for the Callbacks for the OnlineUpdate Window

The title bar of this document window showed us that this is the third file (document) loaded by the editor, and this was the first view of it. This was indicated by the cryptic numbers: `3:1`. It also showed us the host, directory, and file names of the file, although not all of the information is shown, because of the size we chose for the illustration. Notice that the document window also has a menu-bar with a File pull-down. It also has several other pull-downs relating to editing the file and controlling the editor, which were not used at this time.

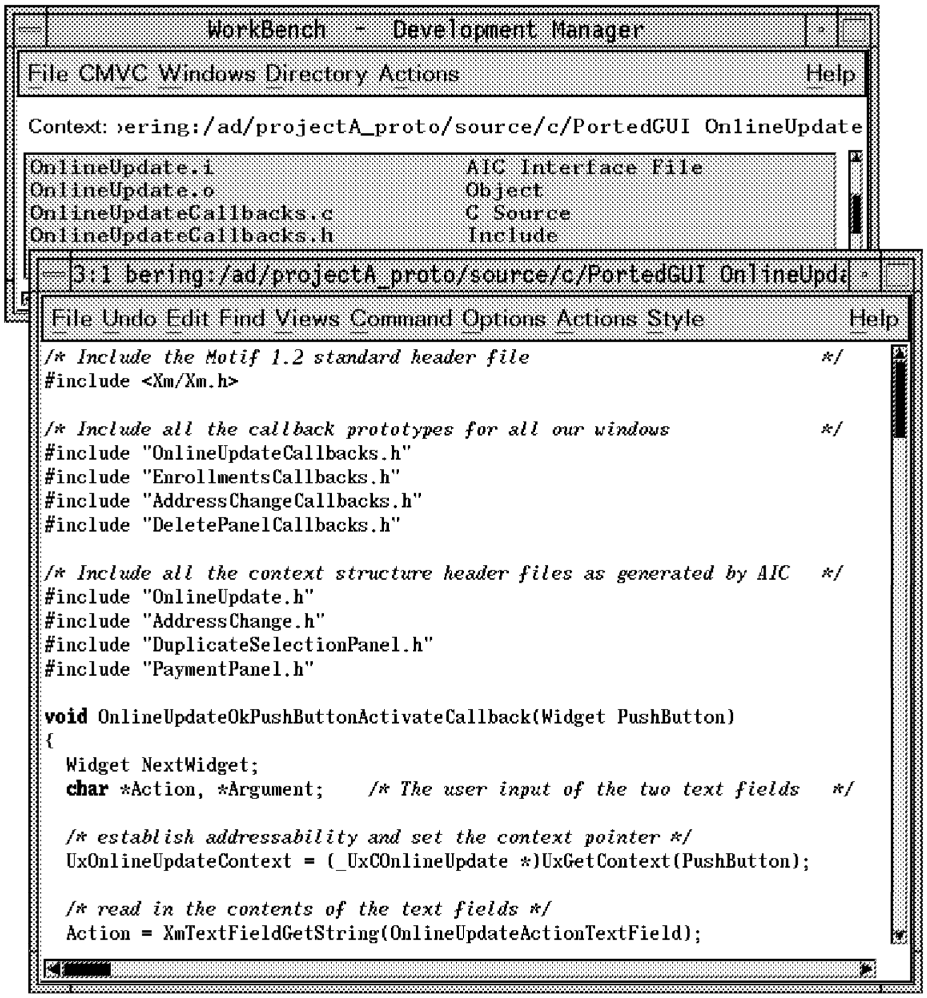
Also, notice in this and subsequent screen captures that Program Editor recognizes syntactical elements of the C programming language and displays them in different fonts. In Figure 38, the key words `extern` and `void` show up in boldface type, while the rest of the statements show up in another font. Were these illustrations in color, you could see that these fonts can appear in different colors, also.

Having entered this file's contents, we selected **Save** from the File pull-down to write the file in the `/ad/projectA_proto/source/c/OnlineUpdate` directory, where all source- and AIC-generated files related to this panel are stored. This file would be named `OnlineUpdateCallbacks.h`, according to our naming conventions. However, we determined earlier that all include files would be also placed in single project directory: `/ad/projectA_proto/include`. To do this we copied the file into that directory, using the **Save As** selection from the File pull-down.

6.2.3.2 Creating Callback Stub Code

We next started to implement the callback code itself by using the **Edit** selection from the File pull-down of the document window. After entering the name of our new file, `OnlineUpdateCallbacks.c`, a second document window appeared.

According to our design principles and the file naming standards explained earlier, we begin this file identifying the appropriate `include` files, as shown in Figure 39 on page 83.


WorkBench - Development Manager
File CMVC Windows Directory Actions Help
3:1 bering:/ad/projectA_proto/source/c/PortedGUI OnlineUpd
File Undo Edit Find Views Command Options Actions Style Help

Context: ›ering:/ad/projectA_proto/source/c/PortedGUI OnlineUpdate

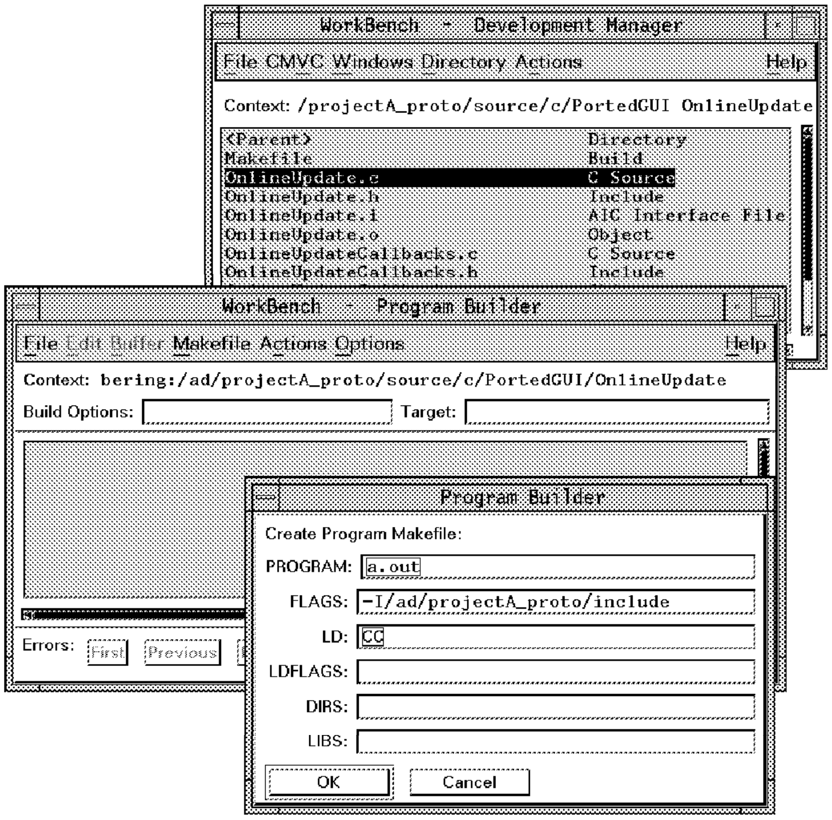
```
OnlineUpdate.i                   AIC Interface File
OnlineUpdate.o                   Object
OnlineUpdateCallbacks.c          C Source
OnlineUpdateCallbacks.h          Include
```

```
/* Include the Motif 1.2 standard header file                    */
#include <Xm/Xm.h>

/* Include all the callback prototypes for all our windows        */
#include "OnlineUpdateCallbacks.h"
#include "EnrollmentsCallbacks.h"
#include "AddressChangeCallbacks.h"
#include "DeletePanelCallbacks.h"

/* Include all the context structure header files as generated by AIC   */
#include "OnlineUpdate.h"
#include "AddressChange.h"
#include "DuplicateSelectionPanel.h"
#include "PaymentPanel.h"

void OnlineUpdateOkPushButtonActivateCallback(Widget PushButton)
{
  Widget NextWidget;
  char *Action, *Argument;    /* The user input of the two text fields   */

  /* establish addressability and set the context pointer */
  UxOnlineUpdateContext = (_UxCOnlineUpdate *)UxGetContext(PushButton);

  /* read in the contents of the text fields */
  Action = XmTextFieldGetString(OnlineUpdateActionTextField);
```

Figure 39. Editing the OnlineUpdateCallbacks.c File Using Program Editor

We designed callback stub code at this point and coded it. The stub code consisted of calls to `UxPopupInterface` and `UxPopdownInterface` only, to provide a visual indication that the callbacks had been successfully triggered during testing. How these work is explained in the discussion of ux library functions in *User Interface Programming Concepts: AIXwindows Interface Composer, Volume 2*. The stub code enabled us to test the user interface code independently of the COBOL logic that it would exercise in the completed application. This stub code would later be replaced by COBOL logic ported over from the original application. Refer to 6.3, “Design and Implementation of the COBOL Code” on page 89 and 6.5, “Integration and Test” on page 122 for details about the changes applied to the original COBOL code when integrating it with the user interface, and about the C-to-COBOL interlanguage calling issues.

Once the `OnlineUpdateCallbacks.c` file was completed, we saved the file and selected the BUILD class tool from the Tool Manager Start window as shown in Figure 136 on page 214. We set the context environment of Program Builder to the /ad/projectA_proto/source/c/PortedGUI/OnlineUpdate directory and selected **Create Program** from the Makefile pull-down of Program Builder. We added the option -I/ad/projectA_proto/include to the FLAGS field in the Program Builder dialog window shown in Figure 40.


WorkBench - Development Manager
File CMVC Windows Directory Actions Help
WorkBench - Program Builder
File Edit Buffer Makefile Actions Options Help
Build Options:
Target:
Errors:
First
Previous
Program Builder
OK
Cancel

Context: /projectA_proto/source/c/PortedGUI OnlineUpdate

```
<Parent>                        Directory
Makefile                        Build
OnlineUpdate.c                  C Source
OnlineUpdate.h                  Include
OnlineUpdate.i                  AIC Interface File
OnlineUpdate.o                  Object
OnlineUpdateCallbacks.c         C Source
OnlineUpdateCallbacks.h         Include
```

Context: bering:/ad/projectA_proto/source/c/PortedGUI/OnlineUpdate

Create Program Makefile:

PROGRAM: a.out

FLAGS: -I/ad/projectA_proto/include

LD: CC

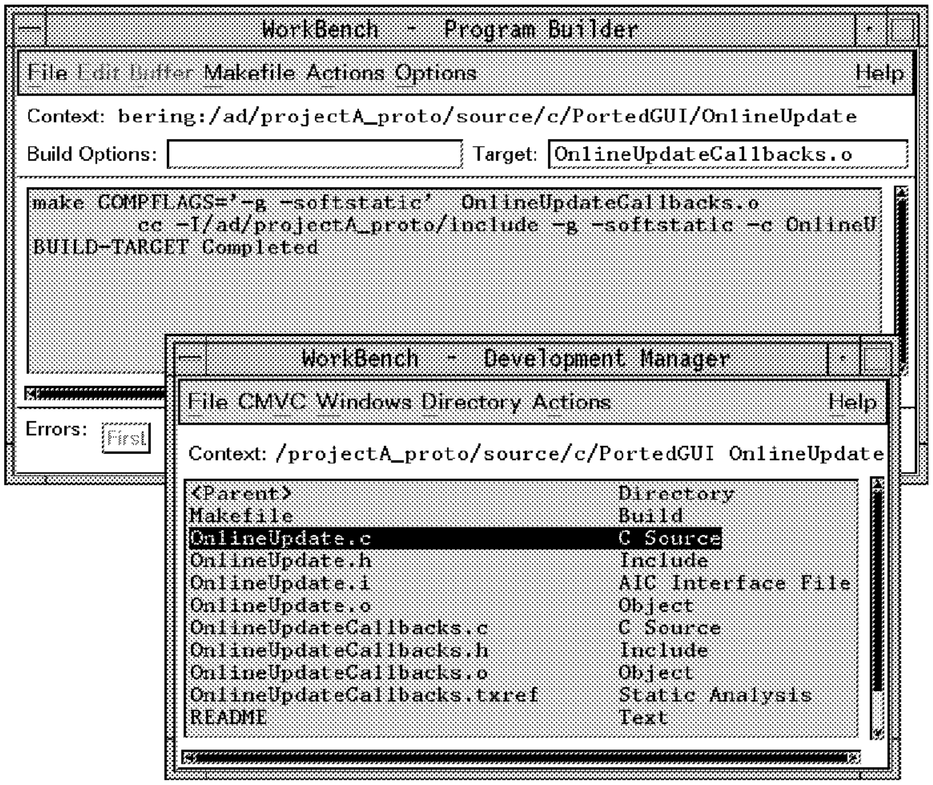
LDFLAGS:

DIRS:

LIBS:

Figure 40. Generating a Makefile File Using Program Builder

When we selected **OK** on the Program Builder window, the Makefile file was generated. We then selected **Update Dependencies** from the Makefile pull-down from the WorkBench Program Builder window to have the generated *makefile* file reflect the dependencies of OnlineUpdateCallbacks.c on the various header files as shown in Figure 39 on page 83. The *makefile* file was updated by Program Builder, and finally we could invoke Program Builder with the proper target of OnlineUpdateCallbacks.o. Program Builder started the compilation with the proper flags -I/ad/projectA_proto/include, and created an object file OnlineUpdateCallbacks.o. This was reflected in Development Manager window shown in Figure 41 on page 85.


WorkBench - Program Builder
File Edit Buffer Makefile Actions Options Help

Context: bering:/ad/projectA_proto/source/c/PortedGUI/OnlineUpdate

Build Options: Target: OnlineUpdateCallbacks.o

```
make COMPFLAGS='-g -softstatic'  OnlineUpdateCallbacks.o
        cc -I/ad/projectA_proto/include -g -softstatic -c OnlineU
BUILD-TARGET Completed
```

Errors: First
WorkBench - Development Manager
File CMVC Windows Directory Actions Help

Context: /projectA_proto/source/c/PortedGUI OnlineUpdate

```
<Parent>                        Directory
Makefile                        Build
OnlineUpdate.c                  C Source
OnlineUpdate.h                  Include
OnlineUpdate.i                  AIC Interface File
OnlineUpdate.o                  Object
OnlineUpdateCallbacks.c         C Source
OnlineUpdateCallbacks.h         Include
OnlineUpdateCallbacks.o         Object
OnlineUpdateCallbacks.txref     Static Analysis
README                          Text
```

Figure 41. Compiling OnlineUpdateCallbacks.c

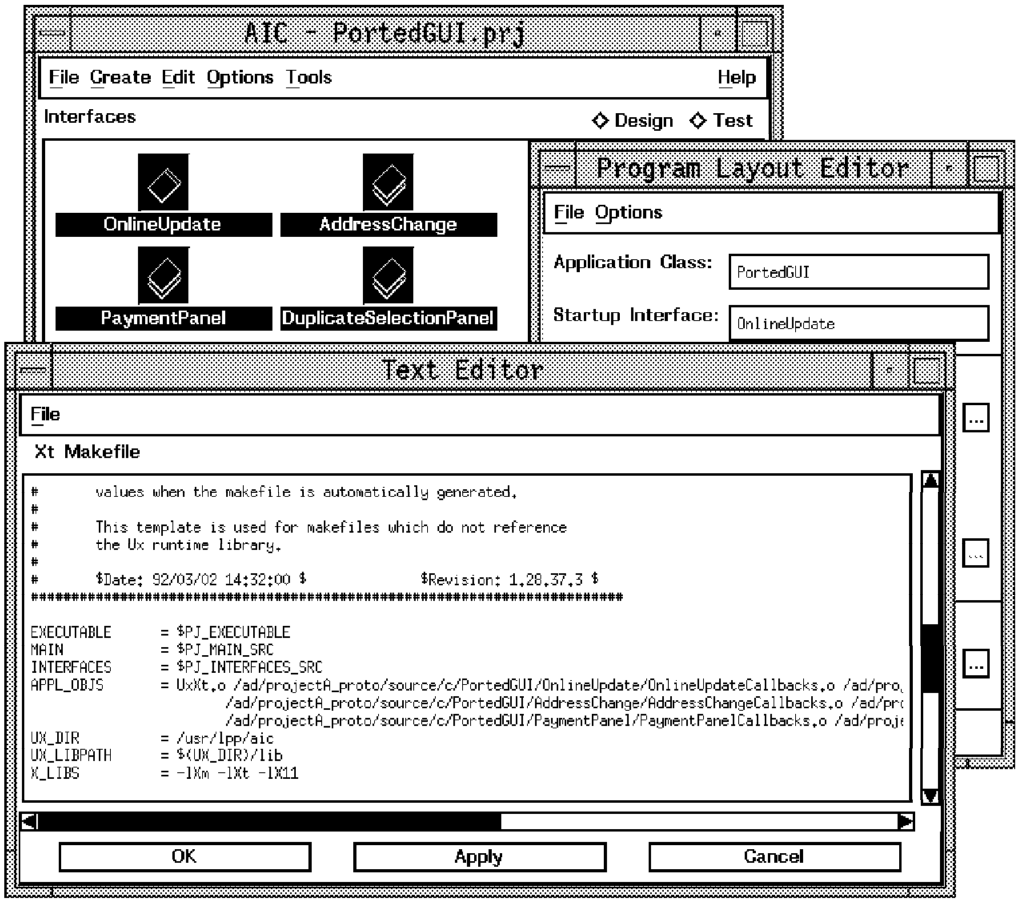
We used the **Close Document** selection from the File pull-down to stop editing these callback source files. We did not need to exit Program Editor, if we wanted to begin editing other files. Using the **Edit** selection from the File pull-down of the message window we could begin the cycle again.

The same sequence of steps was performed for the remaining five callback source files in the corresponding directories and contexts. Finally, these object files were successfully created in their corresponding directories:

- `AddressChangeCallbacks.o`
- `DuplicateSelectionCallbacks.o`
- `OnlineUpdateCallbacks.o`
- `PaymentPanelCallbacks.o`
- `DeletePanelCallbacks.o`
- `EnrollmentsCallbacks.o`.

6.2.3.3 Testing the GUI with Stub Code

We decided to test the end user interface stand-alone. To do this, we had to generate an executable file from the source files generated by AIC and the callback stubs. We loaded all the interface files into the AIC project that would be written out in the file `PortedGUI.prj`. We wanted to use the AIC **Write C Code** selection that can be chosen from the File pull-down in the AIC main window. Before we did this, we had to modify the *makefile* file to add the six callbacks to the list of object modules to be linked. We selected **Program Layout** from the Edit pull-down of the main AIC window, and pressed the **...** button left to the `Xt Makefile` text widget in the Program Layout Editor window to edit the *makefile* file template. We added the names of the object modules for the six callback files to the definition of the `APPL_OBJS` *makefile* variable as shown in Figure 42. This is discussed further in *Supporting Projects in AIC* in the *User Interface Programming Concepts: AIXwindows Interface Composer, Volume 2*.



```
#       values when the makefile is automatically generated.
#
#       This template is used for makefiles which do not reference
#       the Ux runtime library.
#
#       $Date: 92/03/02 14:32:00 $             $Revision: 1.28.37.3 $
##########################################################################

EXECUTABLE      = $PJ_EXECUTABLE
MAIN            = $PJ_MAIN_SRC
INTERFACES      = $PJ_INTERFACES_SRC
APPL_OBJS       = UxXt.o /ad/projectA_proto/source/c/PortedGUI/OnlineUpdate/OnlineUpdateCallbacks.o /ad/pro
                        /ad/projectA_proto/source/c/PortedGUI/AddressChange/AddressChangeCallbacks.o /ad/pr
                        /ad/projectA_proto/source/c/PortedGUI/PaymentPanel/PaymentPanelCallbacks.o /ad/proj
UX_DIR          = /usr/lpp/aic
UX_LIBPATH      = $(UX_DIR)/lib
X_LIBS          = -lXm -lXt -lX11
```

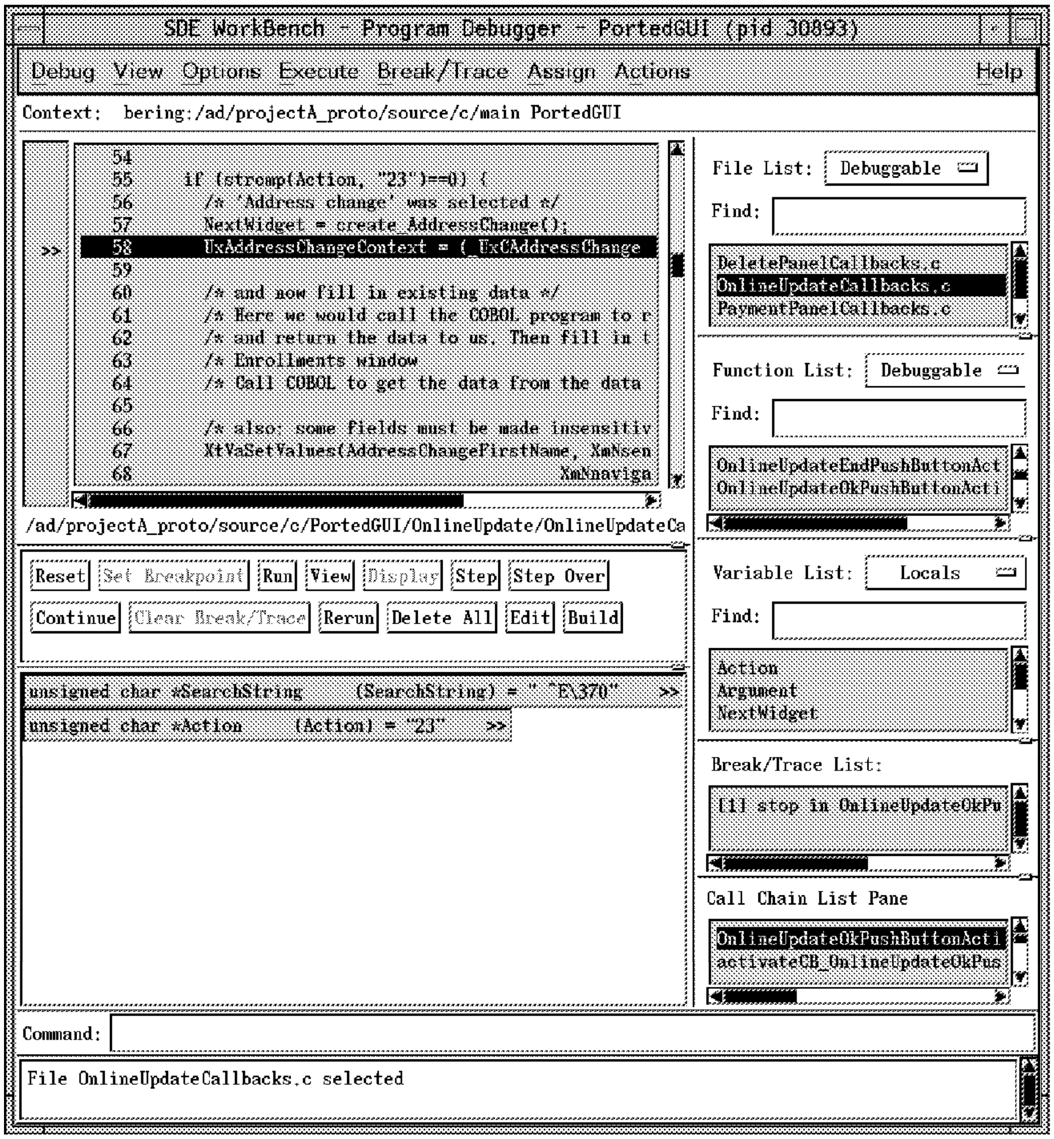
Figure 42. Generating a Makefile File for Testing the Ported GUI

We pressed the **OK** button in the Text Editor window and pressed the **Apply** button in the Program Layout Editor window to activate the changes. We then selected **Write C Code** from the File pull-down of the AIC main window to generate the code, generate the *makefile* file, and build the program. The *makefile* file generated by AIC assumed that the C source file for each interface and the corresponding include files had already been generated by AIC. This is because AIC triggered the generation process before it run the *makefile* file.

As an alternative we could also have edited and created the *makefile* file ourselves, using Program Editor, and run the *makefile* file manually. This *makefile* file could have used the **uxcgen** utility that is shipped with AIC to generate the source code, compile, and link in one *makefile* file.

AIC would generate an executable file called PortedGUI. We decided to use Program Debugger to test the code, and selected the PortedGUI file in the WorkBench Development Manager window with the left mouse button. We then selected **Debug** from the Actions pull-down to start Program Debugger. We configured Program Debugger to use the correct directories to look for the source file by selecting **Dbx Configuration...** from the Options pull-down of Program Debugger. We added the path /ad/projectA_proto/source/c/PortedGUI/OnlineUpdate to the end of the path list in the Use Path text field and pressed the **Apply** button. In the WorkBench Program Debugger window we selected the entry for OnlineUpdateOkPushButtonActivateCallback in the function list and pressed the **Set Breakpoint** button to set a debug break point when entering this function. We then pressed the **Run** button to start the debug session.

The system showed the main menu of the application as shown in Figure 36 on page 80, where we entered 23 in the Action field and 12345 in the Argument field. We pressed the **OK** push button and Program Debugger stopped as the specified break point was reached. We could then use the **Step over** push button to continue with the program execution instruction by instruction and use the **View** option to look at the contents of certain variables. Figure 43 on page 88 shows how we displayed the contents of the variable Action.


SDE WorkBench - Program Debugger - PortedGUI (pid 30893)
Debug View Options Execute Break/Trace Assign Actions Help

Context: bering:/ad/projectA_proto/source/c/main PortedGUI

```

    if (strcmp(Action, "23")==0) {
       /* 'Address change' was selected */
       NextWidget = create_AddressChange();
>>     UxAddressChangeContext = (_UxCAddressChange

       /* and now fill in existing data */
       /* Here we would call the COBOL program to r
       /* and return the data to us. Then fill in t
       /* Enrollments window
       /* Call COBOL to get the data from the data

       /* also: some fields must be made insensitiv
       XtVaSetValues(AddressChangeFirstName, XmNsen
                                             XmNnaviga
```

/ad/projectA_proto/source/c/PortedGUI/OnlineUpdate/OnlineUpdateCa

Reset Set Breakpoint Run View Display Step Step Over
Continue Clear Break/Trace Rerun Delete All Edit Build

```
unsigned char *SearchString     (SearchString) = " ^E\370"   >>
unsigned char *Action     (Action) = "23"    >>
```

File List: Debuggable

Find:

DeletePanelCallbacks.c
OnlineUpdateCallbacks.c
PaymentPanelCallbacks.c

Function List: Debuggable

Find:

OnlineUpdateEndPushButtonAct
OnlineUpdateOkPushButtonActi

Variable List: Locals

Find:

Action
Argument
NextWidget

Break/Trace List:

[1] stop in OnlineUpdateOkPu

Call Chain List Pane

OnlineUpdateOkPushButtonActi
activateCB_OnlineUpdateOkPus

Command:

File OnlineUpdateCallbacks.c selected

Figure 43. Using the Program Debugger

Thus, we used Program Debugger to test the application. Errors were detected and corrected, and the compile and build cycle as described in the previous chapter was repeated until we had a stable version of the application.

This version of the callbacks along with the corresponding version of the AIC interface source files was submitted as the initial baseline version placed under control of CMVC/6000.

6.3 Design and Implementation of the COBOL Code

Our porting efforts were aimed at preserving as much of the original COBOL code as possible. By doing so we hoped to gain a couple of things:

- Reuse of most of the original code
- Low porting cost
- Low maintenance cost
- A single base code.

We determined to keep a common program logic. This is illustrated with Figure 7 on page 37 in Chapter 4, “Defining the Project and Setting Its Goals” on page 29. This decision was only possible because the original code was written in a structured way. Even though we implemented a completely new user interface, the changes to the main logic proved to be minor. The next paragraphs show step-by-step how the original code was transformed from an ISPF-based MVS application to a GUI-based AIX application. We used SDE WorkBench/6000 and the following integrated application development tools from IBM and Micro Focus:

- Development Manager
- A few pieces of the code to show the concept
- Program Builder
- Micro Focus COBOL
- Micro Focus Animator

6.3.1 Design Decisions

When porting an application from one platform to another, the goal should be to keep as much of the original logic as possible. Sometimes this is just not possible, because the code is not suited for it. In our case, we were able to keep most of the original code intact, except what we replaced for the user interface upgrade.

Looking at our original code, we found that it was made up of larger code portions, each representing a logical part of the program. Figure 44 on page 90 illustrates the design of the online program. ISPF calls are made from the `IBMOU00x`-routines, where *x* are between 1 and 6.

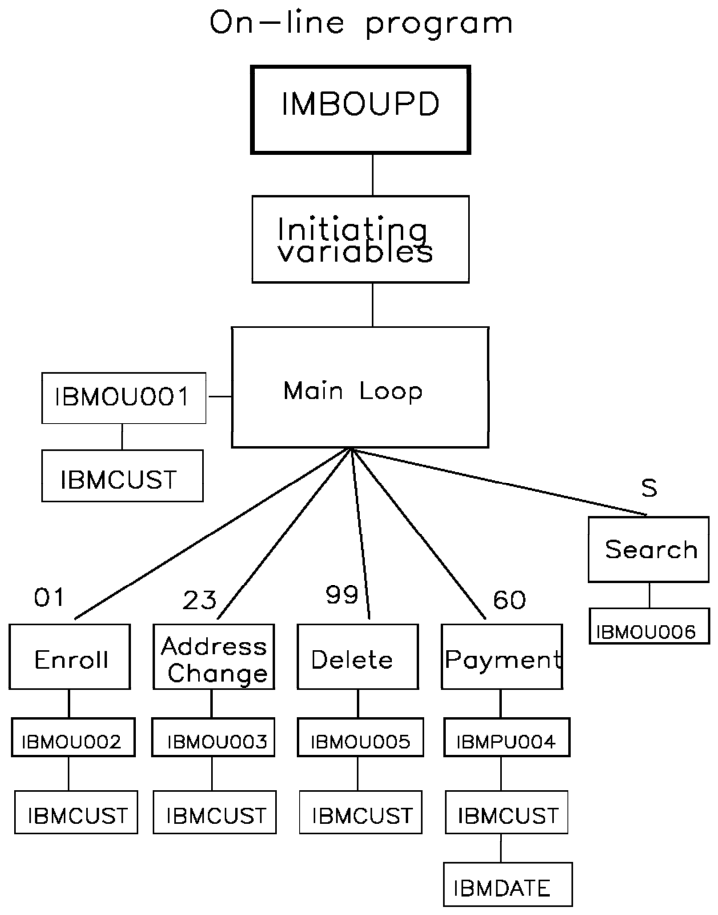


Figure 44. Online Program Logic

The program was organized into a main loop and five subsections. From the original code, most of the logic except for the ISPF panel interactions and some DB2 declarations were kept in the newer MVS and AIX versions. The original code also used two external assembler modules, **IBMCUST** and **IBMDATE**. These modules were rewritten in C so we could keep the logic intact. All routines except for the external ones were called with a `PERFORM` statement. Figure 45 on page 91 shows the main loop, which is a very tight loop consisting of only 14 lines of code (including two labels).

```
       PROCEDURE DIVISION.

       STARTA.
      *
           ACCEPT TODAY FROM DATE.

       A000.
           PERFORM PANEL-INIT.
       A100.
           PERFORM BLANKA-ALL.
           PERFORM PANEL-ANROP-IBMOU001.
           MOVE SPACES TO ERRMSG.
           IF LASTCC = 8 GO TO A999.
           IF ACT = "01" PERFORM ENROLLMENT.
           IF ACT = "23" PERFORM ADDRESS-CHANGE.
           IF ACT = "60" PERFORM PAYMENTS-SUB.
           IF ACT = "99" PERFORM DELETES-SUB.
           IF ACT = "S" PERFORM SEARCH-TAB.
           IF ACT = "s" PERFORM SEARCH-TAB.
           GO TO A100.
       A999.
           STOP RUN.
```

Figure 45. Main Loop

The original program was written in one module, with 1578 lines of code. For the new versions we split the program into smaller chunks of code, and utilized the `COPY` statement extensively to bring them together at compile time. With this approach, we could rearrange part of the code, or even replace part of it, but still keep the main logic intact. The next paragraph shows how we implemented this.

6.3.2 Implementation

In this section we work our way through the original program, showing what we did, and even showing some future design considerations. We do not work our way through the whole program, but rather focus on smaller sections to show the concept. This section examines the following:

- DB2/6000 related changes
- The main loop
- The `ADDRESS-CHANGE SECTION`
- The use of `COPY` files
- The user interface interactions
- Future concepts.

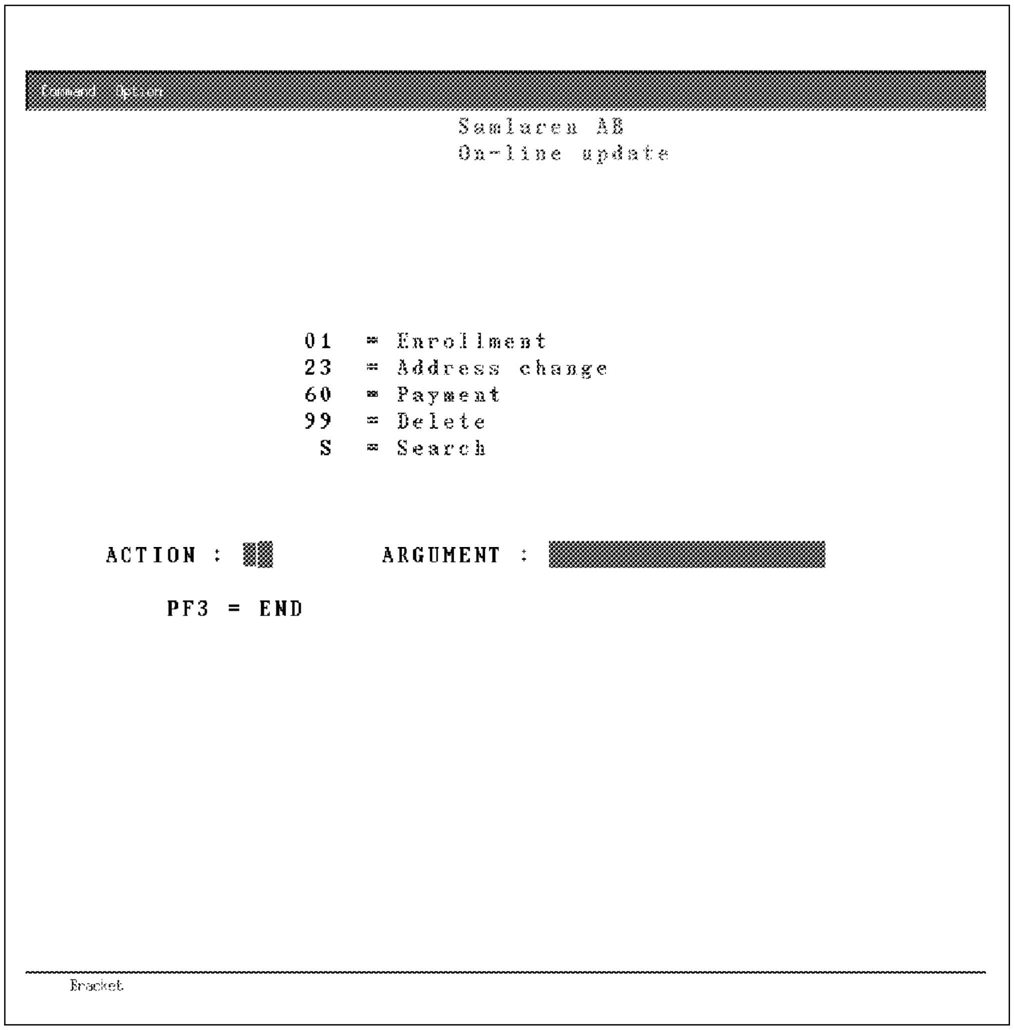
6.3.2.1 DB2/6000 Related Changes

The DB2 interface of the legacy application ported transparently. There were a few statements that were appropriate only for MVS, but these were treated as comments by the COBOL compiler on AIX. No code changes were made related to DB2/6000 at all. Version 3.1.3 of Micro Focus COBOL for AIX, and subsequent releases, include support for DB2/6000. We used the compiler directives **SQLDB “dbname”**, **SQLDB2**, and **IBMCOMP** to compile our IBM VS COBOL II source that included embedded SQL. The SQLDB directive parameter is the name of the database to which our program must connect.

6.3.2.2 The Main Loop

The biggest changes to the program affected the main loop. As shown in Figure 45, the main loop consisted of only 14 lines of code. When we implemented the GUI, the COBOL program was no longer the *main* program. Instead, portions of the COBOL program are called by *callbacks* when user actions trigger them through the user interface. In the original version, the main loop revolved around the main panel (IBMOU001), shown in Figure 46 on page 92.

The new main panel looked almost the same, but was no longer implemented by the COBOL code.



```
                              Samlaren AB
                              On-line update



                    01  = Enrollment
                    23  = Address change
                    60  = Payment
                    99  = Delete
                     S  = Search


     ACTION :             ARGUMENT :

         PF3 = END
```

Figure 46. Main Panel

To enable the COBOL program to be called by other programs, in our case C programs, we had to create a `LINKAGE SECTION`. The `LINKAGE SECTION` contained all the variables that are used when calling the COBOL program from another program, refer to Figure 78 on page 123. One thing to keep in mind, when dealing with interlanguage calls, especially between COBOL and C, is that C terminates strings with a null character (hexadecimal `X'00'`), whilst COBOL does not. This can be dealt with in a couple of ways, either by declaring string variables in the manner shown in Figure 47 or by treating the variables as memory variables in C.

```
WORKING-STORAGE SECTION.
01  STR.
   03  STR-TEXT        PIC X(10).
   03  FILLER          PIC X VALUE X"00".
```

Figure 47. Declaration of Variables to Be Passed Between C and COBOL

The same methods are valid for COBOL programs calling C programs.

The next step was to create the ENTRY points in the COBOL program. The statements in the original version, shown in Figure 48, were replaced by separate ENTRY points.

```
        A000.
            IF ACT = "01" PERFORM ENROLLMENT.
            IF ACT = "23" PERFORM ADDRESS-CHANGE.
            IF ACT = "60" PERFORM PAYMENTS-SUB.
            IF ACT = "99" PERFORM DELETES-SUB.
            IF ACT = "S" PERFORM SEARCH-TAB.
            IF ACT = "s" PERFORM SEARCH-TAB.
```

Figure 48. Legacy Statements to Be Replaced by ENTRY Statements

The last two statements were replaced by one ENTRY point. So, the replaced main loop looked like Figure 49.

```
        PROCEDURE DIVISION.

        STARTA.
       *
            ACCEPT TODAY FROM DATE.

        A000.
            EXEC SQL CONNECT TO AIXDBM IN SHARE MODE END-EXEC.
            EXIT PROGRAM.

       * Here follows the different Entry points.
        ENTRY "enroll" USING CHANGE.
           ...
        EXIT PROGRAM.
       *
        ENTRY "change" USING CHANGE.
            MOVE SPACES TO ERRMSG.
            PERFORM ADDRESS-CHANGE.
            MOVE SQLCODE TO SQLC IN CHANGE.
        EXIT PROGRAM.
       *
        ENTRY "payment" USING CHANGE.
           ...
        EXIT PROGRAM.
       *
        ENTRY "delete" USING CHANGE.
           ...
        EXIT PROGRAM.
       *
        ENTRY "search" USING CHANGE.
           ...
        EXIT PROGRAM.
       *
```

Figure 49. Code Changes for Use of ENTRY Points

6.3.2.3 The ADDRESS-CHANGE SECTION

Figure 50 on page 94 shows the original version of the ADDRESS-CHANGE SECTION.

```
*****************************
*
* ADDRESS-CHANGE PROCEDURE
*
*****************************
* THIS PROCEDURE MAKES ADDRESS CHANGES
 ADDRESS-CHANGE SECTION.
* CHECK IF CUSTOMER EXISTS
 DB000.
     PERFORM PANEL-ANROP-IBMOU003.
     MOVE SPACES TO ERRMSG.
     IF LASTCC = 8
        MOVE ZERO TO LASTCC
        GO TO DB999.
     CALL "IBMCUST" USING PCUSTNO, NCUSTNO, ERRMSG.
     IF ERRMSG IS NOT = " " GO TO DB000.
     MOVE NCUSTNO TO WCUSTNO.
     INSPECT PZIPCODE TALLYING COUNTX FOR CHARACTERS
        REPLACING ALL SPACES BY ZEROS.
     MOVE NZIPCODE TO WZIPCODE.
     MOVE SPACES TO STR.
     MOVE PSTREET TO STREET IN ROAD.
     PERFORM STR-INSPECT.
     MOVE STREET IN ROAD TO PSTREET.
     MOVE SPACES TO STR.
     IF PSTREET IS NOT = STREET IN PGM-NAME OR
        WZIPCODE IS NOT = ZIPCODE IN PGM-NAME
        IF WCUSTNO = CUSTNO IN PGM-CUST
           GO TO DB200.
 DB100.
     EXEC SQL
        WHENEVER SQLERROR GO TO DB990
     END-EXEC.
     EXEC SQL
        SELECT CUSTNO, REFNO, MAILID, SOURCECODE
           INTO :PGM-CUST.CUSTNO,
                :PGM-CUST.REFNO,
                :PGM-CUST.MAILID,
                :PGM-CUST.SOURCECODE
           FROM IBPED.CUST
           WHERE CUSTNO = &column.WCUSTNO
     END-EXEC.
     IF SQLCODE = +100
        MOVE "CUSTOMER DOES NOT EXISTS" TO ERRMSG
        GO TO DB000.
 DB150.
     EXEC SQL
        SELECT CUSTNO, FIRSTNAME, LASTNAME, STREET, ZIPCODE
           INTO :PGM-NAME.CUSTNO,
                :PGM-NAME.FIRSTNAME,
                :PGM-NAME.LASTNAME,
                :PGM-NAME.STREET,
                :PGM-NAME.ZIPCODE
           FROM IBPED.NAME
           WHERE CUSTNO = :PGM-CUST.CUSTNO
     END-EXEC.
     EXEC SQL
        SELECT ZIPCODE, CITY
           INTO :PGM-ZIP.ZIPCODE,
                :PGM-ZIP.CITY
           FROM IBPED.ZIP
           WHERE ZIPCODE = :PGM-NAME.ZIPCODE
     END-EXEC.
     MOVE CUSTNO IN PGM-CUST TO PCUSTNO.
     MOVE REFNO IN PGM-CUST TO PREFNO.
     MOVE MAILID IN PGM-CUST TO PMAILID.
     MOVE SOURCECODE IN PGM-CUST TO PSRC.
     MOVE FIRSTNAME IN PGM-NAME TO PFNAME.
     MOVE LASTNAME IN PGM-NAME TO PLNAME.
     MOVE STREET IN PGM-NAME TO PSTREET.
     MOVE ZIPCODE IN PGM-NAME TO PZIPCODE.
     MOVE CITY IN PGM-ZIP TO PCITY.
     GO TO DB000.
```

Figure 50 (Part 1 of 2). Original ADDRESS-CHANGE SECTION of Legacy Application

```
        DB200.
            IF WZIPCODE = ZIPCODE IN PGM-NAME
               GO TO DB400.
            MOVE WZIPCODE TO ZIPCODE IN PGM-NAME.
       * CHECK IF VALID ZIPCODE
            EXEC SQL
               SELECT ZIPCODE, CITY
                  INTO :PGM-ZIP.ZIPCODE, :PGM-ZIP.CITY
                  FROM IBPED.ZIP
                  WHERE ZIPCODE = &column.WZIPCODE
            END-EXEC.
            IF SQLCODE = +100
               MOVE "MISSING" TO CITY IN PGM-ZIP
               MOVE "ZIPCODE DOES NOT EXISTS IN ZIP-TABLE" TO ERRMSG.
       * IF NOT CREATE NEW RECORD IN THE ZIP-TABLE
            MOVE WZIPCODE TO ZIPCODE IN PGM-ZIP.
            IF CITY IN PGM-ZIP = "MISSING"
               EXEC SQL
                  INSERT INTO IBPED.ZIP
                     (ZIPCODE, CITY)
                     VALUES (:PGM-ZIP.ZIPCODE, :PGM-ZIP.CITY)
               END-EXEC
               EXEC SQL
                  COMMIT WORK
               END-EXEC.
            MOVE CITY IN PGM-ZIP TO PCITY.
       * UPDATE THE NAME-TABLE
        DB400.
            MOVE PSTREET TO STREET IN PGM-NAME.
            MOVE WZIPCODE TO ZIPCODE IN PGM-NAME.
            EXEC SQL
               UPDATE IBPED.NAME
                  SET    STREET = :PGM-NAME.STREET,
                         ZIPCODE = :PGM-NAME.ZIPCODE
                  WHERE  CUSTNO = :PGM-NAME.CUSTNO
            END-EXEC.
            EXEC SQL
               COMMIT WORK
            END-EXEC.
            MOVE "ADDRESS HAS BEEN UPDATED" TO ERRMSG.
            GO TO DB000.
        DB990.
            MOVE SQLERRMC TO ERRMSG.
            GO TO DB000.
        DB999.
            EXIT.
```

Figure 50 (Part 2 of 2). Original ADDRESS-CHANGE SECTION of Legacy Application

In our new AIX version, implemented with a GUI, there was a small rearrangement of this code, and the statements shown in Figure 51 were removed.

```
            PERFORM PANEL-ANROP-IBMOU003.
            MOVE SPACES TO ERRMSG.
            IF LASTCC = 8
               MOVE ZERO TO LASTCC
               GO TO DB999.
```

Figure 51. Statements Deleted from Legacy Application

These lines were related to the ISPF interface and were thus no longer needed. The reader might notice that the line shown in Figure 52 was moved to the `ENTRY` point shown in Figure 49 on page 93 instead.

```
            MOVE SPACES TO ERRMSG.
```

Figure 52. Resetting ERRMSG Variable Moved in Legacy Application

This rearrangement affects the label `DB000`. This label played a central part as a return point to the ISPF panel in the original version. By moving this label to the end of the `ADDRESS-CHANGE SECTION` we solved a lot of potential problems, and created possibilities of future solutions, such as using only one main skeleton for the generation and testing of both MVS and AIX code. This is shown in Figure 53.

```
*****************************
*
* ADDRESS-CHANGE PROCEDURE
*
*****************************
* THIS PROCEDURE MAKES ADDRESS CHANGES
 ADDRESS-CHANGE SECTION.
* CHECK IF CUSTOMER EXISTS
     CALL "IBMCUST" USING PCUSTNO, NCUSTNO, ERRMSG.
     .
     .
     .
 DB999.
 DB000.
     EXIT.
```

Figure 53. New Location for DB000 Label in Legacy Application

6.3.2.4 Use of Copy Files

Typically, COPY files are used for code which is to be shared among several programs. It is a means of modularizing your code used in many languages. We anticipated using COPY files to isolate the changes we made to the COBOL application specifically to support the AIX version. By removing this platform-specific code from the main file, we laid the groundwork for the possibility of later sharing this main module between the MVS and AIX versions of the application. This also opened up the potential to isolate differences between the two planned AIX versions as well. It also broke up the very large COBOL application into units that were more reasonable to manipulate. The following sections were first moved to COPY files:

- Variable declarations
- Call structures
- The ENTRY points
- The ENROLLMENT SECTION
- The ADDRESS-CHANGE SECTION
- The PAYMENT-SUB SECTION
- The DELETE-SUB SECTION
- The SEARCH-TAB SECTION

If this idea were implemented, a future version of our program might look like Figure 54 on page 97.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. IBMOUPD.
       AUTHOR. LEIF TRULSSON.
       DATE-WRITTEN. 930812.

      *REMARKS.
      *         DB2 ONLINE UPDATE PROGRAM.
      *
      *

       ENVIRONMENT DIVISION.

       DATA DIVISION.

       WORKING-STORAGE SECTION.
      *
           EXEC SQL INCLUDE SQLCA END-EXEC.
      *
      * Copy file for SQL-environment variables
           COPY 'sqlenv.cbl' IN '/u/db2/sqllib/include'.
      * Copy file for Level-77 variables
           COPY 'level77var.cbl' .
      * Copy file for Host-variables
           EXEC SQL INCLUDE 'hostvarsql.cbl' END-EXEC.
      * Copy file for Common variables
           COPY 'commonvar.cbl' .
      * Copy file for PANEL VARIABLES
           COPY 'panvarnew.cbl' .
      * Copy file for Call-structures
           COPY 'enroll.cbl' .
           COPY 'change.cbl' .
           COPY 'payment.cbl' .
           COPY 'delete.cbl' .
           COPY 'search.cbl' .
      *
       PROCEDURE DIVISION.

       STARTA.
      *
           ACCEPT TODAY FROM DATE.
      *
           COPY 'entry.cbl' .

      *****************************
      *
      * BLANKA-ALL PROCEDURE
      *
      *****************************
      * THIS PROCEDURE BLANKS ALL PANEL AND TABLE VARIABLES
       BLANKA-ALL SECTION.
           COPY 'blanka.cbl' .
      *****************************
      *
      * ENROLLMENT PROCEDURE
      *
      *****************************
      * THIS PROCEDURE ENROLLS NEW CUSTOMERS
       ENROLLMENT SECTION.
           EXEC SQL INCLUDE 'enrollsql.cbl' END-EXEC.
       DA000.
           EXIT.
      *****************************
      *
      * ADDRESS-CHANGE PROCEDURE
      *
      *****************************
      * THIS PROCEDURE MAKES ADDRESS CHANGES
       ADDRESS-CHANGE SECTION.
           EXEC SQL INCLUDE 'addrchgsql.cbl' END-EXEC.
       DB000.
           EXIT.
```

Figure 54 (Part 1 of 2). AIX Version of the Legacy Application

```
*****************************
*
* PAYMENT PROCEDURE
*
*****************************
* THIS PROCEDURE CREATES RECORD IN PAYMENT TABLE
 PAYMENTS-SUB SECTION.
     EXEC SQL INCLUDE 'paysql.cbl' END-EXEC.
 DC000.
     EXIT.
*****************************
*
* PAYMENT-RECORD PROCEDURE
*
*****************************
* THIS PROCEDURE RETREIVES THE TABLE-VALUES FOR THE PAYMENTS
* AND DELETES PANELS
 PAYMENT-RECORD SECTION.
     EXEC SQL INCLUDE 'payrecsql.cbl' END-EXEC.
*****************************
*
* DELETES PROCEDURE
*
*****************************
* THIS PROCEDURE DELETES CUSTOMERS
 DELETES-SUB SECTION.
     EXEC SQL INCLUDE 'deletesql.cbl' END-EXEC.
 DD000.
     EXIT.
*****************************
*
* SEARCH-TAB PROCEDURE
*
*****************************
* THIS PROCEDURE SEARCHES APPROPRIATE TABEL ACCORDING TO
* SEARCH ARGUMENT
 SEARCH-TAB SECTION.
     EXEC SQL INCLUDE 'searchsql.cbl' END-EXEC.
 DE100.
     EXIT.
*****************************
*
* STR-INSPECT PROCEDURE
*
*****************************
 STR-INSPECT SECTION.
     COPY 'strinsp.cbl' .
```

Figure 54 (Part 2 of 2). AIX Version of the Legacy Application

The `EXEC SQL INCLUDE 'filename' END-EXEC.` would replace the `COPY` statement for the part of the code containing SQL-syntax.

6.3.2.5 The User Interface Interactions

As we said earlier, the new COBOL version does not handle any direct user interactions. But, most of the original ISPF panel variables were still used in the new version by our program to pass information from the database to the new user interface code. We also wanted to be able to test our COBOL portion of the application in a stand-alone mode (especially while the GUI was being concurrently developed). We also planned on continuing the development of new MVS versions of the program on AIX, which presented us with the need to be able to test those versions that would continue to have an ISPF user interface.

For the purpose of testing the `ADDRESS-CHANGE SECTION` in the new AIX version of our program, we created a C program called *testchg.c*, that called the `change` entry. This program is discussed in more detail in 6.3.4, "Test Tools" on page 113 .

To be able to test the MVS version, we had to rewrite the ISPF calls, or we could try to find an ISPF emulator for AIX. We managed to find an IBM Internal Tool that gave us limited ISPF support on AIX. We also looked at rewriting the panel interactions with the Enhanced ACCEPT/DISPLAY and SCREEN SECTION support, as supported by Micro Focus COBOL. We show this in more detail in 6.3.4, "Test Tools" on page 113. We could also have used another Micro Focus tool to build our *new* AIX panels. This tool is the Dialog System from Micro Focus.

6.3.2.6 Future Concepts

As mentioned earlier, we might want to be able to maintain only one main skeleton module, but separate, parallel sets of COPY files with the platform-specific code in them. This possibility depends on the notion that a file naming convention can be devised that is acceptable to both MVS and AIX, and that enables the Micro Focus COBOL compiler to distinguish the AIX from MVS files depending on the build instructions. We explored this possibility and concluded that the following scheme should work.

The Micro Focus COBOL compiler supports the **COPYEXT** compiler directive. This directive enables us to specify the file name extension of the AIX COPY files differently from the MVS COPY files on the compiler invocation command line. This compiler also enables us to cite only the common base portion of the file name in the source code at the COPY statement. Using this directive on AIX, we can compile a version destined for both platforms for purpose of debugging with the Micro Focus Animator.

The MVS compiler does not support this directive, because MVS itself does not support the notion of file base names with file name extensions. However, this was not a problem if the base name (which was the entire file name as far as MVS was concerned) was entered on the COPY statement in the file. The build automation program can ensure that the files get named properly during the upload to MVS.

We decided to store MVS COPY code in files with the **.cpy** extension and AIX modules in files with the **.cbl** extension. Figure 55 on page 100 illustrates this concept of using different compiler directives and file name extensions to maintain one main skeleton for the two platform versions.

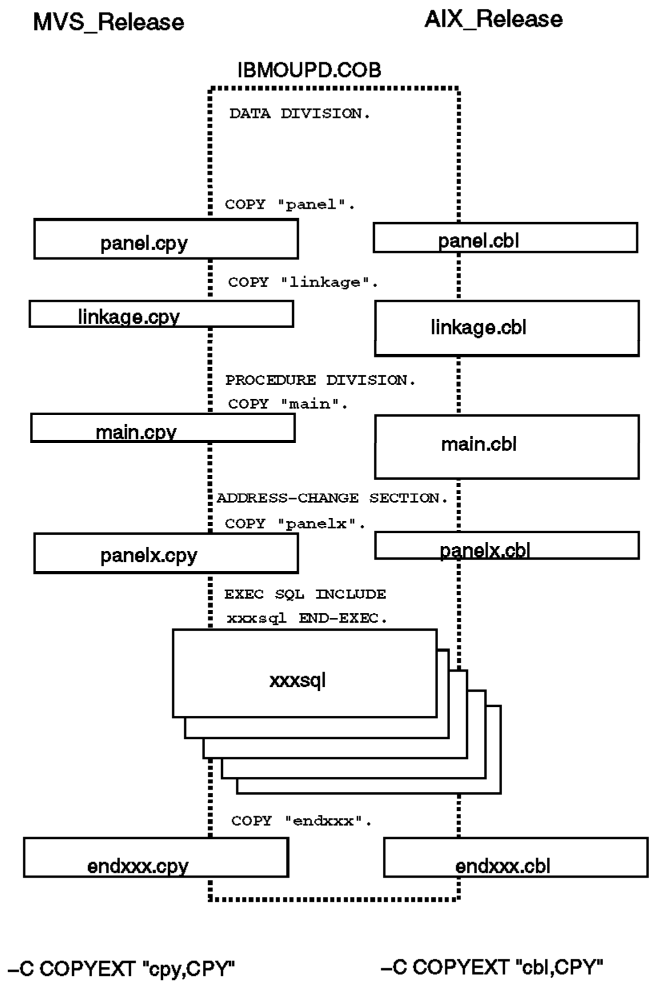
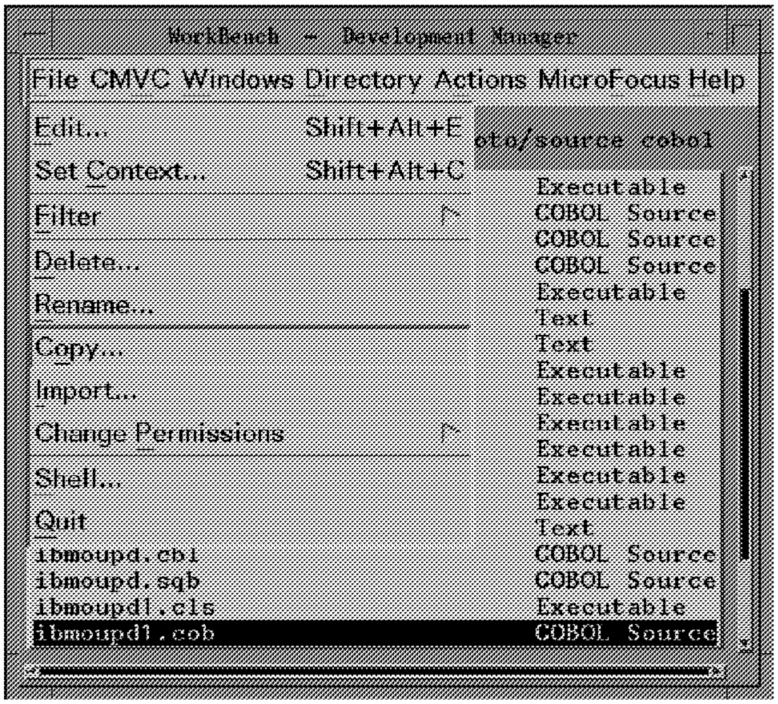


Figure 55. Future Concepts Using Compiler Directives

6.3.3 Edit, Compile, and Debug

In this section, we show how the AIX AD products supported the edit, compile, and debug cycle while we were migrating the COBOL code to AIX. The first thing we did was to create a new main module for the MVS_Release_1 and one for the AIX_Release_1. We did this through SDE WorkBench/6000 with the help of Program Editor. To save us time, we copied the original file. We used Development Manager to do this, and the procedure is shown in Figure 56 on page 101 and in Figure 57 on page 101.

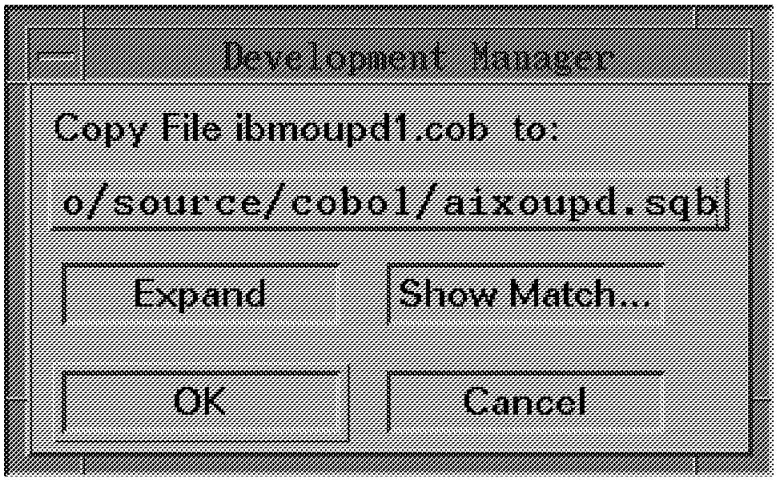
First, we selected the file to copy, by clicking once with the left mouse button on the line displaying the name of the file we wanted to copy. This highlighted the line. By clicking the left mouse button once, while the cursor was over the `File` item in the Action bar of `WorkBench - Development Manager`, we got the pull-down menu shown in Figure 56 on page 101.


WorkBench - Development Manager
File CMVC Windows Directory Actions MicroFocus Help
Edit... Shift+Alt+E
Set Context... Shift+Alt+C
Filter
Delete...
Rename...
Copy...
Import...
Change Permissions
Shell...
Quit

```
oto/source cobol
                         Executable
                         COBOL Source
                         COBOL Source
                         COBOL Source
                         Executable
                         Text
                         Text
                         Executable
                         Executable
                         Executable
                         Executable
                         Executable
                         Executable
                         Text
ibmoupd.cbl              COBOL Source
ibmoupd.sqb              COBOL Source
ibmoupd1.cls             Executable
ibmoupd1.cob             COBOL Source
```

Figure 56. Copying a File with the Development Manager

From this pull-down menu, we selected Copy. Next we got the window shown in Figure 57.



Copy File ibmoupd1.cob to:

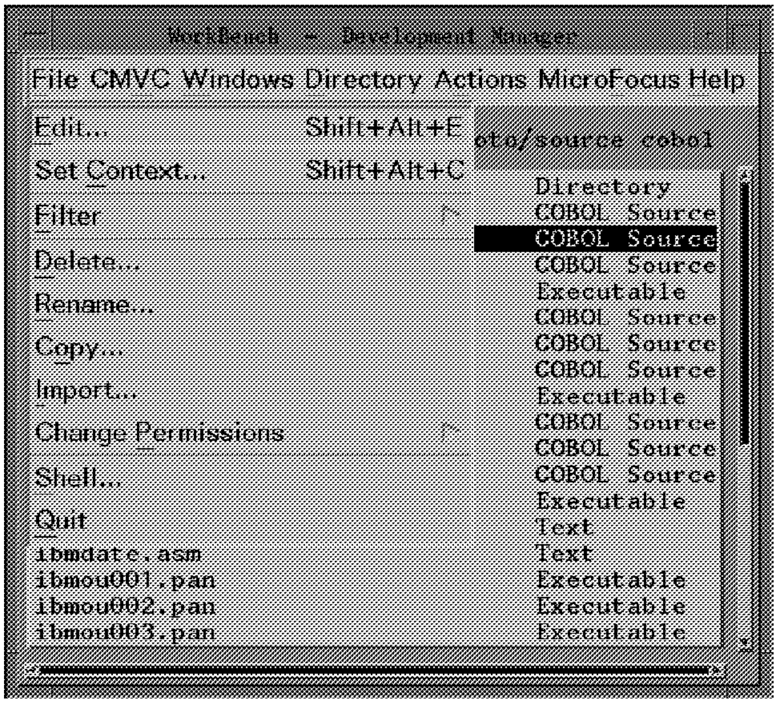
o/source/cobol/aixoupd.sqb

Figure 57. Development Manager Copy File Window

In this window we entered the name of our new file, and then we clicked on OK. The file was copied, and it was time to edit it.

6.3.3.1 Editing the Main module

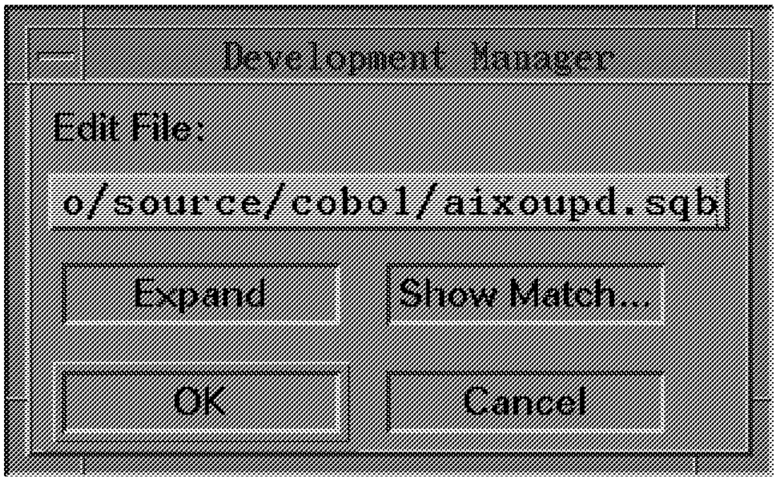
To invoke the editor from the WorkBench Development Manager, we can click twice with the left mouse button on the line, displaying the name of the file to be edited, or select the line displaying the name, and then from the `File` pull-down menu select the `Edit` option, as shown in Figure 58.


WorkBench - Development Manager
File CMVC Windows Directory Actions MicroFocus Help
Edit... Shift+Alt+E
Set Context... Shift+Alt+C
Filter
Delete...
Rename...
Copy...
Import...
Change Permissions
Shell...
Quit

```
oto/source cobol
                    Directory
                    COBOL Source
                    COBOL Source
                    COBOL Source
                    Executable
                    COBOL Source
                    COBOL Source
                    COBOL Source
                    Executable
                    COBOL Source
                    COBOL Source
                    COBOL Source
                    Executable
                    Text
ibmdate.asm         Text
ibmou001.pan        Executable
ibmou002.pan        Executable
ibmou003.pan        Executable
```

Figure 58. To Edit a File Using SDE WorkBench/6000

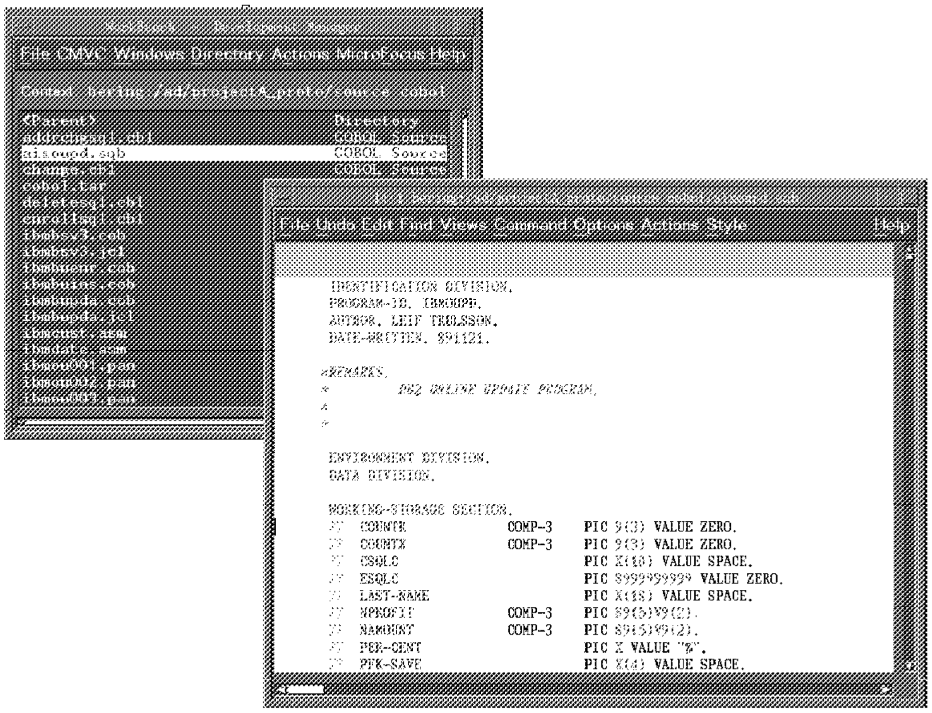
If you select the `Edit` option you get a new window as shown in Figure 59. If you want to continue, you just select `OK`, or press ENTER.



Edit File: o/source/cobol/aixoupd.sqb

Figure 59. Development Manager Edit File Window

This should open up the Program Editor edit window as shown in Figure 60 on page 103.


File CMVC Windows Directory Actions MicroFocus Help
File Undo Edit Find Views Command Options Actions Style Help

```
Context being /ad/projectA_proto/source cobol
<Parent>                         Directory
addrchgsql.cbl                   COBOL Source
aisoupd.sqb                      COBOL Source
change.cbl                       COBOL Source
cobol.tar
deletesql.cbl
enrollsql.cbl
ibmbsv3.cob
ibmbsv3.jcl
ibmbuenr.cob
ibmbuins.cob
ibmbupda.cob
ibmbupda.jcl
ibmcust.asm
ibmdate.asm
ibmou001.pan
ibmou002.pan
ibmou003.pan
```

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. IBMOUPD.
       AUTHOR. LEIF TROLSSON.
       DATE-WRITTEN. 891121.

      *REMARKS.
      *         DB2 ONLINE UPDATE PROGRAM.
      *
      *

       ENVIRONMENT DIVISION.
       DATA DIVISION.

       WORKING-STORAGE SECTION.
       77  COUNTR            COMP-3   PIC 9(3) VALUE ZERO.
       77  COUNTX            COMP-3   PIC 9(3) VALUE ZERO.
       77  CSQLC                      PIC X(10) VALUE SPACE.
       77  ESQLC                      PIC S99999999 VALUE ZERO.
       77  LAST-NAME                  PIC X(18) VALUE SPACE.
       77  NPROFIT           COMP-3   PIC S9(5)V9(2).
       77  NAMOUNT           COMP-3   PIC S9(5)V9(2).
       77  PER-CENT                   PIC X VALUE "%".
       77  PFK-SAVE                   PIC X(4) VALUE SPACE.
```

Figure 60. Editing the New Main Module

Development Manager and Program Editor both recognize the specific file type by examining the files extension. The files extensions that are recognized by default for COBOL source files are:

- CBL
- cbl
- cob

For more information about how to support other file extensions such as COBOL source files, see 9.2.4, "Tailoring Program Editor" on page 206.

6.3.3.2 Compiling the COBOL Code

Because we wanted to evaluate SDE WorkBench/6000, we decided to use both Program Builder and the Micro Focus COBOL Toolbox for compilations. We therefore integrated both the Animator and the Micro Focus COBOL Toolbox with SDE WorkBench/6000 as described in 9.2, "Tailoring SDE WorkBench/6000 for COBOL Programmers" on page 202 .

The next paragraphs illustrate the compilation process, including the use of Micro Focus RTE, *makefile* files, and Micro Focus COBOL Toolbox.

Creating a New Micro Focus Run Time Environment: It is sometimes necessary to create a new Micro Focus RTE. For instance, if you want to Animate a COBOL program that makes calls to subroutines that are written in languages other than COBOL, then you have to include these routines in your Micro Focus RTE. Routines that are included in the Micro Focus RTE do not have to be specified at compile or link time.

The current Micro Focus RTE is located in the **$COBDIR** directory. Before you rebuild the Micro Focus RTE, you might want to rename the file containing the old one. To rebuild the Micro Focus RTE, you must log in as *root*, do **cd $COBDIR/src/rts**, and then run the script **mkrts** with the modules you want to include as parameters. For example, to link the two C language modules called by IBMOUPD, use:

```
mkrts $HOME/source/c/IMBCUST.c $HOME/source/c/IBMDATE.c
```

The new Micro Focus RTE will be created in the current directory. A *clean* Micro Focus RTE can always be created by running the **mkrts** script without any parameters.

The Micro Focus COBOL Version 3.1.3 also includes a script called **St2** that enables you to rebuild the Micro Focus RTE to include DB2/6000 support. Unfortunately, running this script ruins the previously created Micro Focus RTE that includes foreign subroutines. To be able to include both our own subroutines and the DB2/6000 support in the Micro Focus RTE, we created a script called **mkdb2rts**. This script is shown in Figure 61.

```
if test $# -lt 0
then
        echo "usage: mkdb2rts <prog.o> ffflffl<prog.o>“ .. “"
        exit 1
fi

INCFLAGS="-I sqlgintp -I sqlgcmpl -I sqlgahvr -I sqlginit -I sqlgstar   \
          -I sqlgfini -I sqlgaloc -I sqlgcall -I sqlgdloc -I sqlgsets   \
          -I sqlgsetv -I sqlgstop -I sqlgstrt -I sqlgusda -ldb2"

sh -x mkrts -vUDD ${INCFLAGS} $*
```

Figure 61. The mkdb2rts Script. This script rebuilds the Run-Time system with both DB2/6000 support and user defined routines

Compiling With Makefile Files: When building our *makefile* files for the testing of our COBOL programs, we decided to use a modified version of the sample *makefile* file shipped together with DB2/6000. This modified test *makefile* file is shown in Figure 62 on page 105.

```
#################################################################
# MAKEFILE for COBOL ibmoupd  Program                           #
# Enter the Following:                                          #
#                                                               #
#      make all     -- makes all programs                       #
#                                                               #
#      make ibmoupd -- makes ibmoupd       program              #
#      make IBMDATE -- makes IBMDATE       program              #
#      make IBMCUST -- makes IBMCUST       program              #
#                                                               #
#      make cleanup -- removes builds from all sample programs  #
#################################################################

DATABASE=aixdbm
DB2HOME=/u/$(DB2INSTANCE)
LINK_FLAGS=  -a -x -L$(DB2HOME)/sqllib/lib -ldb2
COMPILER=cob
CLIB=$(HOME)/source/c
CCOMP=cc
CC_FLAGS= -c -g
COPY = ibmpan1.cbl ibmpanel2.cbl ibmpanel3.cbl ibmpanel4.cbl \
       ibmpanel5.cbl ibmpanel6.cbl

all : IBMCUST.o IBMDATE.o ibmoupd

cleanup :
       rm -f ibmoupd ibmoupd.bnd ibmoupd.o ibmoupd.cbl ibmoupd.int ibmoupd.lst 2>/dev/null

ibmoupd.cbl : ibmoupd.sqb
      db2 CONNECT TO $(DATABASE)         && \
      db2 PREP ibmoupd.sqb BINDFILE      && \
      db2 BIND ibmoupd.bnd               && \
      db2 CONNECT RESET

ibmoupd : ibmoupd.cbl IBMCUST.o IBMDATE.o $(COPY)
      $(COMPILER) $(LINK_FLAGS) $@.cbl IBMCUST.o IBMDATE.o

IBMCUST.o :  $(CLIB)/IBMCUST.c
      $(CCOMP) $(CC_FLAGS) $(CLIB)/IBMCUST.c

IBMDATE.o :  $(CLIB)/IBMDATE.c IBMCUST.o
      $(CCOMP) $(CC_FLAGS) $(CLIB)/IBMDATE.c IBMCUST.o
```

Figure 62. Test Makefile File

Depending on what build target you use at build time, you can do the following:

- Remove all COBOL executable and intermediate files.
- Build object files from our C files.
- Compile and link the final executable file.

We used the appropriate Micro Focus COBOL compiler directives as shown in Figure 63 on page 106.

```
#####################################################################
# MAKEFILE for COBOL ibmoupd  Program                              #
# Enter the Following:                                             #
#                                                                  #
#      make all     -- makes all programs                          #
#                                                                  #
#      make ibmoupd -- makes ibmoupd       program                 #
#      make IBMDATE -- makes IBMDATE       program                 #
#      make IBMCUST -- makes IBMCUST       program                 #
#                                                                  #
#      make cleanup -- removes builds from all sample programs     #
#####################################################################

DATABASE=aixdbm
DB2HOME=/u/$(DB2INSTANCE)
LINK_FLAGS=  -a -x -C SQLDB=$(DATABASE) SQLDB2 SQLPRE IBMCOMP
COMPILER=cob
CLIB=$(HOME)/source/c
CCOMP=cc
CC_FLAGS= -c -g
COPY = ibmpan1.cbl ibmpanel2.cbl ibmpanel3.cbl ibmpanel4.cbl \
       ibmpanel5.cbl ibmpanel6.cbl

all : IBMCUST.o IBMDATE.o ibmoupd

cleanup :
       rm -f ibmoupd ibmoupd.bnd ibmoupd.o ibmoupd.cbl ibmoupd.int ibmoupd.lst 2>/dev/null

ibmoupd : ibmoupd.cob IBMCUST.o IBMDATE.o $(COPY)
      $(COMPILER) $(LINK_FLAGS) $@.cob IBMCUST.o IBMDATE.o

IBMCUST.o :  $(CLIB)/IBMCUST.c
      $(CCOMP) $(CC_FLAGS) $(CLIB)/IBMCUST.c

IBMDATE.o :  $(CLIB)/IBMDATE.c IBMCUST.o
      $(CCOMP) $(CC_FLAGS) $(CLIB)/IBMDATE.c IBMCUST.o
```

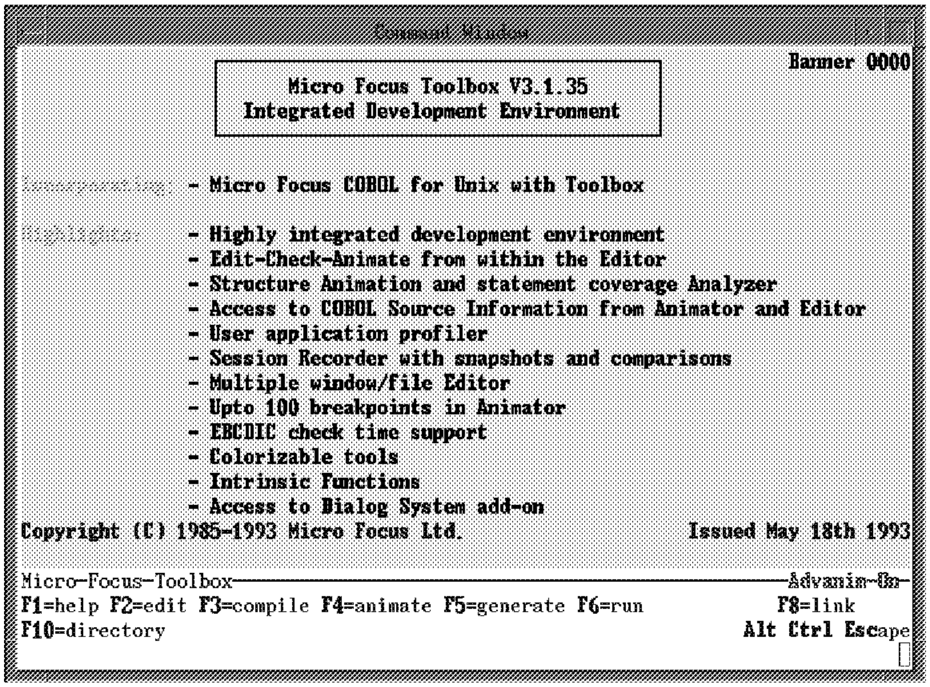
Figure 63. Micro Focus COBOL Version 3.1.3 Test Makefile File

In the early stages of our testing, we did not use CMVC to control our test *makefile* files. But as the environment stabilized, we also put the test *makefile* files under the control of CMVC.

Compiling with Micro Focus COBOL Toolbox: When compiling with the Micro Focus COBOL Toolbox, you do not have to worry about creating *makefile* files. Instead you just specify the compiler directives you want to be in effect.

If you did not have SDE WorkBench/6000 installed, you would invoke the Micro Focus COBOL Toolbox with the command **tbox** from any command shell. But as mentioned earlier, we decided to integrate the Micro Focus COBOL Toolbox with SDE WorkBench/6000. Refer to 9.2, “Tailoring SDE WorkBench/6000 for COBOL Programmers” on page 202 for more information about how to integrate the Micro Focus COBOL Toolbox with SDE WorkBench/6000. So, we invoked the Micro Focus COBOL Toolbox from the pull-down menu. But, before doing so, we set the context to the appropriate directory, in this case **$HOME/source/cobol**. Setting the context variable to the directory of the program to compile, will make life easier for us while we are inside the Micro Focus COBOL Toolbox environment.

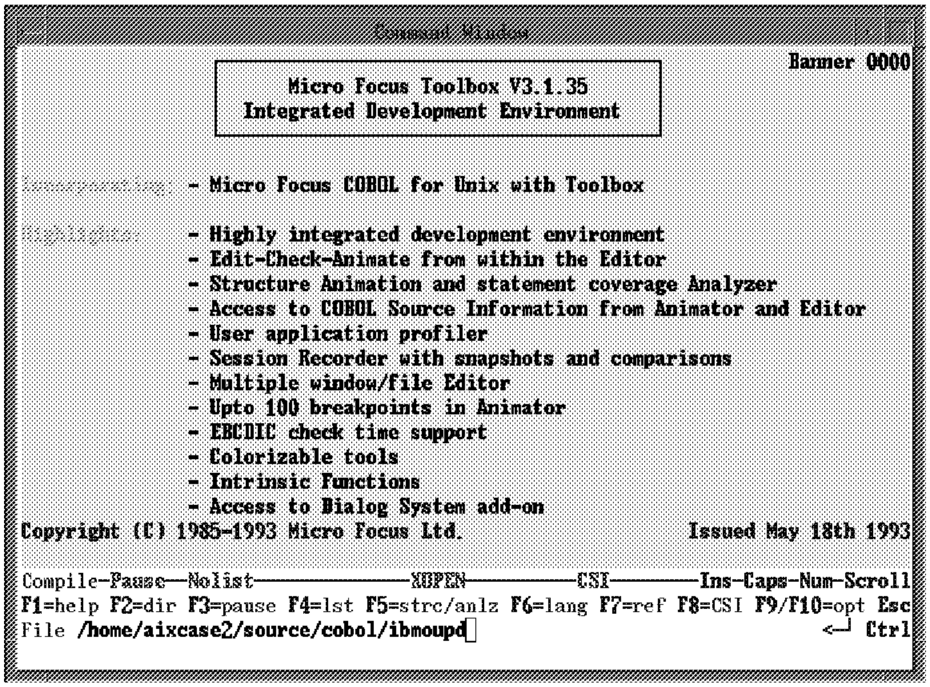
Invoking the Micro Focus COBOL Toolbox brings up a screen as shown in Figure 64 on page 107.



```
                              Command Window
                                                          Banner 0000
             Micro Focus Toolbox V3.1.35
         Integrated Development Environment

Incorporating:  - Micro Focus COBOL for Unix with Toolbox

Highlights:     - Highly integrated development environment
                - Edit-Check-Animate from within the Editor
                - Structure Animation and statement coverage Analyzer
                - Access to COBOL Source Information from Animator and Editor
                - User application profiler
                - Session Recorder with snapshots and comparisons
                - Multiple window/file Editor
                - Upto 100 breakpoints in Animator
                - EBCDIC check time support
                - Colorizable tools
                - Intrinsic Functions
                - Access to Dialog System add-on
Copyright (C) 1985-1993 Micro Focus Ltd.             Issued May 18th 1993

Micro-Focus-Toolbox--------------------------------------------Advanim-On-
F1=help F2=edit F3=compile F4=animate F5=generate F6=run         F8=link
F10=directory                                               Alt Ctrl Escape
```

Figure 64. Micro Focus COBOL Toolbox Main Window

From this window, you select which function to use by pressing the appropriate function key. The Alt and Ctrl menus are reached by entering either **/a** or **/c** and then pressing ENTER. To compile, we pressed F3 and a new window is displayed, as shown in Figure 65 on page 108.

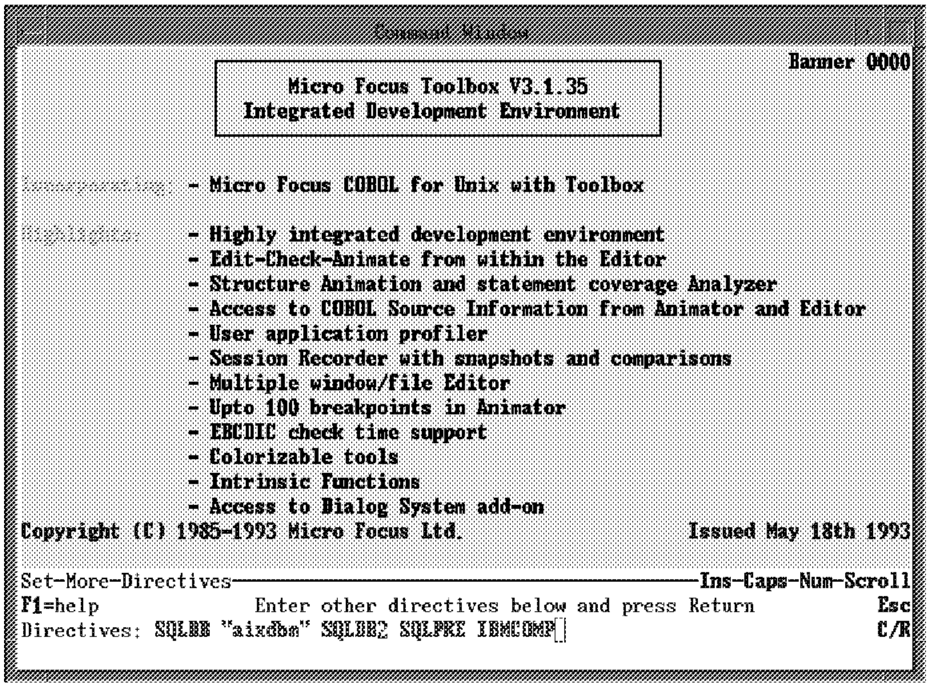


```
                                                               Banner 0000
                    Micro Focus Toolbox V3.1.35
                 Integrated Development Environment

Incorporating: - Micro Focus COBOL for Unix with Toolbox

Highlights:    - Highly integrated development environment
               - Edit-Check-Animate from within the Editor
               - Structure Animation and statement coverage Analyzer
               - Access to COBOL Source Information from Animator and Editor
               - User application profiler
               - Session Recorder with snapshots and comparisons
               - Multiple window/file Editor
               - Upto 100 breakpoints in Animator
               - EBCDIC check time support
               - Colorizable tools
               - Intrinsic Functions
               - Access to Dialog System add-on
Copyright (C) 1985-1993 Micro Focus Ltd.                   Issued May 18th 1993

Compile-Pause—Nolist—————————————XOPEN——————————CSI—————————Ins-Caps-Num-Scroll
F1=help F2=dir F3=pause F4=lst F5=strc/anlz F6=lang F7=ref F8=CSI F9/F10=opt Esc
File /home/aixcase2/source/cobol/ibmoupd                                 <─┘ Ctrl
```

Figure 65. Micro Focus COBOL Toolbox Compile Window

If we had already set the context to the directory of our program, or if we had already positioned ourselves to this directory before invoking the Micro Focus COBOL Toolbox from a command shell, the *path* would already be displayed, and all we have to do is to enter the name of the program, as shown in Figure 65.

To tell the compiler what compiler directives to use, press F9 and then F10. A new window as shown in Figure 66 on page 109 appears.



```
                            Command Window
                                                          Banner 0000
              Micro Focus Toolbox V3.1.35
           Integrated Development Environment

Incorporating:  - Micro Focus COBOL for Unix with Toolbox

Highlights:     - Highly integrated development environment
                - Edit-Check-Animate from within the Editor
                - Structure Animation and statement coverage Analyzer
                - Access to COBOL Source Information from Animator and Editor
                - User application profiler
                - Session Recorder with snapshots and comparisons
                - Multiple window/file Editor
                - Upto 100 breakpoints in Animator
                - EBCDIC check time support
                - Colorizable tools
                - Intrinsic Functions
                - Access to Dialog System add-on
Copyright (C) 1985-1993 Micro Focus Ltd.               Issued May 18th 1993

Set-More-Directives-------------------------------------Ins-Caps-Num-Scroll
F1=help           Enter other directives below and press Return          Esc
Directives: SQLDB "aixdbm" SQLDB2 SQLPRE IBMCOMP                         C/R
```

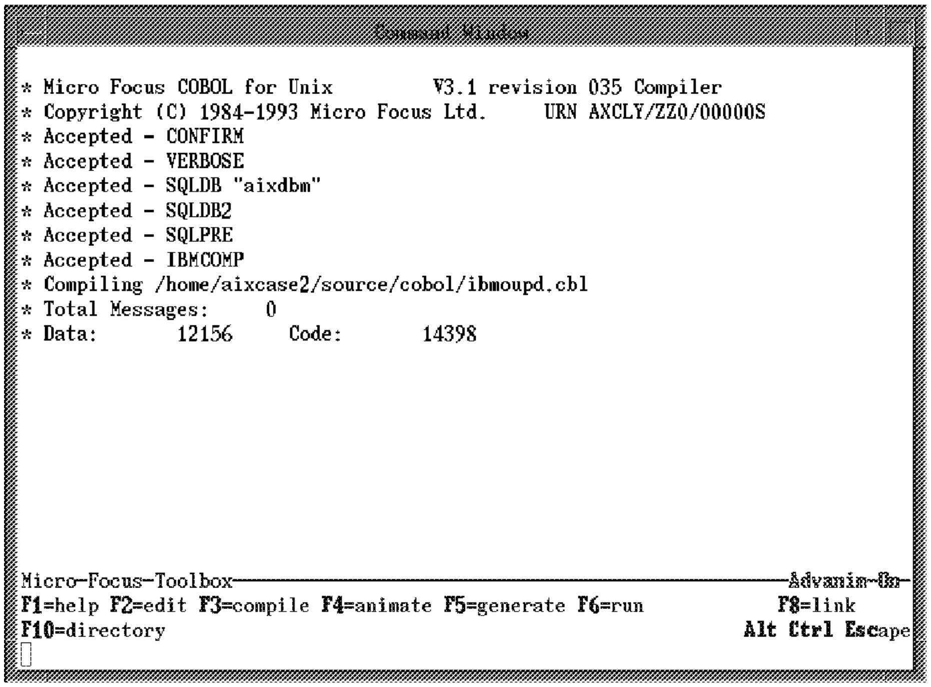
Figure 66. Micro Focus COBOL Toolbox Compiler Directives Window

The compiler directives we specified were:

SQLDB "aixdbm"[1]	Tells the compiler which database we want to connect to.
SQLDB2	Specifies that our program is using DB2 (MVS) syntax.
SQLPRE	This tells the compiler to precompile the SQL-code.
IBMCOMP	Turns on word-storage mode as used on IBM mainframes.

Figure 67 on page 110 shows the messages returned from a successful compilation.

1 Notice the difference in specifying the database, between command line compilation as shown in Figure 63 on page 106, and compiling with the Micro Focus COBOL Toolbox.



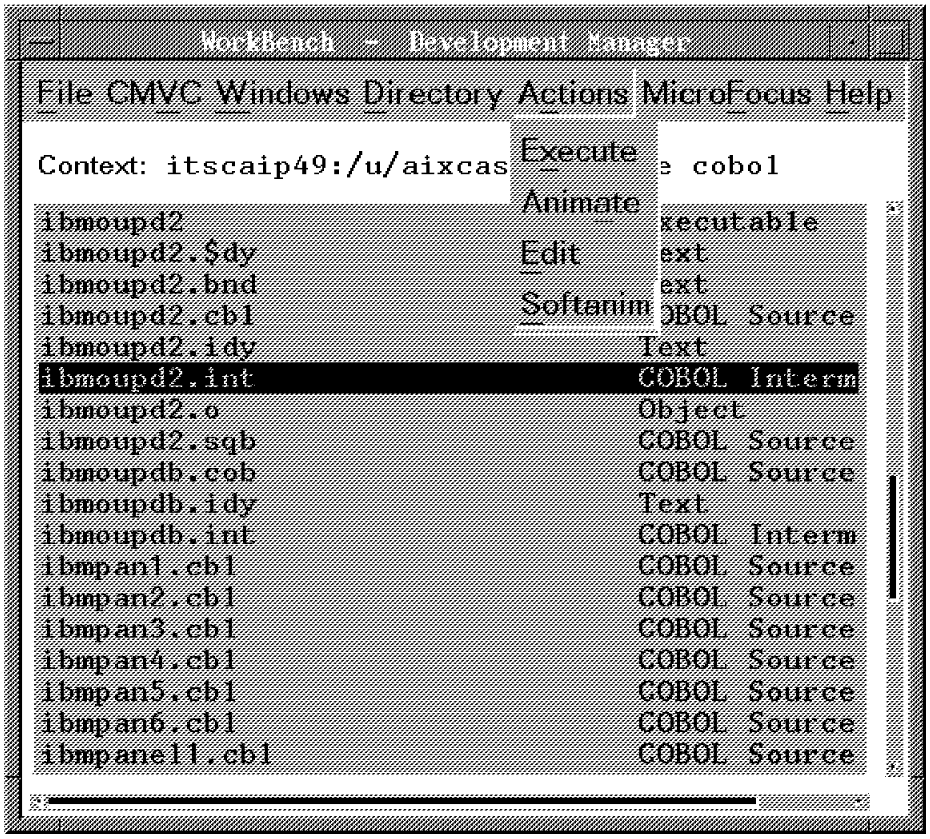
```
* Micro Focus COBOL for Unix         V3.1 revision 035 Compiler
* Copyright (C) 1984-1993 Micro Focus Ltd.     URN AXCLY/ZZ0/00000S
* Accepted - CONFIRM
* Accepted - VERBOSE
* Accepted - SQLDB "aixdbm"
* Accepted - SQLDB2
* Accepted - SQLPRE
* Accepted - IBMCOMP
* Compiling /home/aixcase2/source/cobol/ibmoupd.cbl
* Total Messages:     0
* Data:       12156     Code:       14398
```

Figure 67. Micro Focus COBOL Toolbox Compiler Message Window

6.3.3.3 Debugging the COBOL Code

Even without using the Micro Focus COBOL Toolbox to build our executable files, we still use the Micro Focus Animator for debugging the COBOL code. To be able to use the Animator on COBOL programs calling other language routines, in our case C routines, we have to rebuild the Micro Focus RTE to incorporate these foreign modules into the Micro Focus RTE. The procedure of how to rebuild the Micro Focus RTE is described in “Creating a New Micro Focus Run Time Environment” on page 103.

As we already integrated the Micro Focus Animator with SDE WorkBench/6000, we invoke the Animator from the Development Manager's Actions pull-down menu, as shown in Figure 68 on page 111.


WorkBench - Development Manager
File CMVC Windows Directory Actions MicroFocus Help
Execute
Animate
Edit
Softanim

Context: itscaip49:/u/aixcas ... cobol

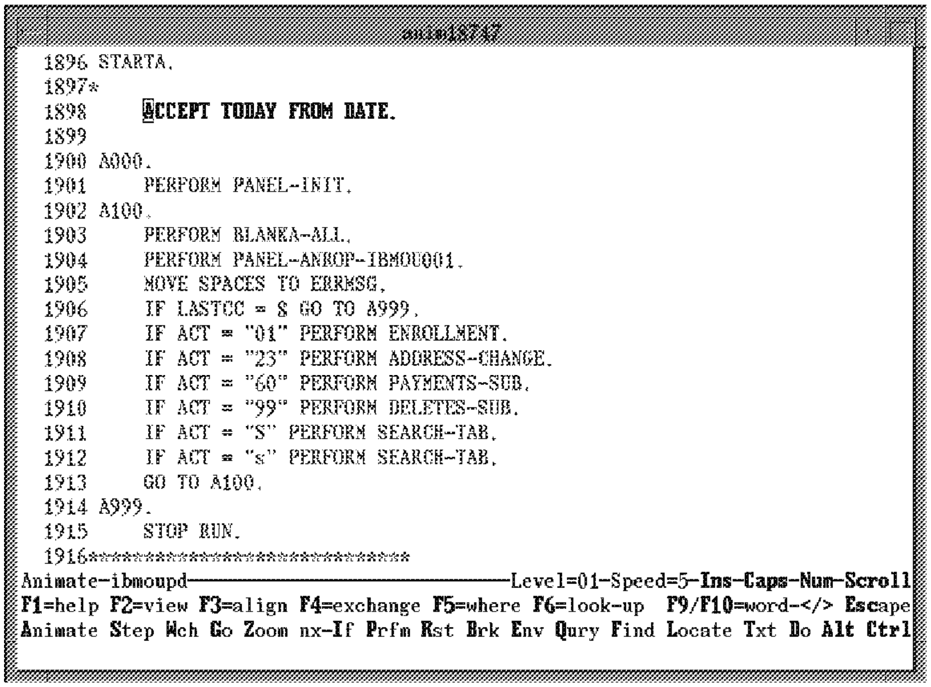
```
ibmoupd2                              ...xecutable
ibmoupd2.$dy                          ...ext
ibmoupd2.bnd                          ...ext
ibmoupd2.cbl                          ...OBOL Source
ibmoupd2.idy                          Text
ibmoupd2.int                          COBOL Interm
ibmoupd2.o                            Object
ibmoupd2.sqb                          COBOL Source
ibmoupdb.cob                          COBOL Source
ibmoupdb.idy                          Text
ibmoupdb.int                          COBOL Interm
ibmpan1.cbl                           COBOL Source
ibmpan2.cbl                           COBOL Source
ibmpan3.cbl                           COBOL Source
ibmpan4.cbl                           COBOL Source
ibmpan5.cbl                           COBOL Source
ibmpan6.cbl                           COBOL Source
ibmpanel1.cbl                         COBOL Source
```

Figure 68. Actions Pull-Down Menu for .int Files

The steps to invoke the Animator are:

1. Mark the COBOL intermediate file (the file with the extension .int).
2. Pull down the Actions menu in the WorkBench Development Manager window.
3. Select **Softanim**

Softanimator will open up a new window, load the intermediate code, and display the first lines of code in the programs `PROCEDURE DIVISION`, as shown in Figure 69 on page 112.



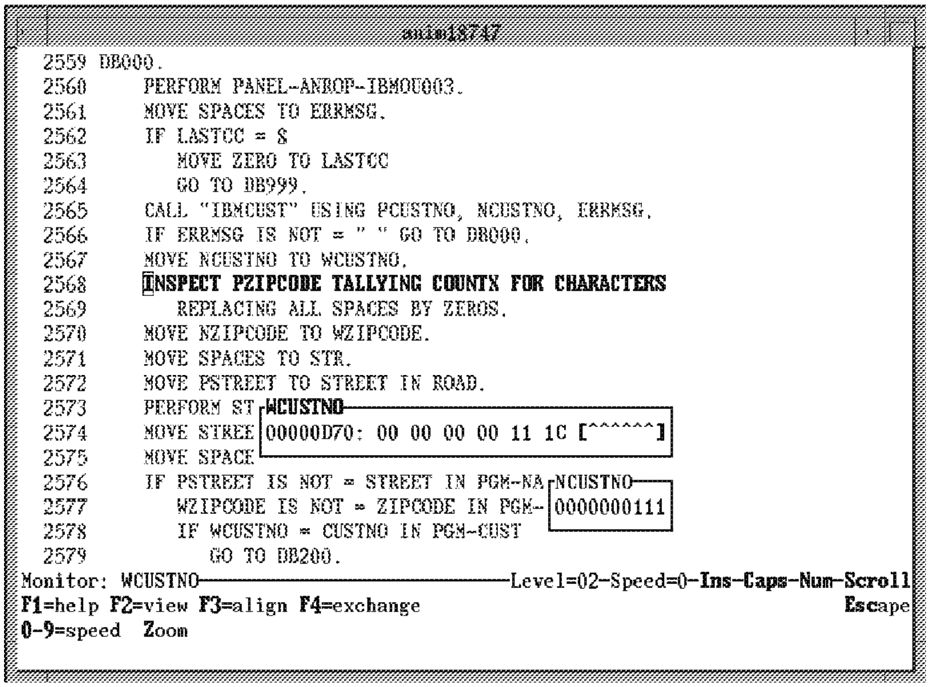
```
anim18747
STARTA.
*
      ACCEPT TODAY FROM DATE.

A000.
      PERFORM PANEL-INIT.
A100.
      PERFORM BLANKA-ALL.
      PERFORM PANEL-ANROP-IBMOU001.
      MOVE SPACES TO ERRMSG.
      IF LASTCC = 8 GO TO A999.
      IF ACT = "01" PERFORM ENROLLMENT.
      IF ACT = "23" PERFORM ADDRESS-CHANGE.
      IF ACT = "60" PERFORM PAYMENTS-SUB.
      IF ACT = "99" PERFORM DELETES-SUB.
      IF ACT = "S" PERFORM SEARCH-TAB.
      IF ACT = "s" PERFORM SEARCH-TAB.
      GO TO A100.
A999.
      STOP RUN.
******************************
Animate-ibmoupd------------------------------------Level=01-Speed=5-Ins-Caps-Num-Scroll
F1=help F2=view F3=align F4=exchange F5=where F6=look-up  F9/F10=word-</> Escape
Animate Step Wch Go Zoom nx-If Prfm Rst Brk Env Qury Find Locate Txt Do Alt Ctrl
```

Figure 69. Showing Animator Window

The Animator enables you to do several things:

- Step through the code.
- Do a full run.
- View selected variables.
- Animate the code. This means that you will see all variables displayed as the Animator processes the code line by line.
- Modify variables.

You start the animation by pressing the A. When in animation mode, you can choose at what speed you want the Animator to process the code, by pressing any of the keys 0-9. An example of what the screen looks like when animating the code is shown in Figure 70 on page 113.



```
                                   anim18747
 DB000.
     PERFORM PANEL-ANBOP-IBMOU003.
     MOVE SPACES TO ERRMSG.
     IF LASTCC = 8
        MOVE ZERO TO LASTCC
        GO TO DB999.
     CALL "IBMCUST" USING PCUSTNO, NCUSTNO, ERRMSG.
     IF ERRMSG IS NOT = " " GO TO DB000.
     MOVE NCUSTNO TO WCUSTNO.
     INSPECT PZIPCODE TALLYING COUNTX FOR CHARACTERS
        REPLACING ALL SPACES BY ZEROS.
     MOVE NZIPCODE TO WZIPCODE.
     MOVE SPACES TO STR.
     MOVE PSTREET TO STREET IN ROAD.
     PERFORM ST ┌WCUSTNO──────────────────────────┐
     MOVE STREE │00000D70: 00 00 00 00 11 1C [^^^^^^]│
     MOVE SPACE └──────────────────────────────────┘
     IF PSTREET IS NOT = STREET IN PGM-NA┌NCUSTNO───┐
        WZIPCODE IS NOT = ZIPCODE IN PGM-│0000000111│
        IF WCUSTNO = CUSTNO IN PGM-CUST  └──────────┘
           GO TO DB200.
Monitor: WCUSTNO─────────────────────────────Level=02-Speed=0-Ins-Caps-Num-Scroll
F1=help F2=view F3=align F4=exchange                                      Escape
0-9=speed  Zoom
```

Figure 70. Using the Animation Mode

6.3.4 Test Tools

To be able to test the COBOL part of our application in stand-alone mode, separated from the GUI, we had to write some additional test programs. These test programs were written in both COBOL and C, depending on their purpose. For the purpose of testing the GUI call interface, the programs were of course written in C, but for testing the SQL calls we used COBOL programs. We also wrote a C program to test the MVS ISPF panels with the help of an IBM Internal tool. For the purpose of testing the different MVS releases on AIX without testing the ISPF panels, we wrote a new character-based screen handler, replacing the ISPF calls. This is to illustrate a mechanism that someone without the IBM internal tool might use to eliminate the ISPF dependency in the AIX environment while testing MVS targeted code.

The following paragraphs show and discuss some of the test programs.

6.3.4.1 Testing the C-to-COBOL Interface (testchg.c)

Our first program is the **testchg.c** program. This program tests the C-to-COBOL interface for the `ADDRESS-CHANGE SECTION`. The program accepts a customer number from the console, and then calls the `change` entry in the AIX version of the **ibmoupd** program.

After having completed the call to the COBOL program, the calling C program will display the completed call structure *chg* on the console and it will also display the return code as returned by DB2/6000. Refer to Figure 50 on page 94 for more details about the COBOL code.

```
/*********************************************************************/
/* testchg.c                                                         */
/* Program to test interface from C to COBOL                         */
/* Author : Leif Trulsson, IBM Sweden                                */
/*                                                                   */
/* Use testchg.mk to build executable                                */
/*********************************************************************/
#include <stdio.h>

#define ERROR    -1

struct chg {            /* Call structure for COBOL call   */
   char custno[10];
   char refno[10];
   char mailid[2];
   char sourcecode[3];
   char firstname[13];
   char lastname[18];
   char street[26];
   char city[20];
   char zipcode[5];
   char errmsg[70];
   int  sqlcode;
   };

char in[80];    /* Input string from console       */

extern int ibmoupd();             /* COBOL program - initialization  */
extern int change();              /* COBOL program - address change  */
extern int cobexit();             /* Close down COBOL system and exit*/

main(argc, argv)

int  argc;
char *argv[]
{
   int  status;
   struct chg chg;

#ifdef DEBUG
   printf("Init ibmoupd()\n");
#endif

  if (status = ibmoupd())         /* Call COBOL to initialize           */
      cobexit(status);

   while(status == 0)
   {
      getnum();                           /* Get a Customer Number          */
      sprintf(chg.custno,"%10s",in);
#ifdef DEBUG
      printf("custno  = %s\n",chg.custno);
      printf("Call change()\n");
#endif
                                 /* Call COBOL to get the zipcode */
      if (status = change(&chg))
      {
         status = 0;
       /*  cobexit(status); */
      }

      printf("chg        = %s\n", chg.custno);  /* Print the whole struc/
      printf("sqlcode    = %d\n", chg.sqlcode);
   };
   cobexit(status);
}
```

Figure 71 (Part 1 of 2). C-to-COBOL Interface Test Program

```
getnum()
{
   char *ret;                          /* Return value from console read  */

   printf("Enter Customer Number : ");
   ret = gets(&in[0]);

   if (strlen(in) == 0)                /* If empty string, terminate  */
      cobexit(0);
{
```

Figure 71 (Part 2 of 2). C-to-COBOL Interface Test Program

The following figure shows the *makefile* file for building our **testchg** executable file.

```
#####################################################################
# MAKEFILE for COBOL ibmoupd Program                                #
# Enter the Following:                                              #
#                                                                   #
#      make all     -- makes all programs                           #
#                                                                   #
#      make ibmoupd -- makes ibmoupd       program                  #
#      make testchg -- makes testchg       program                  #
#      make IBMCUST -- makes IBMCUST       program                  #
#                                                                   #
#      make cleanup -- removes builds from all sample programs      #
#####################################################################

EXECUTABLE=testchg
DATABASE  =aixdbm
DB2HOME   =/u/db2
LINK_FLAGS=  -a -x  -C SQLDB=$(DATABASE) -C SQLDB2 -C SQLPRE \
             -C IBMCOMP -L$(DB2HOME)/sqllib/lib -ldb2
COMPILER  =cob
CLIB      =$(HOME)/source/c
CC        = cc
CFLAGS    = -DDEBUG
COPY      = panvarnew.cbl change.cbl
MAIN      =testchg1.c
APPL_OBJS = $(CLIB)/IBMCUST.c
COBPGM    = ibmoupd.cob
SQLOBJ    = $(COBDIR)/sqlinix.o

OBJS      = $(MAIN:.c=.o) $(APPL_OBJS:.c=.o)

.c.o:
       $(CC) -g -c $(CFLAGS) $< -o $@

$(EXECUTABLE): $(OBJS) $(COBPGM) $(COPY)
       $(COMPILER) $(LINK_FLAGS) $(OBJS) $(COBPGM) $(SQLOBJ)
```

Figure 72. Testchg.mk Makefile File

6.3.4.2 Testing the ISPF panels

Figure 156 on page 236 shows a C program that was written to test the MVS ISPF panel definitions with the IBM Internal Tool **AIXISPF**. This tool enables the testing of the ISPF panels written for MVS on the AIX development platform. This tool made it possible to not only test the COBOL and DB2 code, but the MVS user interface itself, all on AIX. This tool consists of a library of C language calls that the developer uses in writing a panel test program. The source code for a sample test panel program is shown in Appendix B, “Sample Panel Test Program” on page 235.

6.3.4.3 The Character-Based Screen Handler

To be able to test the MVS releases on AIX without the limitation of the ISPF panel interface, we decided to write a character-based screen handler for AIX. This character-based screen handler uses the Micro Focus COBOL SPECIAL-NAMES and SCREEN-SECTIONs. To be consistent with our discussion in 6.3.2.4, "Use of Copy Files" on page 96, we created the following new COPY files:

panvarnew.cbl	Contains the new *Panel* variables definitions, as shown in Figure 157 on page 241. These definitions replace the ISPF panels as defined in the files ibmou001.pan through ibmou006.pan.
paninitnew.cbl	Initializes the new environment, as shown in Figure 73.
ibmpanel1.cbl	Contains the call to the Main panel, as shown in Figure 74.
ibmpanel2.cbl	Contains the call to the Delete panel.
ibmpanel3.cbl	Contains the call to the Address-Change panel.
ibmpanel4.cbl	Contains the call to the Payments panel.
ibmpanel5.cbl	Contains the call to the Delete panel.
ibmpanel6.cbl	Contains the call to the Duplicate Selections panel.

Refer to Appendix C, "Panel Definitions" on page 241 for the source code for the new panel definitions.

COPY File

```
*      Disable all other user function keys
           CALL x"AF" USING SET-BIT-PAIRS DISABLE-ALL-OTHER-KEYS.

*      Enable the F3 key
           CALL x"AF" USING SET-BIT-PAIRS ENABLE-F3.

*      Enable the F10 key
           CALL x"AF" USING SET-BIT-PAIRS ENABLE-F10.
```

Figure 73. New Environment Initialization

```
      ******************************************************
      *                                                    *
      ******************************************************
      *  THIS PROCEDURE CALLS THE MAIN MENU
       N000.
           MOVE 1 TO CURSOR-ROW.
           MOVE 1 TO CURSOR-COLUMN.
           MOVE ZERO TO LASTCC.
           MOVE ZERO TO EXIT-FLAG.
       N100.
      *  Loop until ENTER key or F3 are pressed.
           PERFORM UNTIL EXIT-FLAG = 1
              DISPLAY PANEL1
              ACCEPT PANEL1
              EVALUATE KEY-TYPE
      *  Accept terminated by the ENTER key
                 WHEN "0"
                    MOVE 1 TO EXIT-FLAG
                    IF ACT = "s" OR ACT = "S"
                       MOVE SPACES TO ERRMSG
                       MOVE SPACES TO PCUSTNO
                       CALL "IBMCUST" USING ARG NCUSTNO ERRMSG
```

Figure 74 (Part 1 of 2). New Character-Based Main Panel

```
                    IF ERRMSG = SPACES
                       MOVE NCUSTNO TO WCUSTNO
                       MOVE WCUSTNO TO CUSTNO IN PGM-CUST
                                       CUSTNO IN PGM-NAME
                    ELSE
                       MOVE SPACES TO STR
                       MOVE ARG TO LASTNAME IN L-NAME
                       PERFORM STR-INSPECT
                       MOVE LASTNAME IN L-NAME
                          TO LASTNAME IN PGM-NAME
                    END-IF
                    MOVE SPACES TO ERRMSG
                 END-IF
      *  Accept terminated by a Function key
              WHEN "1"
                 IF KEY-CODE-1 = 3
                    MOVE 1 TO EXIT-FLAG
                    MOVE 8 TO LASTCC
                 END-IF
           END-EVALUATE
        END-PERFORM.
```

Figure 74 (Part 2 of 2). New Character-Based Main Panel

The only other change we had to make to our original program was to include the lines shown in Figure 75, just before the beginning of the `DATA DIVISION`. This sets up the environment to use function keys and to use the console as input/output device.

```
      * Use SPECIAL-NAMES for Screen displays
       SPECIAL-NAMES.
           CONSOLE IS CRT
              CURSOR IS CURSOR-POSITION
              CRT STATUS IS KEY-STATUS.
```

Figure 75. Code Changes for the Use of Function Keys

6.4 Migration of the Database

There are several issues to consider when moving data from one database and environment, to another database and another environment. These include:

- What binary format is used
- National Language Support
- How to extract data from the old system
- In which format the data should be extracted
- Importing the data into the new system.

In our case, the data was moved from MVS where it was stored in EBCDIC format, to AIX where it was to be stored in ASCII format. We were also moving the data from one system (MVS) with the National Language Support of Swedish to another system where the language of choice was U.S. English. This means that the system has to be able to take care of the special Swedish letters. The system also has to handle other language specific matters, such as:

- Sort/collating sequences
- Uppercasing rules
- Date/time formats.

By carefully choosing language **Locales**, **Code Set**, and **Code Page** the same results can be achieved. Please refer to *IBM DATABASE 2 AIX/6000 Administration Guide* for more information about National Language Support.

In the following paragraphs we talk about:

- Creating the database
- Extracting and moving the old data
- Creating tables and importing the data.

6.4.1 Creating the Database

Creating the new database on AIX posed the least problem in our project. In fact, we used the same table definitions as were used when creating the tables on MVS DB2, as shown in Figure 77 on page 120. We created the database by issuing the following command as `sysadm`:

```
db2 create database aixdbm
```

After having created the database, we connected to the database and granted other users ability to connect to the database by issuing the following commands:

```
db2 connect to aixdbm
db2 grant connect on database to public
```

6.4.2 Extracting and Moving the Old Data

After the database had been created, it was time to create the tables, and load them with data. But before we could do this, we had to move the data from DB2 on MVS to AIX as ASCII files. In our case the MVS DB2 database was located in Malmö, Sweden, and our AIX system was located in San Jose, California, U.S.A. Because of this, we were not able to set up our DB2/6000 to utilize DDCS/6000 to connect to the MVS DB2 system as described in Figure 1 on page 20 in Chapter 4, "Defining the Project and Setting Its Goals" on page 29. Normally, this is the easiest and most suitable way to transfer data between systems. With the DDCS/6000 product, it is possible to export data from DB2 on MVS to a PC/IXF (Personal Computer/Integrated Exchange Format) file on the workstation, and import this file into and existing DB2/6000 table, and vice versa.

As the data format produced by the MVS DB2 DUMP utility was not supported by DB2/6000, we had to write our own data extraction (export) program, as shown in Appendix D, "Sample Data Extraction Program" on page 253. We decided to extract the data into character format, so we could use the DB2/6000 ASCII import facility. After extracting the data into ordinary MVS data sets, we had to transfer the data from our MVS system in Sweden to our AIX system the in USA. Figure 76 on page 119 shows the path our data took to reach its destination.

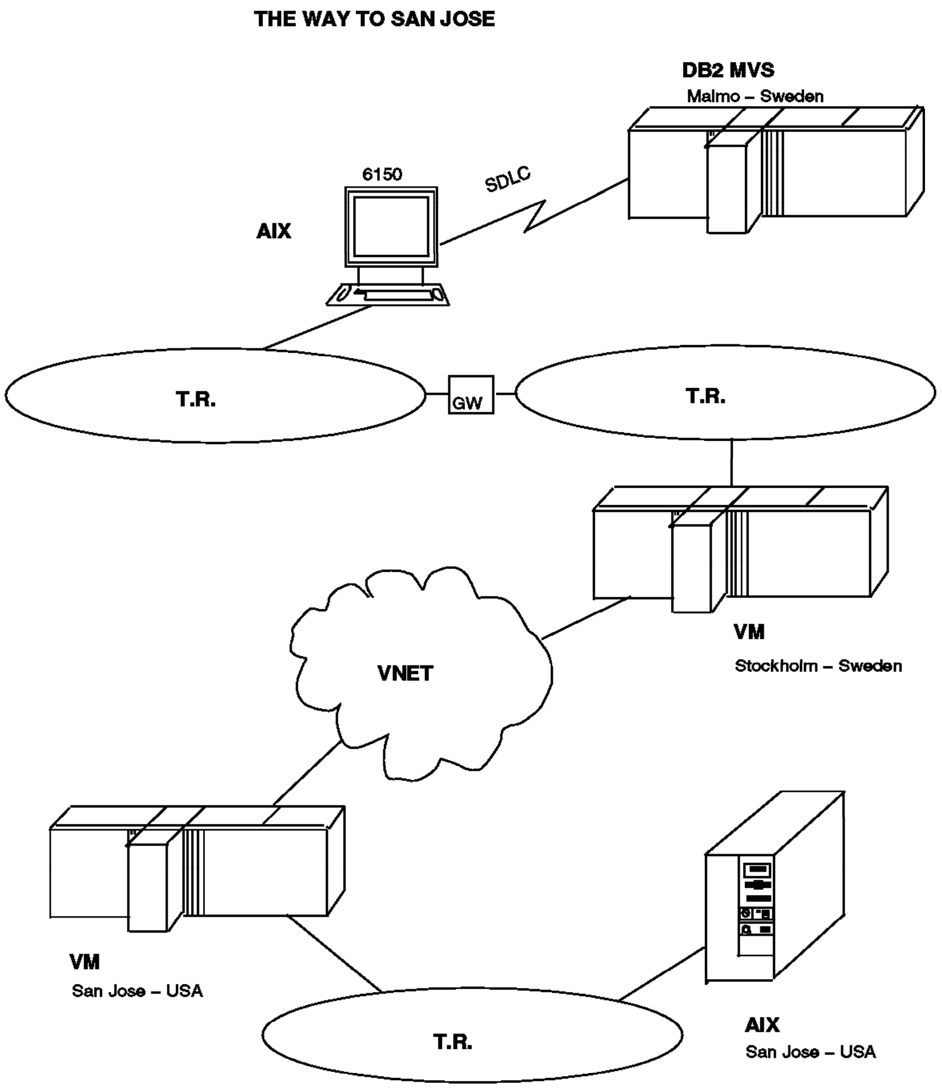


Figure 76. Route Taken by DB2 Data from its MVS Host to AIX

We transferred our four database files to an IBM 6150 RT PC computer using a tool which is called **APPC File Transfer (AFT)**. This tool was developed and is sold by IBM Sweden. From the RT PC, we transferred the files, with the help of the TCP/IP FTP (File Transfer Protocol) utility, to a VM system located in Stockholm, Sweden. From this VM system in Sweden we sent the files over the IBM internal VM network to a VM system in San Jose. And from this VM system we once again used FTP to transfer the data from this VM system down to our target AIX system.

6.4.3 Creating Tables and Importing the Data

With the data finally on our AIX system, we were ready to create our tables and import the data into our previously created database. The data is imported with the help of the same file we used to create our tables, as shown in Figure 77.

```
-- Issue 'db2 -svtf db2cust'. (Note: the quote ' is not part of what
-- you should issue.) The flags "-svtf" have the following meanings:
--  -s -- Stops on SQL error.
--  -v -- Echo the command.
--  -t -- Each SQL statement is terminated by semicolon.
--  -f -- The parameter following this flag is the file containing
--        all the SQL commands.

-- Establish a database connection.
connect to aixdbm;
-- Create the tables and indexes.
CREATE TABLE CUST
  (CUSTNO NUMERIC(10) NOT NULL,
   REFNO NUMERIC(10) NOT NULL,
   ACTDATE  DECIMAL(6) NOT NULL WITH DEFAULT,
   ADDRCHG  DECIMAL(6) NOT NULL WITH DEFAULT,
   PROFIT NUMERIC(7,2),
   MAILID NUMERIC(2) NOT NULL,
   SOURCECODE NUMERIC(3) NOT NULL,
   COLLECTCODE  SMALLINT,
   DUNNCODE  SMALLINT);

CREATE INDEX CUST_IX1 ON CUST (ACTDATE ASC);
CREATE UNIQUE INDEX CUST_IX2 ON CUST (CUSTNO ASC);
```

Figure 77 (Part 1 of 2). File to Create Tables and Import data

```
CREATE TABLE NAME
  (CUSTNO NUMERIC(10) NOT NULL,
   FIRSTNAME    CHARACTER(13) NOT NULL,
   LASTNAME     CHARACTER(18) NOT NULL,
   STREET       CHARACTER(26) NOT NULL,
   ZIPCODE      INTEGER NOT NULL);

CREATE INDEX NAME_IX ON NAME (CUSTNO ASC);

CREATE TABLE PAYMENT
  (REFNO       NUMERIC(10) NOT NULL,
   PAYDATE     DECIMAL(6)  NOT NULL,
   AMOUNT      NUMERIC(7,2) NOT NULL);

CREATE INDEX PAYMENT_IX ON PAYMENT (REFNO ASC);

CREATE TABLE ZIP
  (ZIPCODE     INTEGER NOT NULL,
   CITY        CHARACTER(20) NOT NULL);

CREATE UNIQUE INDEX ZIP_IX ON ZIP (ZIPCODE ASC);

-- Do commit to save work.
commit work;

-- Import the data using ASC method into the tables.

import from cust.tab of asc
  method l (1 10, 12 21,23 28,30 35,37 45,47 48,50 52,54 54,56 56) insert into cust;

import from name.tab of asc
  method l (1 10, 12 24,26 43,45 70,72 76) insert into name;

import from payment.tab of asc
  method l (1 10, 12 17,19 26) insert into payment;

import from zip.tab of asc
  method l (1 5, 6 25) insert into zip;

-- Do commit to save work.
commit work;
```

Figure 77 (Part 2 of 2). File to Create Tables and Import data

Having created the tables, we had to grant other users access rights to our tables. The commands to do this are:

```
db2 grant all on table cust to public
db2 grant all on table name to public
db2 grant all on table payment to public
db2 grant all on table zip to public
```

Finally, we were ready to test the application on our data.

6.5 Integration and Test

This section describes the various steps to integrate the calls to the modified COBOL code with the generated GUI code from AIC 1.1 and the callbacks. It describes the Application Program Interface (API) from C to COBOL and shows the use of the various tools that are integrated with SDE WorkBench/6000 to implement and test this interface.

6.5.1 Application Program Interface Conventions

We now needed to enhance the callbacks with calls to corresponding COBOL functions. These COBOL functions would then actually call the DB2 database to read, add, change or delete from or to the database. The COBOL environment had to be prepared before any other COBOL function was called, and a corresponding termination call had to be made when returning from the application. With each call to a specific COBOL function we decided to pass along a data structure with all interface parameters. We decided to have a separate data structure for each of the main selections that were available on the main menu shown in Figure 36 on page 80. So we had to define five interface structures, and we decided to call them

- `Enroll`
- `Change`
- `Payment`
- `Delete`
- `Search`.

For both COBOL and C we would have a separate file to contain these structure definitions. These files had the same name, but the file name extension was `.cbl` for the COBOL file, and `.h` for the C file. For example, for the *change()* function we had a COBOL source file `change.cbl` and a C source file `change.h`. These two files had to reflect the same offsets of the data, so we had to make sure that both languages read and write to the same offset in the data structure when accessing a particular field. Refer to Figure 78 on page 123 for the data structure definitions in the `change.cbl` file and Figure 79 for the `change.h` file.

```
*      Structure to be used when called from Main C-pgm to the
*      ADDRESS-CHANGE routine.
*      This is a part of the LINKAGE-SECTION

       LINKAGE SECTION.
       01 CHANGE.
          03  PCUSTNO         PIC X(10) VALUE SPACE.
          03  NCUSTNO REDEFINES PCUSTNO   PIC 9(10).
          03  PREFNO          PIC X(10) VALUE SPACE.
          03  NREFNO REDEFINES PREFNO   PIC 9(10).
          03  PMAILID         PIC X(2) VALUE SPACE.
          03  NMAILID REDEFINES PMAILID   PIC 9(2).
          03  PSRC            PIC X(3) VALUE SPACE.
          03  NSRC REDEFINES PSRC   PIC 9(3).
          03  PFNAME          PIC X(13) VALUE SPACE.
          03  PLNAME          PIC X(18) VALUE SPACE.
          03  PSTREET         PIC X(26) VALUE SPACE.
          03  PCITY           PIC X(20) VALUE SPACE.
          03  PZIPCODE        PIC X(5) VALUE SPACE.
          03  NZIPCODE REDEFINES PZIPCODE   PIC 9(5).
          03  ERRMSG           PIC X(70) VALUE SPACE.
          03  SQLC           PIC X(2) COMP-5.
```

Figure 78. The change.cbl File for the COBOL API Structure

```
/* Structure to be used when calling the change() subroutine to */
/* interface to the COBOL function.                             */

typedef struct change {
      char     custno[10];
      char     refno[10];
      char     mailid[2];
      char     sourcecode[3];
      char     firstname[13];
      char     lastname[18];
      char     street[26];
      char     city[20];
      char     ZIPCode[5];
      char     SQLErrorMessage[70];
      int      SQLCODE;
   } ChangeStruct;
```

Figure 79. The change.h File for the C API Structure

6.5.2 Implementing the Calls to COBOL in the Callbacks

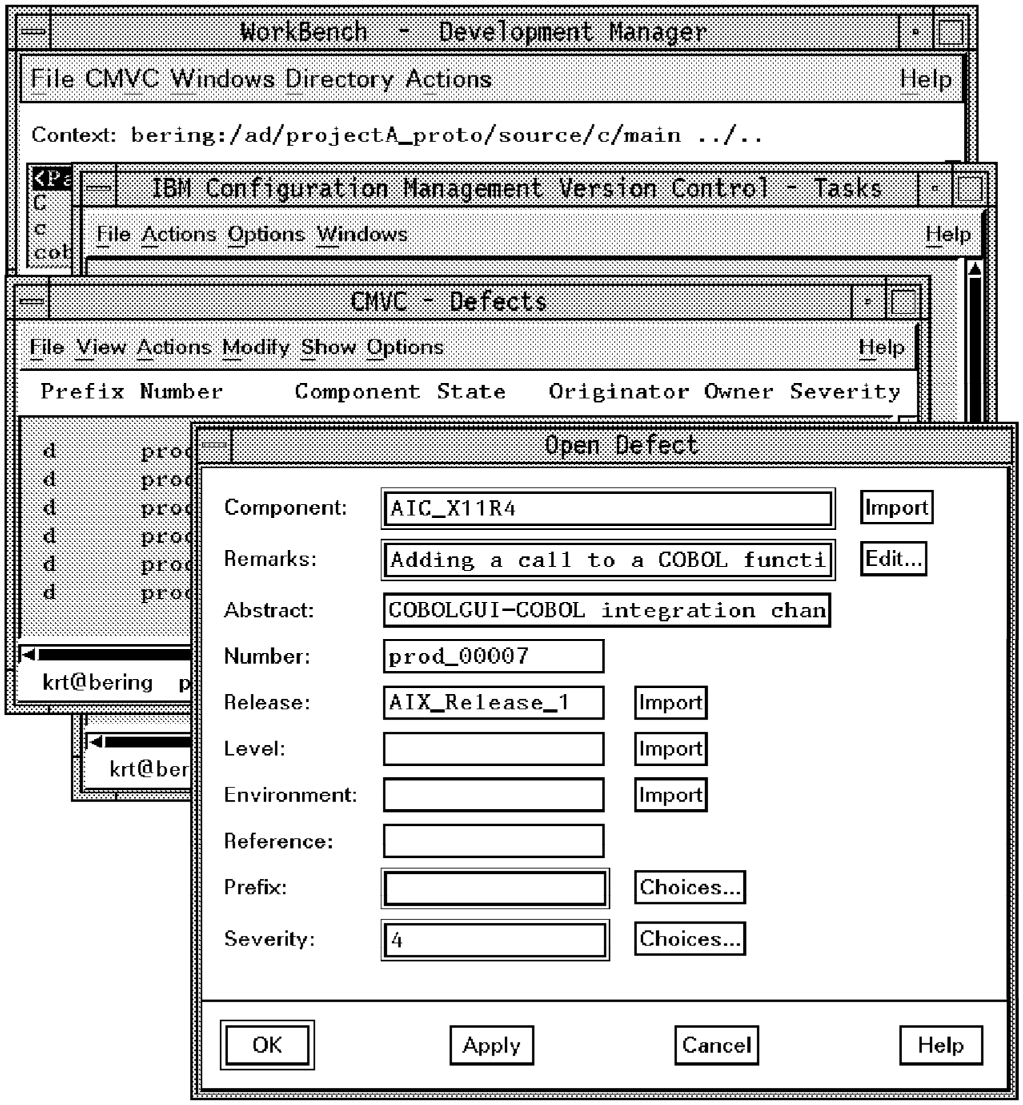
Once the interface structure files as shown in Figure 78 and Figure 79 were implemented and saved in the corresponding directories, we started to modify the callbacks to implement the calls to the COBOL code.

The extracts of the code shown in this chapter process the action code 23, shown in Figure 36 on page 80, for a given customer number that is entered in the Argument text field. Doing so, a user could request an address change for a customer identified by a given customer number. Once the user selects **OK**, the

OnlineUpdateOkPushButtonActivateCallback gets started. The callback should take the customer number and insert it into the corresponding `custno` field in the `ChangeStruct` structure, as shown in Figure 79, initialize the COBOL environment, and call the *change* function, implemented in COBOL, passing the interface structure. The *change* COBOL routine would fill all the other fields in the structure after having queried the database. The callback would then create a new window to show the customer details and fill the text fields of that window with the corresponding data values.

As a result of the activities as described in 6.2, “Design and Implementation of Our User Interface” on page 68 the new source files were placed under control of CMVC also. The first step in adding the COBOL logic to the callbacks was to check out the `OnlineUpdateCallbacks.c` from the CMVC library. To do this, we first created a defect that should be used to gather all code changes for enabling the calls to COBOL.

We selected **Defects** from the Windows pull-down of the WorkBench Development Manager window. CMVC started and the CMVC Tasks window and the CMVC Defects window appeared. We selected **Open** from the Actions pull-down of the CMVC Defects window and entered the data in the Open Defect window as shown in Figure 80 on page 125.


WorkBench - Development Manager
File CMVC Windows Directory Actions Help
IBM Configuration Management Version Control - Tasks
File Actions Options Windows Help
CMVC - Defects
File View Actions Modify Show Options Help
Open Defect
Import
Edit...
Choices...
OK
Apply
Cancel
Help

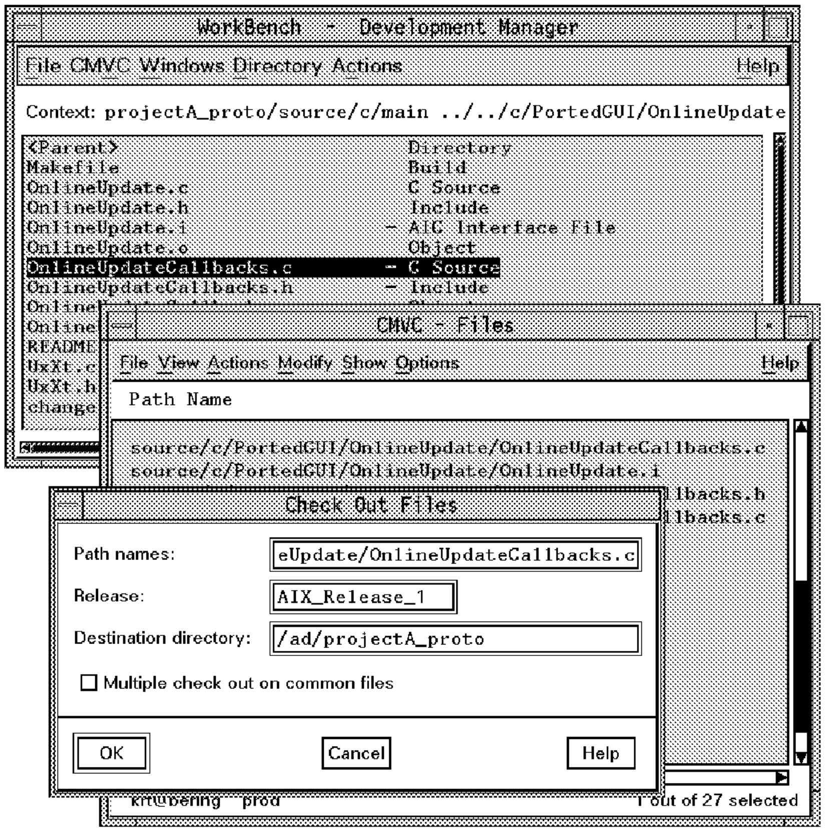
Context: bering:/ad/projectA_proto/source/c/main ../..

Prefix Number Component State Originator Owner Severity

Component:	AIC_X11R4
Remarks:	Adding a call to a COBOL functi
Abstract:	COBOLGUI-COBOL integration chan
Number:	prod_00007
Release:	AIX_Release_1
Level:	
Environment:	
Reference:	
Prefix:	
Severity:	4

Figure 80. Creating a Defect to Add the Calls to COBOL

We pressed the **OK** button to create the defect in CMVC. Once we had the defect created, we could check out the `OnlineUpdateCallbacks.c` file. We changed the directory of Development Manager to show the `/ad/projectA_proto/source/c/PortedGUI/OnlineUpdate` directory. The entry for the `OnlineUpdateCallbacks.c` file showed a dash character following the file name to indicate that the file was read-only and currently checked in. We selected the file in the list and then selected **Check Out...** from the CMVC pull-down of WorkBench Development Manager window and saw the windows as shown in Figure 81 on page 126.


WorkBench - Development Manager
File CMVC Windows Directory Actions
Help
CMVC - Files
File View Actions Modify Show Options
Help
Check Out Files
OK
Cancel
Help

```
Context: projectA_proto/source/c/main ../../c/PortedGUI/OnlineUpdate

<Parent>                          Directory
Makefile                          Build
OnlineUpdate.c                    C Source
OnlineUpdate.h                    Include
OnlineUpdate.i                  - AIC Interface File
OnlineUpdate.o                    Object
OnlineUpdateCallbacks.c         - C Source
OnlineUpdateCallbacks.h         - Include
Online
Online
README
UxXt.c
UxXt.h
change
```

```
Path Name

source/c/PortedGUI/OnlineUpdate/OnlineUpdateCallbacks.c
source/c/PortedGUI/OnlineUpdate/OnlineUpdate.i
llbacks.h
llbacks.c

krt@bering   prod                              1 out of 27 selected
```

Path names: eUpdate/OnlineUpdateCallbacks.c

Release: AIX_Release_1

Destination directory: /ad/projectA_proto

☐ Multiple check out on common files

Figure 81. Checking Out the OnlineUpdateCallbacks.c Source File from CMVC

We pressed **OK** to check the file out so we could edit it. We then invoked Program Editor to edit the `OnlineUpdateCallbacks.c` file by selecting **Edit** from the Actions pull-down of Development Manager. Figure 82 on page 127 shows the initialization part of `OnlineUpdateCallbacks.c`. We added the `#include` directive to include the `change.h` header file (**1**), and added `extern` definitions for the functions `ibmoupd`, `change` and `cobexit` (**2**). We then added the call to intialize the COBOL environment depending on the value of a static variable as shown in **3** at the beginning of the activate callback for the **OK** push button. If the call to *ibmoupd()* failed for any reason, the connection from C to COBOL could not be established. In that case, a proper error handling routine would be invoked.

```
#include "change.h"                                  1

static init = 0;         /* Did we initialize the COBOL environment?   */

extern int ibmoupd();
extern int cobexit();                                2
extern int change(ChangeStruct *Parms);

void OnlineUpdateOkPushButtonActivateCallback(Widget PushButton)
{
  Widget NextWidget;
  char *Action, *Argument; /* The user input of the two text fields    */


  /* establish addressability and set the context pointer             */
  UxOnlineUpdateContext = (_UxCOnlineUpdate *)UxGetContext(PushButton);

  if (!init) {
    /* It's the first time that we activate the button. We need to     */
    /* initialize the COBOL environment.                               */
    init = 1;
    if (  (rc = ibmoupd()) !=0)                     3
       OnlineUpdateProcessInitializationErrors();
  }
```

Figure 82. The COBOL Environment Initialization Code in OnlineUpdateCallbacks.c

Figure 83 shows the termination code of `OnlineUpdateCallbacks.c`. The corresponding code to shut down the COBOL environment in an orderly fashion was done in the activate callback of the **End** push button as shown at the place indicated by 4. If the user pressed this button, the application terminated and the COBOL environment was cleaned up.

```
void OnlineUpdateEndPushButtonActivateCallback(Widget PushButton)
{
  /* End of the application                                        */
  /* rc is a static global variable that is set to a corresponding */
  /* numeric value. It is passed to cobexit, and is returned       */
  /* to the operating system shell.                                */
  cobexit(rc);                                       4
}
```

Figure 83. The COBOL Environment Termination Code in OnlineUpdateCallbacks.c

The remaining activity was in fact the most important one: we had to implement the call to the *change()* function, which was implemented in COBOL as explained in 6.3, “Design and Implementation of the COBOL Code” on page 89.

We invoked Program Editor from WorkBench Development Manager window by selecting **Edit** from the Actions pull-down. We then changed the `OnlineUpdateCallbacks.c` file to contain the corresponding code required to invoke the COBOL function as shown in Figure 84 on page 128 and indicated with 5.

```
  /* Read in the contents of the text fields                          */
  Action = XmTextFieldGetString(OnlineUpdateActionTextField);
  Argument = XmTextFieldGetString(OnlineUpdateArgumentTextField);

  if (strcmp(Action, "23")==0) {
    /* 'Address change' was selected                                  */
    Widget NextWidget;    /* Widget for the CustomerDetails window   */
    ChangeStruct Parms;   /* Interface structure C to COBOL          */
    char field[27];       /* String to be written to text fields     */

    /* Initialize the structure to be passed to the COBOL function   */
    memset(&Parms, ' ', sizeof(Parms));
    Parms.SQLCODE=0;

    /* Create a Widget for the CustomerDetails window by using the   */
    /* function as generated by AIC 1.1                               */
    NextWidget = create_AddressChange();
    /* Initialize the static context variable that is declared in    */
    /* the header file generated by AIC 1.1                           */
    UxAddressChangeContext = (_UxCAddressChange *)UxGetContext(NextWidget);

    /* Fill the interface structure with the customer number as it   */
    /* was read from the window.                                      */
    memcpy(Parms.custno, Argument, strlen(Argument));

    /* Call the COBOL program to read from the DB and to return the  */
    /* result data. Use the interface structure to pass data to      */
    /* the COBOL function as well as to return results.              */

    if (rc=change(&Parms)!=0)                4
    {  /* Error occured during C-to-COBOL call */
      OnlineUpdateProcessCOBOLErrors(OnlineUpdateEndPushButton);
    }

    if (Parms.SQLCODE!=0) {
      /* SQL error occurred */
      OnlineUpdateProcessDBErrors(Parms);
    } else {
      /* The DB2 query succeeded and the COBOL function filled in    */
      /* the reply data. Read the structure and fill the text fields */
      /* one by one.                                                  */
      /* The customer number is already available from the main      */
      /* window.                                                      */
      XmTextFieldSetString(AddressChangeCustomerNo, Argument);
      /* All the others are taken from the interface structure       */
      memcpy(&field, Parms.refno, 10);
      field[10]='\0';
      XmTextFieldSetString(AddressChangeReferenceNo, field);

      memcpy(&field, Parms.mailid, 2);
      field[2]='\0';
      XmTextFieldSetString(AddressChangeMailId, field);

      /* The other fields are omitted here in this example.
*/

      /* ... and now show the customer details window filled with data
*/
      UxPopupInterface(NextWidget, no_grab);

    }
  }
```

Figure 84. Calling the COBOL Function from the C Callback

Once the changes were edited, we selected **Compile** from the Actions pull-down of the WorkBench Development Manager window to compile this new version of the `OnlineUpdateCallbacks.c` file. This is similar to what is shown in Figure 41 on page 85.

6.5.3 Link and Test

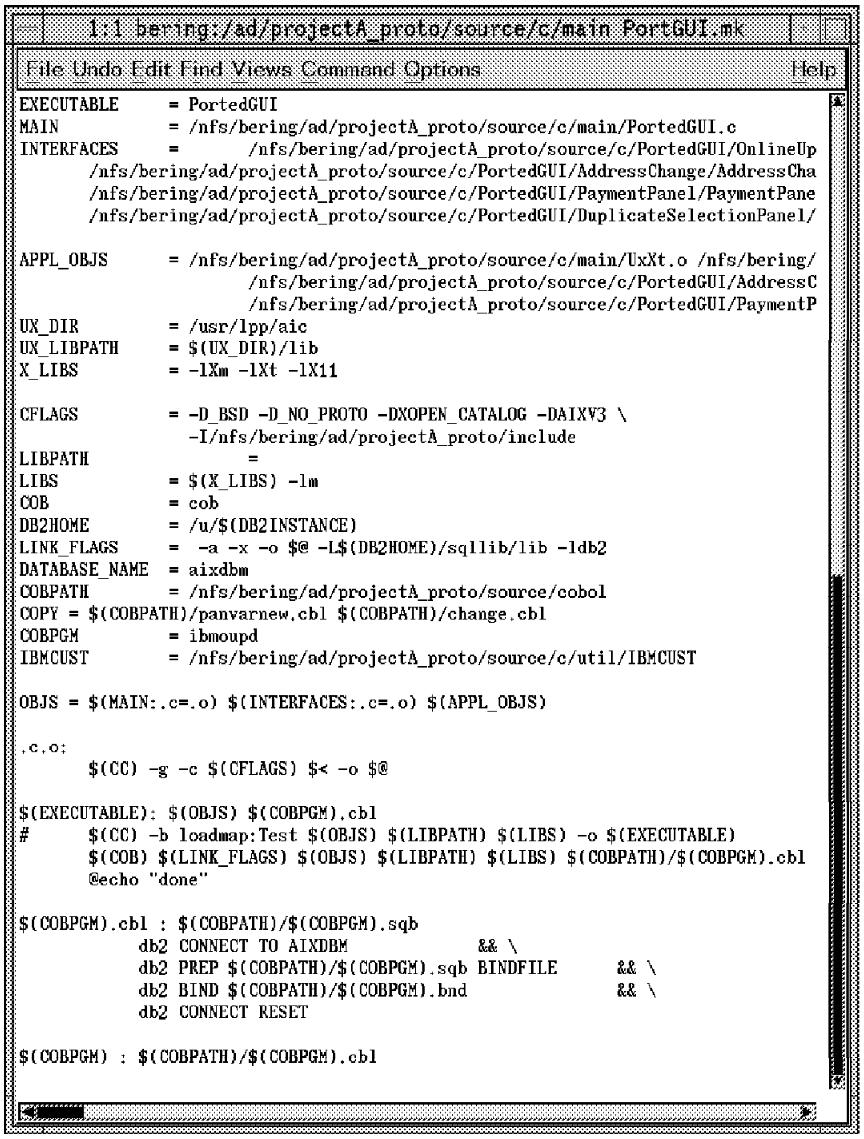
After both the changes to the callbacks implemented in the `OnlineUpdateCallbacks.c` file, and the database logic implemented in the `ibmoupd.cbl` file, were successfully compiled, we had to link these together. To do this, we decided to have a *makefile* file that can be called for compilation of the COBOL programs, compilation of the C programs, and linking the various object modules and libraries together. The *makefile* file we started from was the one generated from AIC because this was satisfactory. A *makefile* file used after an independent extract from CMVC would have used a *makefile* file that did not require the generation of the interface source and include files like the one we used.

We invoked Program Editor from WorkBench Development Manager window and edited the `PortedGUI.mk` *makefile* file.

The *makefile* file consisted of rules to:

- Compile the C source files generated by AIC
- Compile the C source files for some edit checks being called by the COBOL programs
- Compile the COBOL source files with DB2 support
- Link the object modules of the compiled files together with object modules of the compiled callbacks.

Figure 85 on page 130 shows how Program Editor was used to edit the `PortedGUI.mk` *makefile* file.

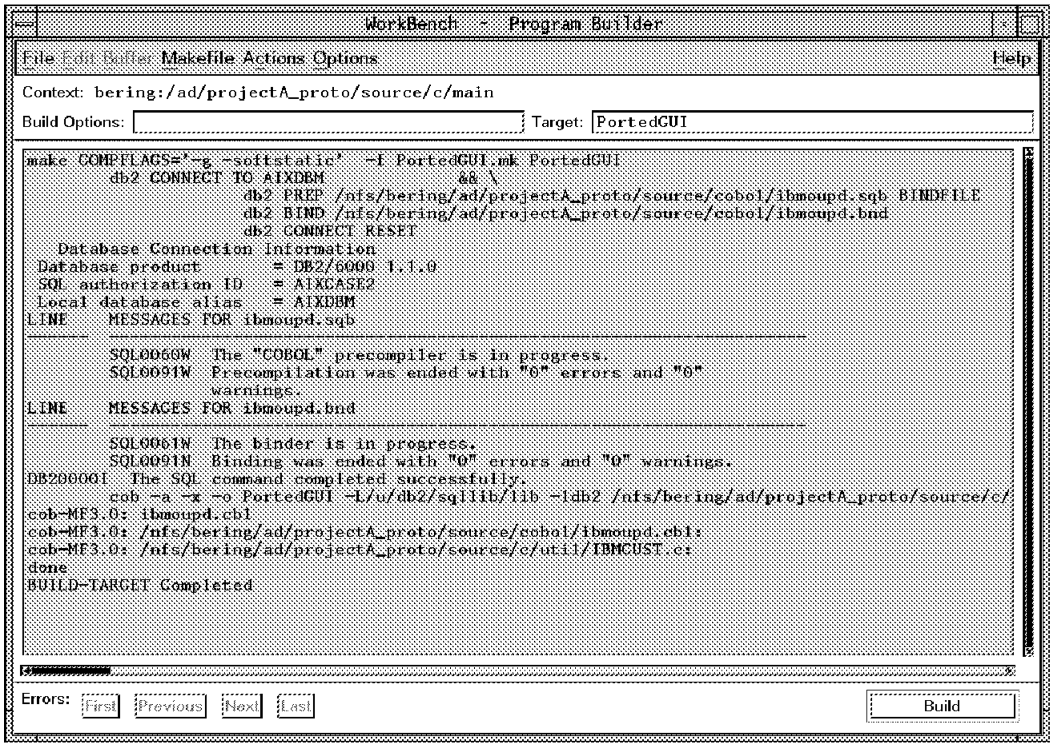

1:1 bering:/ad/projectA_proto/source/c/main PortGUI.mk
File Undo Edit Find Views Command Options
Help

```
EXECUTABLE     = PortedGUI
MAIN           = /nfs/bering/ad/projectA_proto/source/c/main/PortedGUI.c
INTERFACES     =        /nfs/bering/ad/projectA_proto/source/c/PortedGUI/OnlineUp
       /nfs/bering/ad/projectA_proto/source/c/PortedGUI/AddressChange/AddressCha
       /nfs/bering/ad/projectA_proto/source/c/PortedGUI/PaymentPanel/PaymentPane
       /nfs/bering/ad/projectA_proto/source/c/PortedGUI/DuplicateSelectionPanel/

APPL_OBJS      = /nfs/bering/ad/projectA_proto/source/c/main/UxXt.o /nfs/bering/
                        /nfs/bering/ad/projectA_proto/source/c/PortedGUI/AddressC
                        /nfs/bering/ad/projectA_proto/source/c/PortedGUI/PaymentP
UX_DIR         = /usr/lpp/aic
UX_LIBPATH     = $(UX_DIR)/lib
X_LIBS         = -lXm -lXt -lX11

CFLAGS         = -D_BSD -D_NO_PROTO -DXOPEN_CATALOG -DAIXV3 \
                 -I/nfs/bering/ad/projectA_proto/include
LIBPATH                 =
LIBS           = $(X_LIBS) -lm
COB            = cob
DB2HOME        = /u/$(DB2INSTANCE)
LINK_FLAGS     =  -a -x -o $@ -L$(DB2HOME)/sqllib/lib -ldb2
DATABASE_NAME  = aixdbm
COBPATH        = /nfs/bering/ad/projectA_proto/source/cobol
COPY = $(COBPATH)/panvarnew.cbl $(COBPATH)/change.cbl
COBPGM         = ibmoupd
IBMCUST        = /nfs/bering/ad/projectA_proto/source/c/util/IBMCUST

OBJS = $(MAIN:.c=.o) $(INTERFACES:.c=.o) $(APPL_OBJS)

.c.o:
       $(CC) -g -c $(CFLAGS) $< -o $@

$(EXECUTABLE): $(OBJS) $(COBPGM).cbl
#      $(CC) -b loadmap:Test $(OBJS) $(LIBPATH) $(LIBS) -o $(EXECUTABLE)
       $(COB) $(LINK_FLAGS) $(OBJS) $(LIBPATH) $(LIBS) $(COBPATH)/$(COBPGM).cbl
       @echo "done"

$(COBPGM).cbl : $(COBPATH)/$(COBPGM).sqb
            db2 CONNECT TO AIXDBM               && \
            db2 PREP $(COBPATH)/$(COBPGM).sqb BINDFILE      && \
            db2 BIND $(COBPATH)/$(COBPGM).bnd               && \
            db2 CONNECT RESET

$(COBPGM) : $(COBPATH)/$(COBPGM).cbl
```

Figure 85. Editing the Makfile File to Create the PortedGUI Program

We saved the *makefile* file and started Program Builder from the Tool Manager Start dialog box. We selected **Set Name...** from the Makefile pull-down in Program Builder, we entered the name `PortedGUI.mk` in the text field that showed up, and we pressed the **OK** button. We entered the name `PortedGUI` in the Target field and pressed the **Build** push button.

Program Builder run the *makefile* file and built an executable file program called `PortedGUI` as shown in Figure 86 on page 131.


WorkBench - Program Builder
File Edit Buffer Makefile Actions Options Help

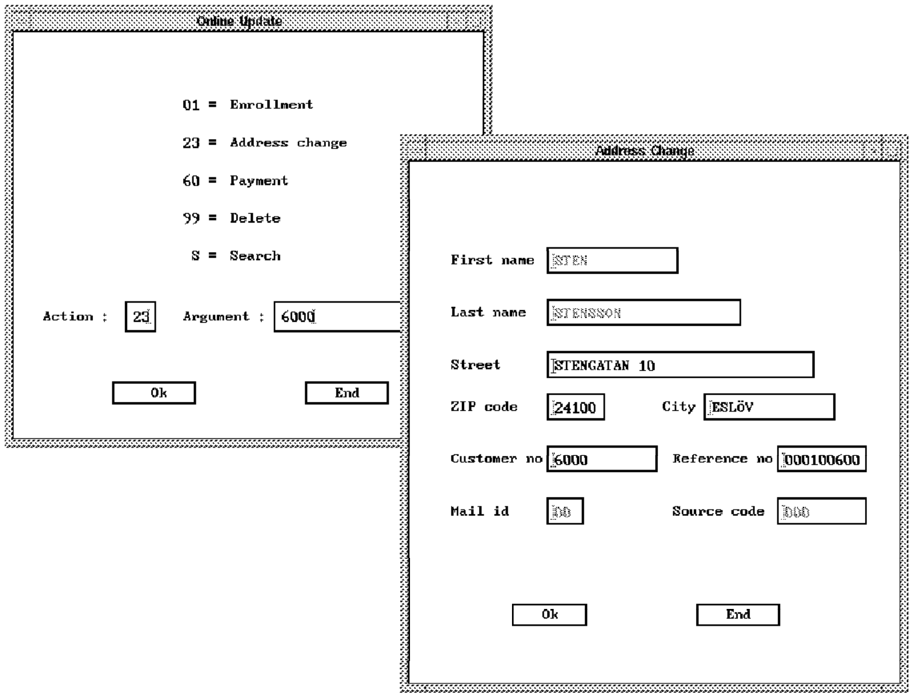
Context: bering:/ad/projectA_proto/source/c/main

Build Options: Target: PortedGUI

```
make COMPFLAGS='-g -softstatic'  -f PortedGUI.mk PortedGUI
        db2 CONNECT TO AIXDBM               && \
                    db2 PREP /nfs/bering/ad/projectA_proto/source/cobol/ibmoupd.sqb BINDFILE
                    db2 BIND /nfs/bering/ad/projectA_proto/source/cobol/ibmoupd.bnd
                    db2 CONNECT RESET
   Database Connection Information
 Database product       = DB2/6000 1.1.0
 SQL authorization ID   = AIXCASE2
 Local database alias   = AIXDBM
LINE    MESSAGES FOR ibmoupd.sqb
------  --------------------------------------------------------------------
        SQL0060W  The "COBOL" precompiler is in progress.
        SQL0091W  Precompilation was ended with "0" errors and "0"
                  warnings.
LINE    MESSAGES FOR ibmoupd.bnd
------  --------------------------------------------------------------------
        SQL0061W  The binder is in progress.
        SQL0091N  Binding was ended with "0" errors and "0" warnings.
DB20000I  The SQL command completed successfully.
        cob -a -x -o PortedGUI -L/u/db2/sqllib/lib -ldb2 /nfs/bering/ad/projectA_proto/source/c/
cob-MF3.0: ibmoupd.cbl
cob-MF3.0: /nfs/bering/ad/projectA_proto/source/cobol/ibmoupd.cbl:
cob-MF3.0: /nfs/bering/ad/projectA_proto/source/c/util/IBMCUST.c:
done
BUILD-TARGET Completed
```

Figure 86. Using Program Builder to Create an Executable File

We could invoke this program by double-clicking the entry for it in the WorkBench Development Manager window, or by entering the program name in any window that offers a command line interface. The system would show the main menu of the application, where we could enter `23` for the Action text field and `1111` for the Argument text field. When we selected **OK**, the system would look up the data for the specified customer number in the DB2 database, and display them as shown in Figure 87 on page 132.


Online Update

01 = Enrollment

23 = Address change

60 = Payment

99 = Delete

S = Search

Action : 23 Argument : 6000

Address Change

First name	STEN		
Last name	STENSSON		
Street	STENGATAN 10		
ZIP code	24100	City	ESLÖV
Customer no	6000	Reference no	000100600
Mail id	DD	Source code	DDD

Figure 87. Testing the Complete Application

The explanations and examples shown so far dealt with the logic required for displaying customer data during a request for changing a customer address. Similar enhancements had to be added to the various other callbacks and COBOL source files to implement the corresponding logic for displaying, changing, and deleting customers and for displaying and adding customer payments.

Once the code changes were done, we had to:

1. Create a track for the defect, so the defect changes into fix state
2. Check-in the modified source files into the library for the created track
3. Modify the track to put it into integrate state.

Once this was done, *AIXtester*, the person doing the integration, would be notified that certain tracks are to be integrated. Once all the tracks associated with a particular level were in this state, the build would be extracted and the level status would change from *integrate* to *commit* after a successful compile and link cycle. Because of our customized process, we would change the defect to verify state to inform the testers that the defect was fixed and is ready for an independent test. A successful execution of the test would then enable closing the defect.

Chapter 7. Improving the Application's GUI

This chapter describes our approach to improving the GUI of the ported legacy application. The goal of the first migrating activity, described in Chapter 6, "Migrating the Legacy Application to AIX" on page 61, was to port the legacy application to AIX with minimum programming effort, and with an identical user interface. This approach might be used where one wished to avoid end user retraining. The result of this effort, however, was a user interface that did not take advantage of the capabilities of the AIXwindows system and that was quite old-fashioned in its look-and-feel.

Our next step was to develop a new user interface for the legacy application, that was based on modern human factors principles and object-oriented program design considerations. This chapter looks at our design decisions, how we implemented them and how we used the tools that were integrated with SDE WorkBench/6000 while we improved our application's GUI.

7.1 Using SDE WorkBench/6000 and Integrated Tools to Improve the GUI

Our GUI developer made use of the same tools as used earlier when creating the first graphical user interface, except that we now chose to make use of the newer version of AIXwindows Interface Composer/6000. We moved to AIC 1.2 at this time because it offered us a chance to look at the impact that an increased degree of integration with SDE WorkBench/6000 would have on the developer. It also positioned us well to move on to the next phase of the project which would include generating C^{++} code from AIC and reimplementing the COBOL in C^{++}. An additional benefit was that it gave us a chance to look at issues related to having multiple tools in a single SDE WorkBench/6000 tool class operating on different hosts in our development environment.

7.1.1 Multiple Tools in a Single Tool Class on Separate Hosts

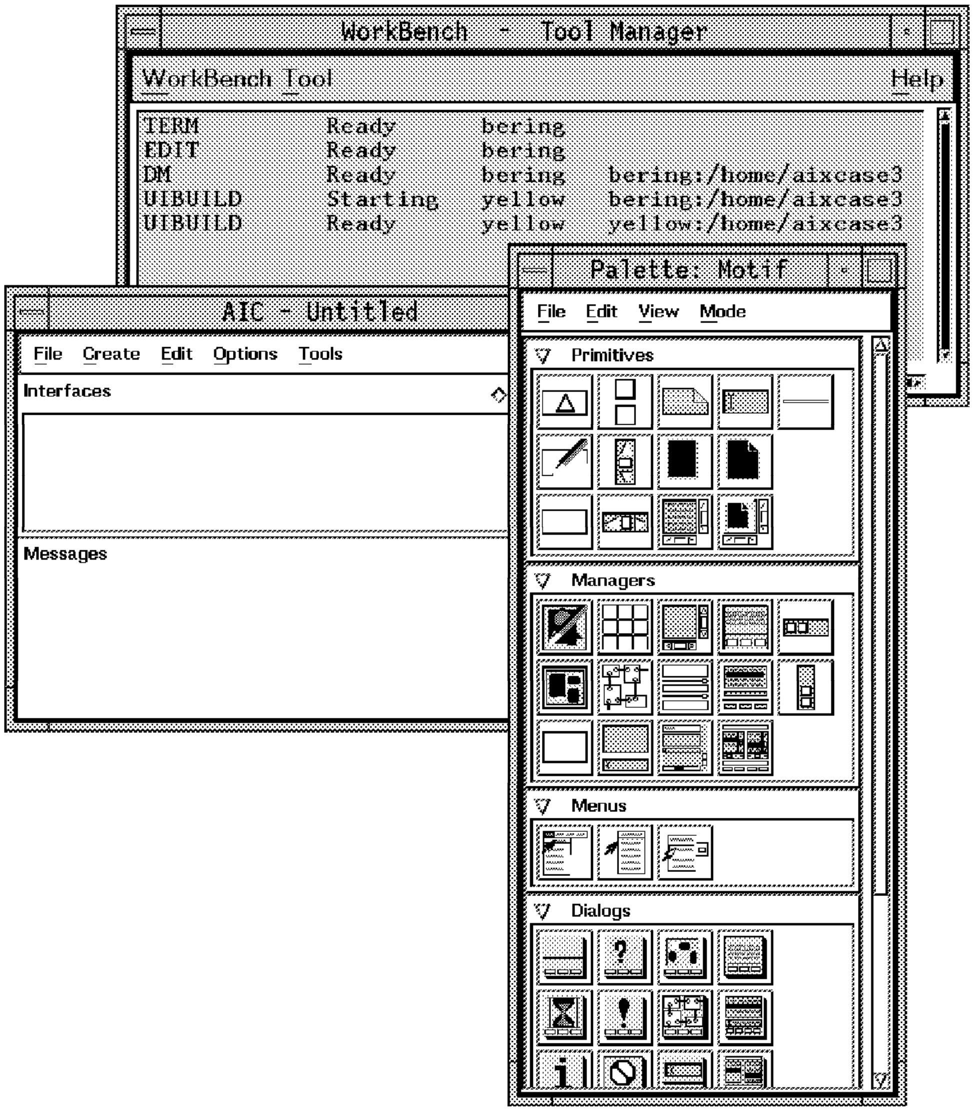
As we mentioned earlier, AIC 1.2 was installed only on *yellow*, while the older version was installed on *bering*. This was necessary because the newer version of AIC generates code compatible only with the newer version of AIX, which was installed on yellow. It was not surprising that AIC 1.2 would run only under the newer version of AIX.

Installing two versions of AIC illustrates the situation where a development environment needs multiple versions of the very same tool to support applications that can run on multiple versions of a single operating system. It is very similar to the situation where the development environment needs platform-specific tools of the same type, because the application is targeted to run on platforms supplied by different vendors. The compiler, link-editor, and debuggers, for instance, would be unique to each vendor's platform. Though the source code might be identical, the executable binaries would be uniquely generated for each platform.

To ensure a tool is run on a particular host, the SDE WorkBench/6000 user simply specifies the execution host before starting up an instance of that tool class. If the tool is host-scoped, it is possible to have multiple tools of the same class executing simultaneously on different hosts in the environment. Data context can be manipulated separately; therefore the same source files can be specified as input to various compilers in the network, for instance, if the NFS underpinnings are correctly implemented.

7.1.2 Remote Execution of AIC 1.2

Our GUI developer had to ensure that the host *yellow* was added to the `/etc/X0.hosts` file on *sargasso*, thereby enabling X Windows clients executing on that remote host to display their output at the local X server instance. Then, while remotely logged in *bering* where SDE WorkBench/6000 was run, our developer brought up the Tool Manager - Start window. The developer specified `yellow` as the execution host, and selected the `UIBUILD` class of tool. The windows associated with starting AIC 1.2 through SDE WorkBench/6000 are shown in Figure 88.

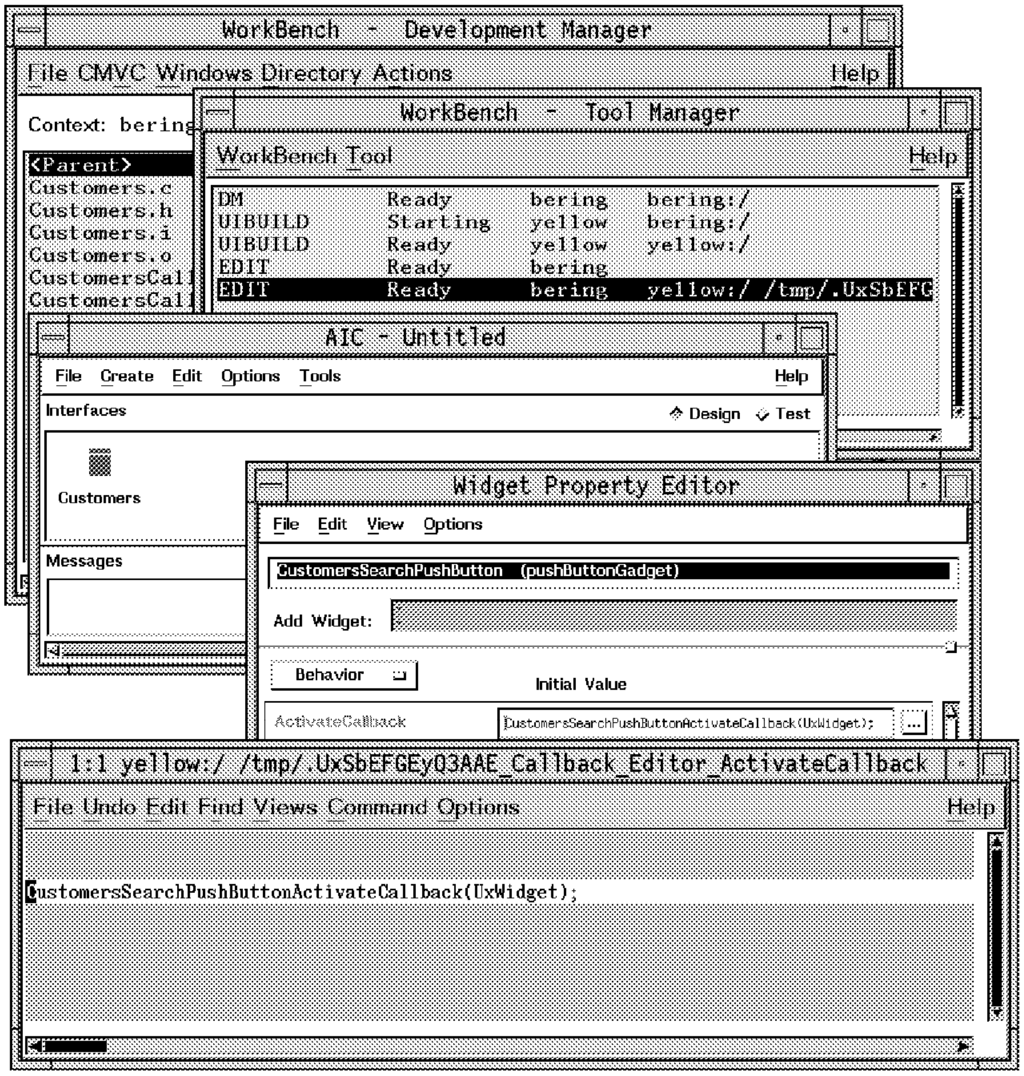


```
TERM        Ready      bering
EDIT        Ready      bering
DM          Ready      bering   bering:/home/aixcase3
UIBUILD     Starting   yellow   bering:/home/aixcase3
UIBUILD     Ready      yellow   yellow:/home/aixcase3
```

Figure 88. AIC 1.2 Started from SDE WorkBench/6000

We were now able to take

advantage of the improved integration of AIC 1. 2 with SDE WorkBench/6000. Our developer could, for instance, now invoke Program Editor directly from the AIC 1.2 widget property editor. AIC 1.2 also provided for BMS interactions with Program Builder to run *makefile* files, and with Configuration Management class tool. This created a richer, more powerful environment. Executing the SDE WorkBench/6000 on *bering*, we could start up AIC 1.2 on *yellow* using Tool Manager, and later AIC 1.2 would automatically cause Program Editor already executing on *bering* to display a callback header file. Figure 89 shows this situation. Notice that the WorkBench Development Manager window shows the host on which the different types of tools are executing. Notice also that each tool is operating in an independently specified data context. The editor text window is operating on a file in `/tmp` on *yellow*, while Development Manager is operating in a subdirectory of the / file SYSSTMT on *bering*.



Context: bering

```
<Parent>
Customers.c
Customers.h
Customers.i
Customers.o
CustomersCal
CustomersCal
```

```
DM          Ready      bering   bering:/
UIBUILD     Starting   yellow   bering:/
UIBUILD     Ready      yellow   yellow:/
EDIT        Ready      bering
EDIT        Ready      bering   yellow:/ /tmp/.UxSbEFG
```

CustomersSearchPushButton (pushButtonGadget)

Add Widget:

ActivateCallback CustomersSearchPushButtonActivateCallback(UxWidget);

1:1 yellow:/ /tmp/.UxSbEFGEyQ3AAE_Callback_Editor_ActivateCallback

```
CustomersSearchPushButtonActivateCallback(UxWidget);
```

Figure 89. Distribution of Various Tools among Multiple Hosts

7.2 Redesigning our User Interface

In developing this second, improved GUI, we chose to follow the graphical user interface standards defined by Common User Access (CUA). This section introduces the principles of object-oriented user interfaces as defined by CUA. This section describes how users move from one window to another, and shows the contents of the new windows. The section also describes the various steps we took to implement the callback subroutines required to link the windows together.

7.2.1 Graphical User Interfaces

Different kinds of user interfaces for interactive applications are available, such as:

- Command-line interfaces, where the user has to type in commands to communicate with the application
- Menu-driven interfaces, where the user follows a hierarchy of organized sets of choices
- Graphical user interfaces, where the user points and clicks to visible elements on the screen by using a mouse.

In order to have the same look-and-feel for multiple applications, various organizations have suggested standards for user interfaces like IBM's CUA definition. The CUA principles of user interface design are based on object-oriented relationships. As a result, users of products with an interface designed according to CUA should find these products easy to use and efficient. The object-oriented structure of CUA also creates a working environment where multiple objects can interact with one another, so the aspect of reuse becomes important and boundaries that distinguish applications from one another are no longer apparent. For more details about CUA refer to *SAA Common User Access Guide to User Interface Design* or *SAA Common User Access Advanced Interface Design Guide*.

The user interface design of the ported legacy application, as described in Chapter 6, “Migrating the Legacy Application to AIX” on page 61, was totally based on compliance with the existing ISPF maps. However, the second step of migration as described in this chapter tries to implement a user interface based on CUA principles. We tried to follow the object-oriented model of CUA and strictly follow the object-action paradigm.

7.2.2 Window Contents and Application Flow

According to CUA guidelines we would have two different primary windows, one for each class of objects the user manipulates with the application, namely*Customers* and *Payments*. Each of these primary windows would have a scrolled list of multiple objects, from which a single entry can be selected. Each primary window would have a menu bar with one pull-down for class-specific actions, such as **New**, **Edit**, or **Delete** and one pull-down for help functions.

The figure shown inFigure 90 on page 137 shows the two primary windows.

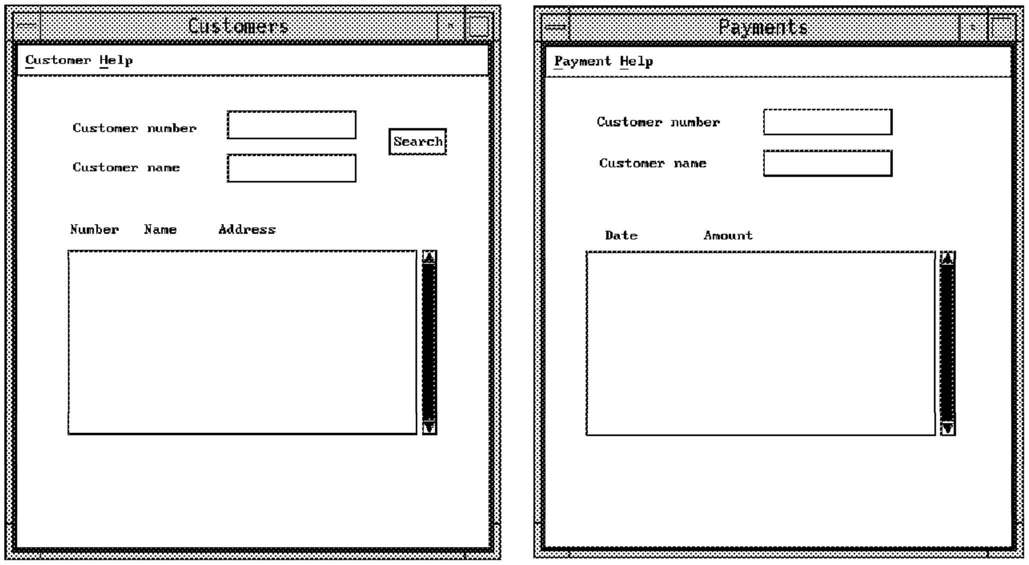


Figure 90. The Primary Windows for the Improved GUI

In addition to the two primary windows, there is one secondary window for each of these. This secondary window would show details for one object, and this secondary window would not have a menu bar. Instead, window help is offered through a push button at the bottom of the window. The figure shown in Figure 91 on page 138 shows the two secondary windows.

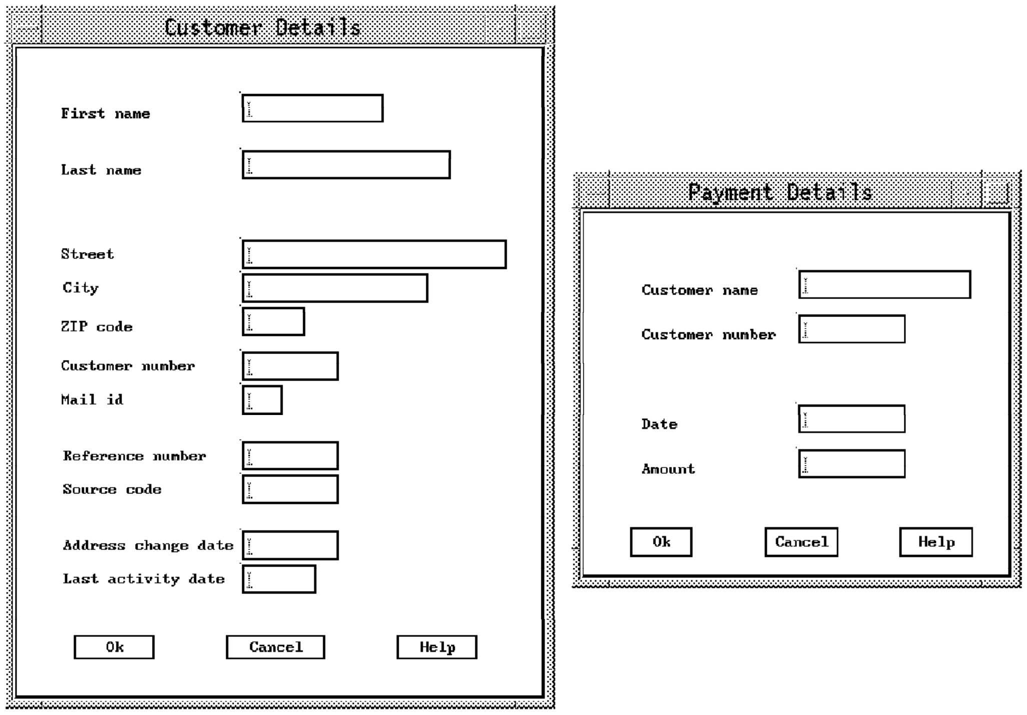


Figure 91. The Secondary Windows for the Improved GUI

The first window presented by the application would show a list of customers, which initially is empty.By entering search criteria, this list can be filled with entries. The user can select one customer in the list and request either the detailed data for that customer or the payments for that customer. In the latter case, the primary Payments window would be displayed with a list of payments made by the previously selected customer. The user can quit the application by selecting **Close** from the window menu button in the title bar of the Customers or Payments window.

The Customers and Payments windows look similar to each other, as do the Customer Details and Payment Details windows.The menu bars, pull-downs, and push buttons also look similar on the two different windows. The learning effort for users is minimal, because users can work with different classes of objects in the same way. In addition, the GUI implementation of these objects, *Payment* and *Customer*, can be reused in another application, because there is no dependency on a specific application logic in the user interface code.

So, even this small example shows the advantage of a user interface based on CUA.

7.3 User Interface Implementation

This section shows the various steps we took to implement the windows we just described and illustrated.

7.3.1 Conventions and Files

The naming conventions we used during development of the improved GUI was very similar to the ones that we used when we developed ISPF-like GUI. These are described in 6.2.1, “Selecting Directory Organization and Naming Conventions” on page 69. We decided to have one directory for each of the windows, and named these directories after the window title. Thus, we created these four directories for the improved GUI:

- Customers
- CustomerDetails
- Payments
- PaymentDetails

Each of these directories contained the interface file for the window that we generated using AIC 1.2. We chose the names of the AIC interfaces to be identical to the names of the windows:

- `Customers.i`
- `CustomerDetails.i`
- `Payments.i`
- `PaymentDetails.i`

We created one callback C source file for each interface.We appended the string *Callbacks*. to the window name and then added the *.c* file type extension:

- `CustomersCallbacks.c`
- `CustomerDetailsCallbacks.c`
- `PaymentsCallbacks.c`
- `PaymentsDetailsCallbacks.c`.

Again, we made widget names unique for all widgets in all the interfaces. Valid widget names began with the window name. For example:

- *CustomersCustomerName*

  or

- *CustomersSearchPushButton*

We named the callback functions after the corresponding widgets and appended a string identifying the type of callback.For example:

- *CustomersCustomerNameModifyVerifyCallback*

  or

- *CustomersSearchPushButtonActivateCallback*.

The only parameter that was passed to each callback was the ID of the corresponding widget.

To enable early compiler checks, and to detect typing errors as early as possible, we also decided to collect the function declarations of all callbacks for each window into a separate header file.We named these header files the same as the corresponding C source files that implemented the callbacks, except that we replaced the *.c* with a *.h*:

- `CustomersCallbacks.h`
- `CustomerDetailsCallbacks.h`
- `PaymentsCallbacks.h`
- `PaymentsDetailsCallbacks.h`.

We also decided to have AIC generate a header file for each generated interface. We did this by setting the *Aic.includeFile* resource to *True* as shown in Figure 146 on page 222. For more information on the AIC resource file, refer to *Installing and Configuring AIC, Version 1.2*.

For the four windows that were implemented as an AIC interface, we created three source files each, and AIC generated an additional include file and a C source file. For example we had the following five source files for the Customers window:

Customers.i	AIC interface source file for the Customers window
Customers.h	Include file generated by AIC with a structure definition for the interface
Customers.c	C source file generated by AIC for the Customers interface
CustomersCallbacks.h	Extern declarations of the callbacks for Customers
CustomersCallbacks.c	Implementation of the callbacks for Customers

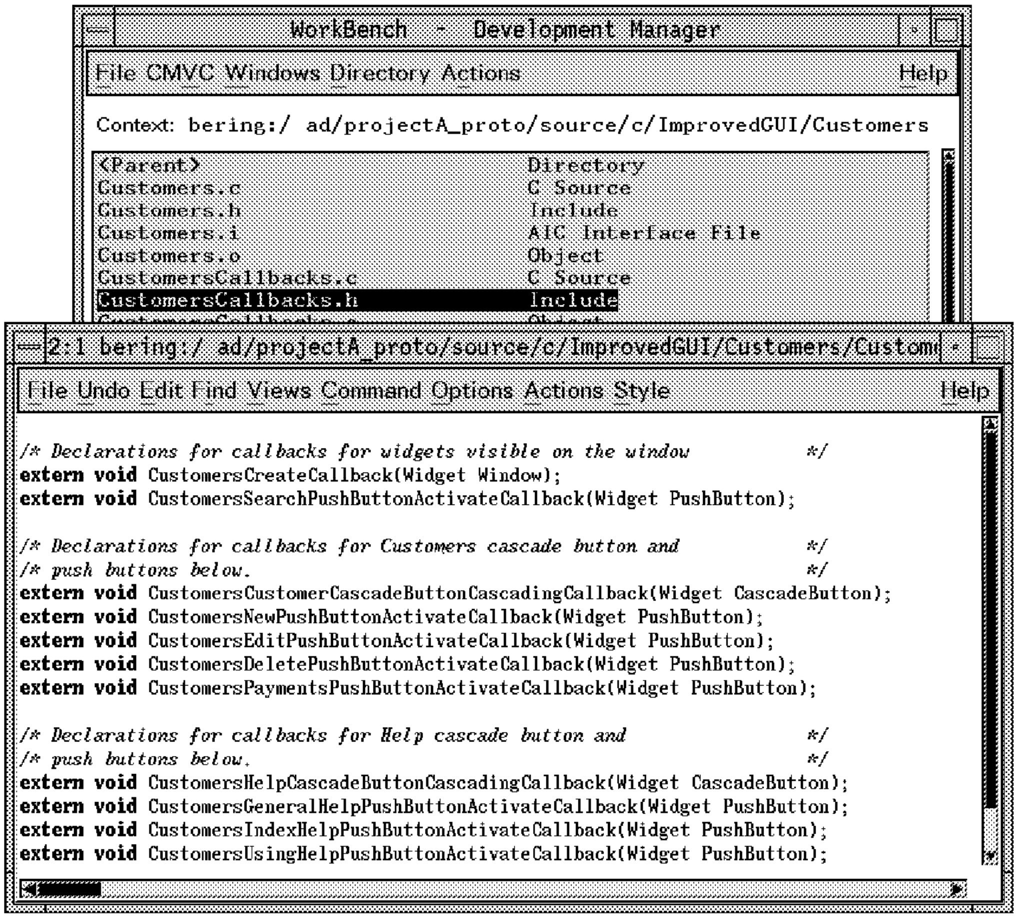
After these conventions were defined, we started the design of the windows for the improved GUI.

7.3.2 Design and Implementation of the Windows Using AIC 1.2

The process of designing the windows shown inFigure 90 on page 137 and Figure 91 on page 138 using AIC 1. 2 is similar to that described in 6.2.2, “Design of the Windows Using AIC” on page 76. We followed the same naming conventions for the improved GUI as we did for the initial implementation of the GUI. Because we designed four windows for the improved GUI, we had four files implementing the callbacks and four header files. These header files contained the *extern* declarations for the functions implemented in each corresponding callback source file. AIC 1.2 generated a header file along with a C source file for each interface, and we had one interface for each window. Thus, we had four source files associated with each window.

As described in 6.2.3, “Implementing the Callbacks” on page 80, we invoked Program Editor and edited the callback source and header files one after another. Because the improved GUI's windows contain menu bars and push buttons, several callbacks needed to be implemented. The calls to the COBOL code that contained the application logic and accessed the database were already in the callbacks of the ISPF-like GUI. These calls just had to be moved into these callbacks.

Figure 92 on page 141 shows a snippet of the callback header file for the Customers window. Figure 93 on page 142 shows a snippet of the callback source code. In both figures, we show the use of Program Editor, invoked from Development Manager, to edit the files.

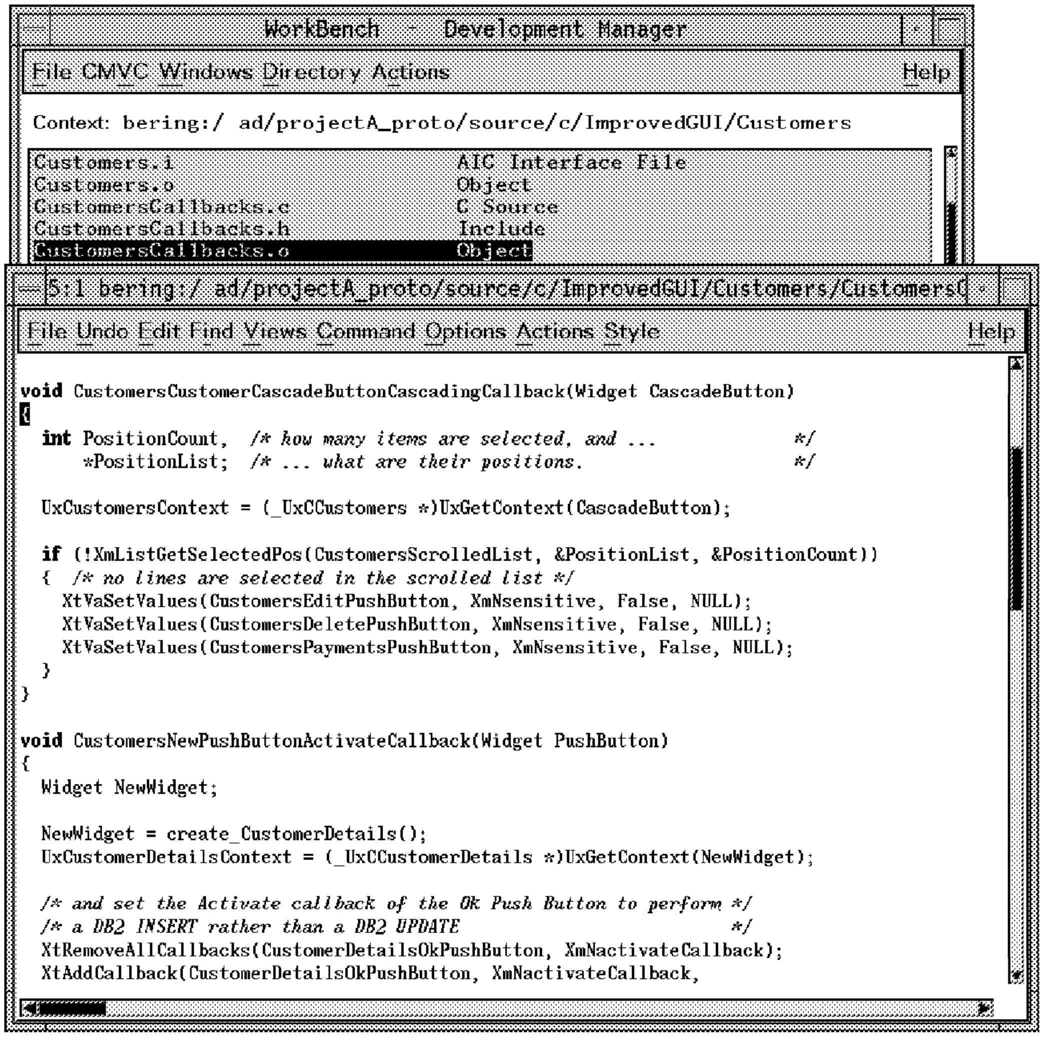

WorkBench - Development Manager
File CMVC Windows Directory Actions Help
2:1 bering:/ ad/projectA_proto/source/c/ImprovedGUI/Custom
File Undo Edit Find Views Command Options Actions Style Help

```
Context: bering:/ ad/projectA_proto/source/c/ImprovedGUI/Customers

<Parent>                        Directory
Customers.c                     C Source
Customers.h                     Include
Customers.i                     AIC Interface File
Customers.o                     Object
CustomersCallbacks.c            C Source
CustomersCallbacks.h            Include
```

```
/* Declarations for callbacks for widgets visible on the window         */
extern void CustomersCreateCallback(Widget Window);
extern void CustomersSearchPushButtonActivateCallback(Widget PushButton);

/* Declarations for callbacks for Customers cascade button and          */
/* push buttons below.                                                  */
extern void CustomersCustomerCascadeButtonCascadingCallback(Widget CascadeButton);
extern void CustomersNewPushButtonActivateCallback(Widget PushButton);
extern void CustomersEditPushButtonActivateCallback(Widget PushButton);
extern void CustomersDeletePushButtonActivateCallback(Widget PushButton);
extern void CustomersPaymentsPushButtonActivateCallback(Widget PushButton);

/* Declarations for callbacks for Help cascade button and               */
/* push buttons below.                                                  */
extern void CustomersHelpCascadeButtonCascadingCallback(Widget CascadeButton);
extern void CustomersGeneralHelpPushButtonActivateCallback(Widget PushButton);
extern void CustomersIndexHelpPushButtonActivateCallback(Widget PushButton);
extern void CustomersUsingHelpPushButtonActivateCallback(Widget PushButton);
```

Figure 92. Editing the Header File for CustomerCallbacks.h

As shown in Figure 92 there are activation callbacks as well as cascading callbacks implemented. We wanted to use the cascading callbacks to set some push buttons to insensitive when there is no item in the list and when there is no item selected.


WorkBench - Development Manager
File CMVC Windows Directory Actions Help

Context: bering:/ ad/projectA_proto/source/c/ImprovedGUI/Customers

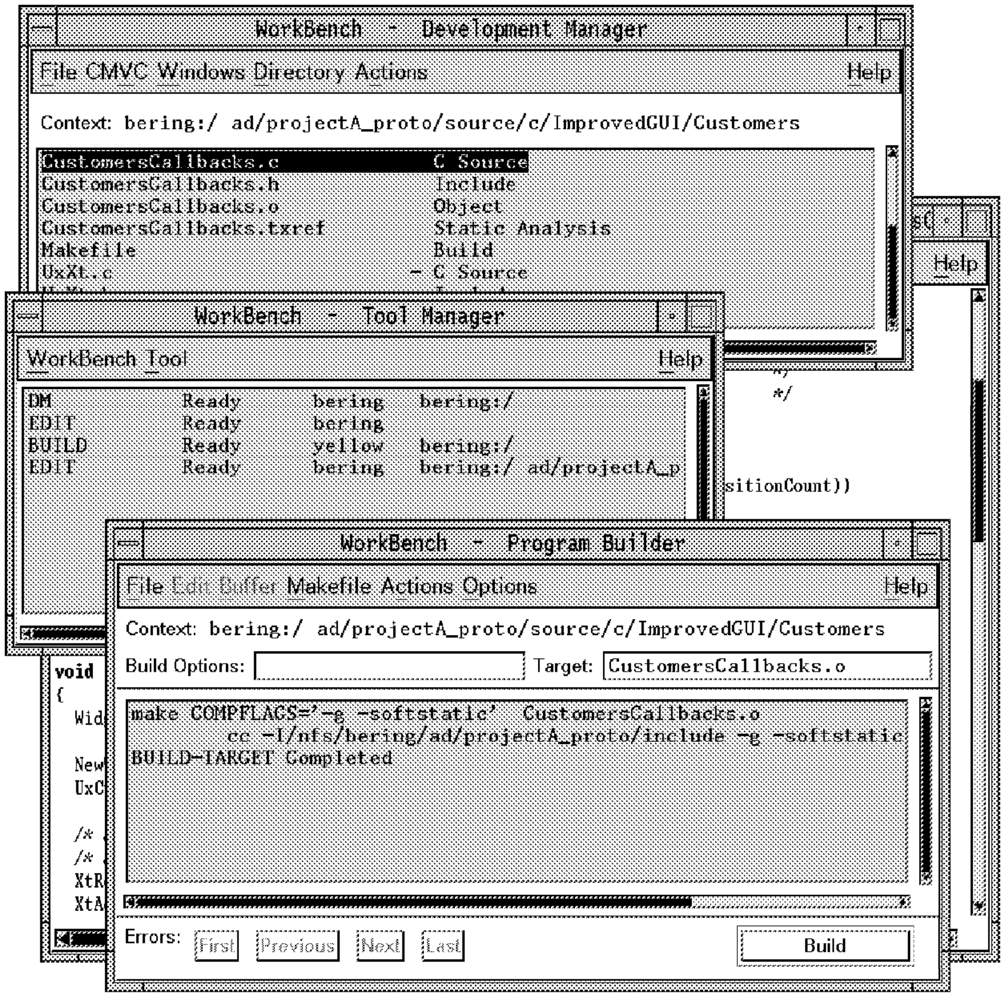
```
Customers.i                      AIC Interface File
Customers.o                      Object
CustomersCallbacks.c             C Source
CustomersCallbacks.h             Include
CustomersCallbacks.o             Object
```

5:1 bering:/ ad/projectA_proto/source/c/ImprovedGUI/Customers/CustomersC
File Undo Edit Find Views Command Options Actions Style Help

```
void CustomersCustomerCascadeButtonCascadingCallback(Widget CascadeButton)
{
  int PositionCount,  /* how many items are selected, and ...            */
      *PositionList;  /* ... what are their positions.                   */

  UxCustomersContext = (_UxCCustomers *)UxGetContext(CascadeButton);

  if (!XmListGetSelectedPos(CustomersScrolledList, &PositionList, &PositionCount))
  {  /* no lines are selected in the scrolled list */
    XtVaSetValues(CustomersEditPushButton, XmNsensitive, False, NULL);
    XtVaSetValues(CustomersDeletePushButton, XmNsensitive, False, NULL);
    XtVaSetValues(CustomersPaymentsPushButton, XmNsensitive, False, NULL);
  }
}

void CustomersNewPushButtonActivateCallback(Widget PushButton)
{
  Widget NewWidget;

  NewWidget = create_CustomerDetails();
  UxCustomerDetailsContext = (_UxCCustomerDetails *)UxGetContext(NewWidget);

  /* and set the Activate callback of the Ok Push Button to perform */
  /* a DB2 INSERT rather than a DB2 UPDATE                          */
  XtRemoveAllCallbacks(CustomerDetailsOkPushButton, XmNactivateCallback);
  XtAddCallback(CustomerDetailsOkPushButton, XmNactivateCallback,
```

Figure 93. Editing the Callback Implementation for CustomerCallbacks.c

As we did for the ISFP-like GUI, we saved the files when editing was complete, and started Program Builder using Tool Manager. In this case, it would not matter which execution host we specified for Program Builder, so long as it had access to the files specified as its data context through proper NFS underpinning. Figure 94 on page 143 shows, how Program Builder was started on *yellow* with a data context of the `/ad/projectA_proto/source/c/ImprovedGUI/Customers/` on the host *bering* and a build target of CustomersCallbacks.o. The source file was actually on *bering*, but made available through NFS to Program Builder which looked for it by prefixing the file's path name with `/nfs/bering/`.


WorkBench - Development Manager
File CMVC Windows Directory Actions Help
WorkBench - Tool Manager
WorkBench Tool Help
WorkBench - Program Builder
File Edit Buffer Makefile Actions Options Help
Errors: First Previous Next Last
Build

Context: bering:/ ad/projectA_proto/source/c/ImprovedGUI/Customers

```
CustomersCallbacks.c            C Source
CustomersCallbacks.h            Include
CustomersCallbacks.o            Object
CustomersCallbacks.txref        Static Analysis
Makefile                        Build
UxXt.c                        - C Source
```

```
DM          Ready     bering    bering:/
EDIT        Ready     bering
BUILD       Ready     yellow    bering:/
EDIT        Ready     bering    bering:/ ad/projectA_p
```

Context: bering:/ ad/projectA_proto/source/c/ImprovedGUI/Customers

Build Options: Target: CustomersCallbacks.o

```
make COMPFLAGS='-g -softstatic'  CustomersCallbacks.o
        cc -I/nfs/bering/ad/projectA_proto/include -g -softstatic
BUILD-TARGET Completed
```

Figure 94. Compiling CustomerCallbacks.c on the Host yellow

After all the callbacks had been implemented and compiled successfully, Program Builder was started to link all intermediate files and to create an executable program. Program Builder had to link the four compiled files generated by AIC, the callback files we implemented, and all the libraries that are required for the prerequisite software products and for system functions.

Chapter 8. Object-Orientation of Our Legacy Application

This chapter describes our effort to convert the legacy application into an object-oriented program. It describes the design of the application class hierarchy, and highlights how this code interfaces with both the DB2/6000 database and the code generated by AIC. It also discusses how we used AIC 1.2 to generate C^{++} code and integrate that generated code with the application logic. This chapter begins by taking a look at the ongoing development of objected-oriented technology and the relationship between it and the evolution of user interface programming.

8.1 Using SDE WorkBench/6000 and Integrated Tools for Object-Oriented Development

One of the original goals of the project was to see how well SDE WorkBench/6000 and the integrated tools supported object-oriented development. We found that the C^{++} programmer interacted with SDE WorkBench/6000 and most of the standard integrated tools in the same manner as the C programmer, deriving exactly the same benefits and advantages we have already described.

The edit, compile, and debug process for the C^{++} developer differs very little from the process for the C developer. Program Editor recognizes C^{++} source code keywords and lexical symbols just as it does with C source code. The XL C^{++} compiler does, however, include a graphical class browser that the C compiler does not require. Program Builder works identically. Program Debugger enables C^{++} source level debugging, just as it does for C. In 9.3, “Tailoring SDE WorkBench/6000 for C^{++} Programmers” on page 207 we describe how we customized SDE WorkBench/6000 for our C^{++} developer.

The basic use of AIC 1.2 to design an interface does not change significantly whether AIC is used to generate C or C^{++} source code. There are increased benefits derived from the improved integration of AIC 1.2 with SDE WorkBench/6000. The programmer must learn how AIC defines classes and methods, and how to write C^{++} application code that is integrated with the GUI acceptable to AIC's C interpreter. A few things are different in setting up AIC 1.2 to generate C^{++} source code. This chapter focuses on these differences.

In 9.5, “Tailoring AIC” on page 221, we describe how we tailored AIC 1.2. In 9.4, “Tailoring SDE WorkBench/6000 for AIC Programmers” on page 212 we describe how we customized SDE WorkBench/6000 for an AIC developer. Finally, we discuss how our GUI developer gained accesss to AIC 1.2 across the network in 7.1.1, “Multiple Tools in a Single Tool Class on Separate Hosts” on page 133.

8.2 Object-Oriented Technology and Graphical User Interfaces

Object-oriented technology is not just a new way of extending current software development technology—it is conceptually different. Object-oriented technology was developed in response to difficulties encountered by software engineers and programmers as the problems they attacked and the solutions they devised became increasingly larger and more complex, had increasingly rigorous reliability requirements, and became unacceptably expensive. The development and acceptance of object-oriented programming is linked with the concurrent evolution of graphical user interfaces. This section briefly examines this relationship and the development of both technologies.

8.2.1 Classes and Objects—the Basic Building Blocks

Objects are the basic building blocks of object-oriented programming. Software entities are built of objects, and can also be an object. An object is a collection of data and application program code that operates on that data. Objects are sent messages requesting that they perform a particular functionality of which they are capable. Objects model reality, in that the programmer, or software, that utilizes an object is not required to “know” how the object performs its function, simply how to request the performance. For example, a TV is an object. It “knows” how to display a picture, change a channel, show the time, and other such operations. To use a TV, we do not have to know or understand how it works; we only need to know how to issue a simple set of commands.

A class is a generalization, of which an object is an instance. It is used to generalize the characteristics common to all of its instances. If, for example, we wanted to gather information about different characteristics of operating systems, we would establish a class, *operating system*. Then, we would be able to manipulate objects representing AIX, OS/2, VM, MVS, and MS-DOS as instances—objects—of this class, and we would be able to treat them uniformly regardless of their differences.

8.2.2 Evolution of User Interfaces

One of the first and most successful uses of the object-oriented paradigm has been with the software technology that made graphical user interfaces possible. Having a GUI is fast becoming an absolute requirement for new applications under development. Many older applications are being given a modernized look-and-feel with the addition of a GUI. GUIs have evolved from very primitive and much less successful user interfaces. Their evolution provides an excellent example of the successful application of object-oriented programming principles to software engineering.

The first computers had very poor user interfaces. Commands were issued in batches on punched cards, and results were presented on paper printouts. With the appearance of terminals, user interfaces became interactive. A program issued a message, waited for a user to respond, and reported the result back to the user. Such “command-line dialogs” were repeated over and over again.

Command-line interfaces were replaced by screen-based user interfaces. These screens had input and output fields in fixed positions on the screen. They had a fixed position on the screen where messages to the user could be displayed, and a fixed position where commands could be entered also. The cursor moved among these fields to indicate to the user where input was currently being accepted.

As applications became more complex, a single screen was found to be too small to hold all of the fields an application might need. But it was also discovered that for a specific user interaction, the application did not need to have all possible fields displayed simultaneously. Character-based windows were created as a way of dividing the total number of fields into appropriate subsets for given user interactions. Programmable function keys were created to replace numerous and complicated sets of commands. Function keys also provided mechanisms for moving the cursor among various fields and for moving between the various windows. The combination of alphanumeric and programmable function keys, along with character-based screens, provided a simple repertoire of mechanisms for developing user interfaces. Although simple for the programmer to code, they were difficult for the application user to use.

With the introduction of the mouse (and other types of pointing devices) and the graphical (all-points-addressable) display screen, it became possible to develop graphical user interfaces. GUIs were quickly found to be both intuitive and easy to use. Through a GUI, the programmer could facilitate user interactions with applications that model real-life objects and actions on these objects.

Virtual desktops—populated with icons, windows, and dialog boxes— were developed. Icons, visual metaphors of reality, were designed to mimic real objects, not only visually, but functionally as well. The user could now use the mouse to *push* a graphical representation of a *push button* to change the state of some object. The user could also use the mouse to *drag* an icon representing a file folder across a virtual desktop to another icon representing a trash can and then *drop* it to indicate its removal. The icon representing a printer “knew” how to print a document when an icon representing that document was dropped on it.

The advent of GUIs offered many more choices than before in application user interface engineering, but it also presented the programmer with more complex and difficult challenges. One challenge was the proliferation of variety in GUIs being developed. Each new GUI was developed from scratch, borrowing design principles but not reusing code from other GUI examples in the developer's experience. GUIs were created for specific operating systems and for specific flavors of a common operating system, UNIX. This created the need for standardization of both the visual vocabulary (types of visual objects and their associated behaviors) and of the broader grammar (“look and feel”) of GUIs.

Efforts at MIT** to develop the X Window System and Project Athena, or at OSF to popularize and widely distribute OSF/Motif were undertaken consciously to eliminate this chaos. The visual objects were organized into hierarchies whereby complex objects were built up from (or derived from) less complicated objects, and could be presumed to inherit qualities from their predecessor objects. Style guides were developed to ensure that developers made appropriate use of the objects provided in the GUI libraries.

This trend assured reusability and portability, and enabled the GUI developer to concentrate on the needs of the application, instead of developing the mechanics of graphically presenting and accepting information.

Another challenge was the inherent difficulty in programming an application's GUI, even when using a standard library of GUI subroutines and advised by a style guide. GUI code is characterized by long and complicated variable and function naming conventions, a complex hierarchy of objects, and it turns the traditional program structure inside out. While the traditional program's flow of control is envisioned as one in which the application logic is at the core, and the user interface code fragmented at the extremities of logic paths, the typical graphical application consists of a central event-driven user interface loop, with the program logic fragmented at the extremities.

Because of these challenges, GUI programming did not really begin to spread until GUI builders were developed. GUI builders are tools that allow the developer to design the GUI visually and interactively, and then generate automatically the complex and difficult source code that implemented the design. GUI builders eliminated much of the difficulty of programming GUIs. Typically, GUI builders generate C source language because the underlying GUI libraries of which they make use were implemented in C. Most of the newly developed applications being developed for the open systems platforms were also being written in C; so this was an obvious language choice for the vendors developing GUI builders.

8.2.3 GUIs Demonstrate Object-Oriented Principles

The modern GUI demonstrates several characteristics that are considered hallmarks of object-oriented programming:

- Modelling of a realty in software
- Clustering together functions (behavior) with visual entities (appearance)
- Class hierarchies from which instances are created
- Inheritance of characteristics from base classes
- Communication between objects

Likewise the maintainability, reusability, and manageability of the GUI software has focused attention on its underlying object-oriented nature.

8.2.4 Object-Oriented Application Development and GUIs

Object-oriented languages evolved in parallel with, but independently of, GUI technology. One object-oriented language, C^{++}, has gained the widest acceptance. This language is highly compatible with C, the language in which most of the GUI libraries and application code has been written.

The accepted superiority of GUIs over traditional user interfaces proved that significant benefits could be derived from the object-oriented programming principles, though neither GUI libraries nor GUI code was typically written in an object-oriented language. This proof was not sufficient to cause the widespread introduction of object-oriented languages, or even object-oriented practices with traditional languages, however. The bulk of commercial, technical, and scientific application programming continued along traditional lines, using non-object-oriented technology. Meanwhile, object-oriented methodologies, design tools, language products, and databases continued to mature and proliferate.

As programmers came to understand object-oriented principles, they began to recognize that combining an object-oriented GUI component with a non-object-oriented application code and data component creates difficulties for the programmer and strains the object-oriented paradigm of the GUI technology. They concluded that these difficulties were only going to be resolved by extending the object-oriented paradigm to the application logic itself.

Vendors of GUI builders reached this same conclusion and began recently to make their GUI builders capable of generating C^{++}. For the first time it became practical to easily generate GUI code and application logic code in the same object-oriented language, C^{++}. Even though the underlying GUI libraries are still implemented in C, implementing the application's GUI in C^{++} makes very good sense from this perspective.

AIC 1.2 can generate C^{++} source code from the same GUI design information, from which it generates C source code. This makes it possible to take an existing interface design, regenerate the GUI code in C^{++}, and link it with a new application logic written also in C^{++}. How to integrate the C^{++} GUI code generated by AIC 1.2, with application code written by ourselves in C^{++}, was a central question we wanted to explore in this last phase of our project.

8.2.5 Continuing the Evolution of Object-Oriented Technology

As more object-oriented applications have been developed, the problems encountered integrating object-oriented applications with non-object-oriented operating systems has pointed out the need for inherently object-oriented operating systems. In theory, object-oriented applications should be built of objects, which themselves should be built of objects, and so on. As more and more objects are built, greater and greater numbers of applications can be built from reused objects in the application's execution or compilation environment. The fundamental objects used in this process should be the operating system resources on which all applications depend.

However, the object-oriented nature of applications currently being written break down at the level of operating system calls and subroutines. The interface to the traditional operating system services is not one of objects, but of traditional non-object-oriented programming.

IBM and other operating system vendors recognizing this have made investments in the development of object-oriented compiler and operating system technologies. The development of System Object Model (SOM) and Distributed System Object Model (DSOM) support in compilers, within a single operating system and across network environments, is a first step in this direction. The Taligent effort, an attempt to build a commercial operating system from the ground up using object-oriented technology, is the logical extension of this technology. When an application executes in an object-oriented operating system environment, such as Taligent promises, it will access operating system resources in exactly the same manner as it accesses other program resources.

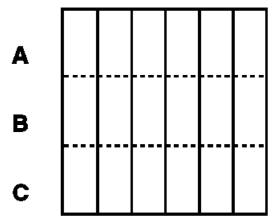
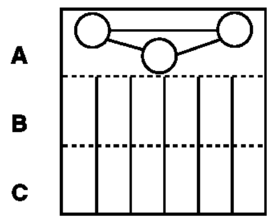
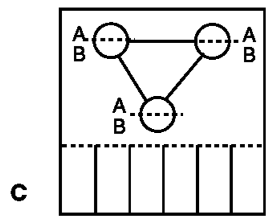
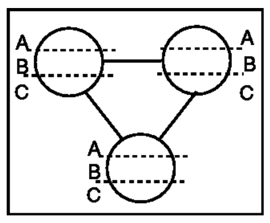
Successful, large-scale code reuse can be achieved only when programmers can build applications from preexisting objects in the object-oriented operating environment. This will be possible only when object-oriented operating systems come onto the market. Object-oriented operating systems represent the future direction of software engineering, according to many experts today. Once object-oriented operating systems are in place, the evolution of object-oriented technology will have reached the point where all of the benefits promised by the technology can be realized.

Table 4 on page 150 summarizes the characteristics of object-oriented systems at different stages of their evolution. Three layers—A, B, and C—denote three distinct components of software products as we perceive them today:

- A—GUI
- B—data and application code
- C—operating system interface.

As systems move toward inherently object-oriented systems, the layers as such disappear, and their function becomes incorporated in objects, thus simplifying the programming paradigm and making it more powerful for developers and users alike. By redesigning our application in C^{++} and using AIC/6000 to generate its GUI, we intended to advance our legacy application along this evolutionary path as far as it could go, until it can be rehosted on an object-oriented operating system.

Table 4. Evolution of Object-Oriented Systems

Non-Object-Oriented	**Object-Oriented GUI**	**Object-Oriented Applications**	**Inherently Object-Oriented**
			
Legend: A - GUI, B - data and application code, C - operating system interface			
No notion of objects in application.	Objects are distinctive only in GUI part of application.	Objects are part of application.	Objects are part of object space.
All communication has to be hard-coded into application (for example, by means of shared memory, sockets, and other communication mechanisms).	GUI enables “primitive” interobject communication (for example cut-and-paste between different fields).	Interobject communication among objects in same application is possible and easy, but it is more difficult among objects from different applications (for example drag-and-drop of icons, fields).	Interobject communication always possible and easy.
Application must store execution data in flat files or databases.	GUI objects are mapped to data structures, and it is the application's responsibility to store data in either flat files or databases.	Objects are transient, they exist only during execution of the application, and it is the application's responsibility to store data about objects from one execution to another in flat files or databases, or to store objects in an object-oriented database.	Objects are persistent; they take care of all the data they need, as long as they exist in the object space.
Code reuse by means of developing and using common libraries.	Code reuse of GUI elements.	Code reuse of classes in different applications.	Code reuse is the way of developing new objects.

8.3 Our Object-Oriented Design Effort

We did not start out from scratch designing an object-oriented application. We accepted several constraints on our development effort, including non-object-oriented interfaces to the operating system, to our data base through the DB2 CLI for C, and a desire to reuse the improved GUI design that we had just itegrated with the existing COBOL application logic. We also did not have the time to completely reanalyze the customer's complete problem domain, nor did we set out to design and develop a generalized application that might be independent of its underlying database and operating system technology.

Our effort illustrates a practical approach to gradually incorporating object-oriented principles and technologies with an existing application heritage. It addresses several issues related to the use of AIC to generate GUI code in C^{++} and link that GUI code with separately developed C^{++} application code.

This section explains the fundamental design we used to restructure the legacy application on an object-oriented model. It also explains the decisions we made about the design and hierarchy of the application classes. Additionally, it shows how the code, generated by AIC, was combined with new code we wrote to replace the COBOL code.

8.3.1 Our Base and Derived Classes (Class Hierarchy)

Figure 90 on page 137 and Figure 91 on page 138 are pictures of what users of our *Improved GUI* application will see on their screens. The four windows, Customers, Customer Details, Payments, and Payment Details, focus the user's attention on two types of entities: customer and payment. These represent the *classes of objects* that the users of our application are familiar with from their work. The Customer and Payment windows present the user with limited information about several objects in each class. The Customer Details and Payment Details windows show only one object of each class, but show all the characteristics of that object. From these windows, users perform actions on these objects in a predefined number of ways. From this perspective, our enhanced GUI is object-oriented. Therefore, we decided to use the objects defined in the GUI design as a basis for our design of an object-oriented version of this application.

From the point of view of AIC, a project represents a set of interfaces. The code AIC generates to implement these interfaces constitutes the application. Each interface represents a type of window. The interface is implemented as a unique class in the C^{++} source code that AIC generates.

In the simplest case, all the application logic is embedded within each interface, tied to the callback functions that define the behavior of the widgets in the interface. When an object of that class is instantiated, an actual window appears on the screen. Multiple windows of the same interface can be displayed at one time; therefore there can be multiple instances (objects) of a given class, each more or less self-contained (data and functionality).

In a more complex case, the developer would want to have the user interface and the application logic separated and independent of one another. There are several reasons for this, including:

- The GUI and application logic can be developed and tested independently of one another.
- The GUI is reusable with other applications.
- The application logic is reusable with other GUI implementations.
- Maintenance of the GUI and application are simplified.

In this case, the developer would not embed application logic by means of the callback functionality in the interface. Instead, the developer would embed debugging functionality in the GUI, so the GUI code could be tested independently. The developer would create one base class implementing the application logic corresponding to each base class (interface) in the GUI. Non-GUI test programs would be coded for testing these application logic classes. One application class would be written for each interface that derives from the two base classes corresponding to that interface. The two base classes “know” nothing about each other, but the application class—derived from each of these base classes—can make use of data and methods (functionality) inherited from them both. The derived class can also override these inherited methods, creating unique functionality that neither base class exhibits. The “application” would consist of a set of these derived classes, one for each interface in the project.

We chose to follow the second strategy, which is suited for a larger scale development effort than our three-person project, because it gave the potential to reap more of the benefits promised by object-oriented technology.

Our *Improved GUI* design, consisted of four interfaces which corresponded to the Payments, Payment Details, Customers and Customer Details windows. The design included four application classes:

- *Payments*
- *Payment* (Payment Details)
- *Customers*
- *Customer* (Customer Details).

Each of these application classes were derived from one of four unique GUI base classes:

- *PaymentsGUI*
- *PaymentGUI* (Payment Details)
- *CustomersGUI*
- *CustomerGUI* (Customer Details)

and from one of four unique application logic base classes:

- *PaymentsApp*
- *PaymentApp* (Payment Details)
- *CustomersApp*
- *CustomerApp* (Customer Details).

Figure 95 on page 153 illustrates the relationship between the base, derived, and component classes implementing the Payment Details and Customer Details windows. This illustration, based on Coad-Yourdon notation conventions, shows that the application class *Payment* is derived from the GUI base class *PaymentGUI* and from the application logic class *PaymentApp*. It also shows how the derivation of the application class *Customer* similarly. These GUI base classes, defined in the C^{++} source code that was generated by AIC, are named slightly differently in the actual source code. The application logic base classes, implemented in C^{++} source code by our developers, are named exactly as shown in the illustration. The application classes that derive from the base classes, implemented by our developers in C^{++}, are also named exactly as shown.

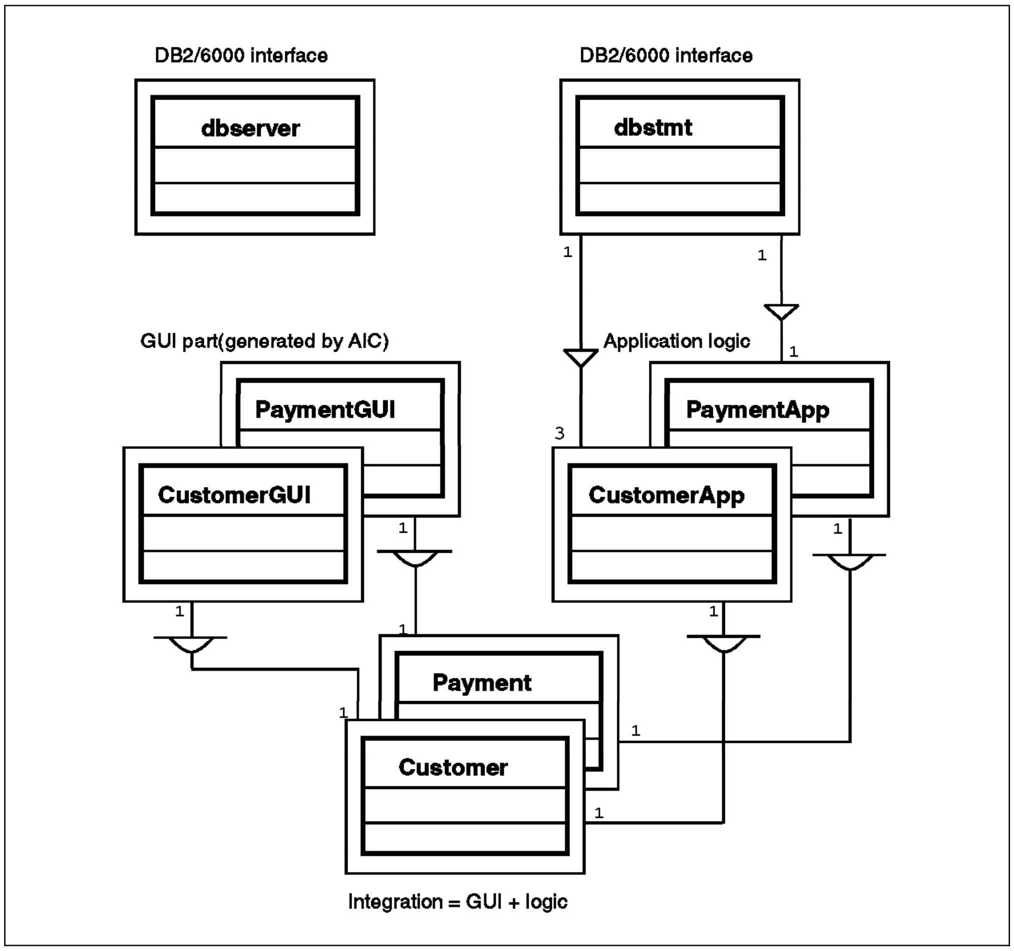


Figure 95. The C++ Implementation Class Hierarchy

8.3.2 Other Relationships between Objects

Now that we have chosen classes and defined a class derivation hierarchy for our application, we need to examine other relationships among the classes of our application. The relationship between classes follows the relationship between the real life entities they represent. For instance, one customer can make many payments, but a given payment can only be made by a single customer.

Figure 96 on page 154 illustrates this entity relationship, using Coad-Yourdon notation conventions.

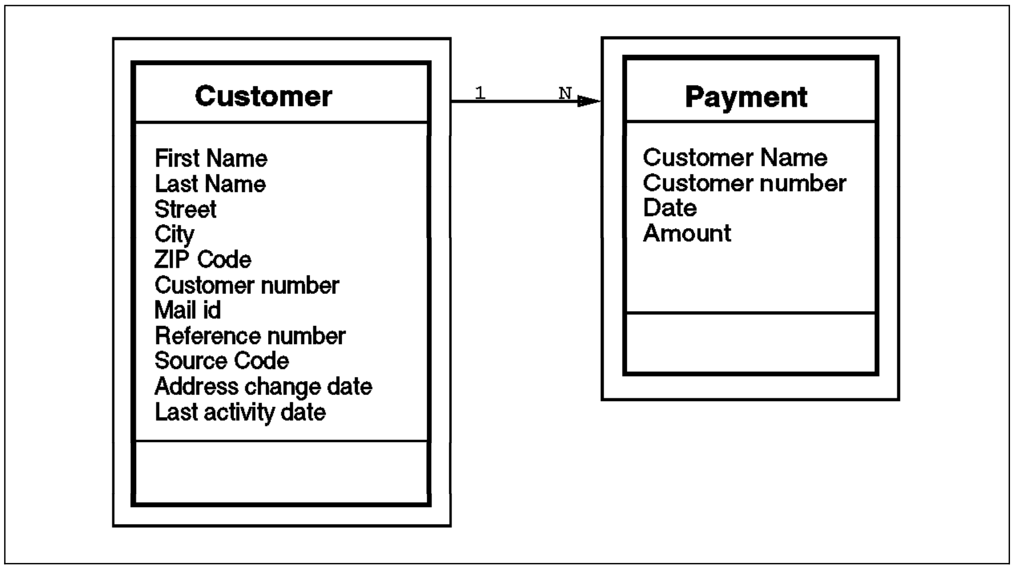


Figure 96. The Relationship between Customer and Payment Classes

8.3.3 Component Classes

Other classes were also required to implement our application. These classes enabled the application logic classes, but were not be in any way unique to the specific application logic classes. Called “component” classes, these classes are often general purpose and can be reused by other application logic classes.

We contemplated two categories of component classes in designing our object-oriented version of this application. The most necessary category was a set of classes with which to wrap object-oriented code around the DB2 CLI for C. The second category was a set of classes with which we could describe the data stored within the database.

In 8.4, “Our DB2/6000 Object-Oriented Interface” on page 155 we discuss the classes we implemented to access DB2/6000. These classes are illustrated in Figure 95 on page 153. The relationship between these component classes and the application logic classes is not always as direct and easy to illustrate as the hierarchical relationship between base and derived classes. In the following paragraphs we describe other candidates for component classes that we chose not to implement.

Figure 96 shows attributes associated with the two classes. Some attributes have similar characteristics and can be grouped accordingly:

Character strings	First name, Last name, Street, City, Mail ID, Source code, Customer name
Numerical values	ZIP code, Reference no, Customer number, Amount
Dates	Last activity, Address change, (Payment) Date.

Such grouping indicates possible classes (component classes) from which Customer and Payment classes could be built. Classes like *characterString* and *Date* are good examples of component classes — building blocks for other classes. This type of class would typically be found in general purpose class libraries. These would be purchased commercially, or developed in-house, if significant reuse on subsequent projects was anticipated. We did not feel an investment in this type of class library was necessary for this project.

8.3.4 Object Behavior and Interactions in Our Application Design

At execution, an object of the derived class corresponding to the main window type, or primary interface, of the project would be instantiated through execution of the main module (generated by AIC 1.2). From the user's perspective, the main window would be displayed when the program starts. Ensuing user actions would trigger callback functions in this derived class that were inherited from its GUI class. Callback functions, which affect GUI behavior, might also cause the execution of application logic inherited from its application logic class, which in turn would result in database query results being displayed.

Ultimately, callbacks could cause the dynamic instantiation of one or more objects of the remaining derived classes. In other words, as a result of user interactions with the main window, the database would be queried, results displayed, and multiple copies of the remaining windows would be displayed as needed.

8.4 Our DB2/6000 Object-Oriented Interface

One group of component classes that our application logic needed was a set of classes to provide an object-oriented interface to the DB2/6000 relational database. Developing this set of classes seemed a good investment because it seemed likely to be reused on future projects, and it illustrated the general problem of making a non-object-oriented programming interface available to an object-oriented application. In the future, if all database access for our applications were restricted to these interface classes, then any changes in our database interface would be localized and we could prevent applications from being affected by such changes. Similarly, to enable applications to make use of an alternate database management system (DBMS), we would need only to provide a set of such interface classes. Our applications would not require modifications. With the growth of object-oriented systems, it is likely that such interface classes soon will be provided either by database vendors themselves, or by third party companies, specializing in development of class libraries.

The database interface classes we developed for DB2/6000 are:

dbserver	This class is used to establish connection to the DB2/6000 server; to allocate and initialize a communication area prior to the startup of the application; and to do cleanup, freeing the DB2/6000 resources on completion of the application.
dbstmt	This class is used for preparing and executing DB2/6000 SQL queries and making the results available to the application.
dberror	If an error occurs during the DB2/6000 session, the appearance of an object of this class contains detailed information about the error condition, enabling the application to decide how to proceed.

8.4.1 DB2/6000 C^{++} Classes

Classes for DB2/6000 services were built using the DB2 CLI for C. *IBM DATABASE 2 AIX/6000 Call Level Interface Guide and Reference* explains in detail how DB2 CLI works in the C environment. C^{++} is a superset of C; so it was easy and straightforward to use the DB2 CLI for C in our C^{++} code.

Our choice of classes to encapsulate the DB2 CLI for C services was based on the fact that some functions in the DB2 CLI for C are effective for an entire application, whereas other functions are oriented toward single SQL statements. The class *dbserver* encapsulates functions that provide services for the whole application (for example, establishing and releasing the connection to DB2/6000). DB2 CLI for C functions that provide SQL services are encapsulated in the *dbstmt* class. They all have an SQL statement handle (a variable of *SQLHSTMT* type) as the first parameter in their parameter list (except `SQLAllocStmt`, which allocates a new statement handle).

Our encapsulated functions have the same names as the corresponding DB2 CLI functions. The parameters are in the same order, except that parameters that are known to the class are omitted.

For error handling, we have developed the *dberror* class, instances of which are created and run when an error condition is encountered. All DB2 CLI functions report their success or failure in their return code. Typical error codes and our code's resulting actions are as follows:

SQL_SUCCESS	No action; execution continues.
SQL_SUCCESS_WITH_INFO	Print a message; execution continues.
SQL_NO_DATA_FOUND	Print a message; execution continues.
SQL_ERROR	Print a message; an exception is triggered and the execution is interrupted.
SQL_INVALID_HANDLE	This error should never occur. If it does, we proceed as in case of SQL_ERROR.

Header comments from our code samples are not included in the figures because they take up a lot of space, and are really not relevant to our discussion.

Names of all protected class members start with an underscore (_), so they can be easily differentiated from publicly accessible members, names of which start with a character.

8.4.2 Class dbserver

This is a top level class used to connect our application to the DB2/6000 database. The application needs at least one object of this class to connect to the database. It was not clear to us, from our reading of *IBM DATABASE 2 AIX/6000 Call Level Interface Guide and Reference* whether one such object is all that is needed, or one is needed for each separate database instance that the application connects to simultaneously. Resolution of this issue was not necessary for this project, as we used only one database instance. The class declaration is shown in Figure 97 on page 157.

```
#ifndef dbserver_h
#define dbserver_h

#include "iostream.h"

extern "C"                                                        1
{
#include "sqlcli.h"
}

class dberror;                                                    2

class dbserver
{
  protected:
    typedef enum
    {
      NAME_LEN      =  20,
      STATE_LEN     =   6
    } _info;

    SQLHENV      _henv;  // environment handle
    SQLHDBC      _hdbc;  // connection handle
    SQLRETURN    _rc;    // return code

    SQLCHAR      _server_name[NAME_LEN+1],
                 _user_name[NAME_LEN+1],
                 _password[NAME_LEN+1] ;

  public:

    dbserver(const char* serv_name,
             const char* user_name,
             const char* password) throw(dberror);

    virtual ~dbserver(void) throw(dberror);
    SQLHENV get_henv(void) const;
    SQLHDBC get_hdbc(void) const;

};

#endif
```

Figure 97. dbserver.h, dbserver Class Declaration

To use the DB2 CLI for C, the appropriate header file, `sqlcli.h`, must be included in the source code. C++ enables C header files to be included, but it requires that the include directive be enclosed in an `extern "C" {}` statement (1).

We need to make a forward declaration of the class *dberror* (2), to be able to specify that an instance of the *dberror* class will be thrown in case an exception occurs.

The implementation of the class *dbserver* is shown in Figure 98 on page 158.

```
#include <stdio.h>
#include <iostream.h>
#include <iomanip.h>
#include <string.h>

#include "dbserver.h"
#include "dberror.h"

dbserver::dbserver(const char* server_name,
                   const char* user_name,
                   const char* password) throw(dberror)
{
  strncpy((char *)_server_name,server_name,NAME_LEN);
  strncpy((char *)_user_name,user_name,NAME_LEN);
  strncpy((char *)_password,password,NAME_LEN);

  _rc = SQLAllocEnv(&_henv);                                    1

  if(SQL_ERROR == _rc)
    throw(dberror(this,"dbserver::dbserver SQLAllocEnv"));

  _rc = SQLAllocConnect(_henv,&_hdbc);                          2

  if(SQL_SUCCESS != _rc)
    throw(dberror(this,"dbserver::dbserver SQLAllocConnect"));

  _rc = SQLConnect(_hdbc,                                       3
                   _server_name,SQL_NTS,
                   _user_name, SQL_NTS,
                   _password, SQL_NTS);
  switch(_rc)
  {
    case SQL_SUCCESS:
      break;
    case SQL_SUCCESS_WITH_INFO:
      cout << dberror(this,"dbserver::dbserver SQLConnect info").msg();
      break;
    default:
      throw(dberror(this,"dbserver::dbserver SQLConnect"));
  }
}
```

Figure 98 (Part 1 of 2). dbserver.C, dbserver Class Implementation

```
dbserver::~dbserver(void) throw(dberror)
{
  _rc = SQLTransact(_henv,_hdbc,SQL_ROLLBACK);                   4

  switch(_rc)
  {
    case SQL_SUCCESS:
      break;
    case SQL_SUCCESS_WITH_INFO:
      cout << dberror(this,"dbserver::~dbserver SQLTransact info").msg();
      break;
    default:
      throw(dberror(this,"dbserver::~dbserver SQLTransact"));
  }

  _rc = SQLDisconnect(_hdbc);                                    5

  switch(_rc)
  {
    case SQL_SUCCESS:
      break;
    case SQL_SUCCESS_WITH_INFO:
      cout << dberror(this,"dbserver::~dbserver SQLDisconnect info").msg();
      break;
    default:
      throw(dberror(this,"dbserver::~dbserver SQLDisconnect"));
  }

  _rc = SQLFreeConnect(_hdbc);                                   6

  if(SQL_SUCCESS != _rc)
    throw(dberror(this,"dbserver::~dbserver SQLFreeConnect"));

  _rc = SQLFreeEnv(_henv);                                       7

  if(SQL_ERROR == _rc)
    throw(dberror(this,"dbserver::~dbserver SQLFreeEnv"));
}

SQLHENV dbserver::get_henv(void) const                           8
{
  return _henv;
}

SQLHDBC dbserver::get_hdbc(void) const                           9
{
  return _hdbc;
}
```

Figure 98 (Part 2 of 2). dbserver.C, dbserver Class Implementation

The constructor allocates resources (1, 2) needed for the DB2/6000 connection to work (3). In the destructor, the transaction is completed (4), the connection is closed (5) and the resources are released (6, 7). Member functions (8, 9) provide *Read Only* access to protected *dbserver* variables for clients that need to know them (for example, `SQLError()` function that is used in the *dberror* class).

8.4.3 Class dbstmt

Objects of class the *dbstmt* issue SQL requests and return results in C++ format. Figure 99 shows the declaration for this class.

```
#ifndef dbstmt_h
#define dbstmt_h

extern "C"                                                          1
{
#include "sqlcli.h"
}

class dbserver;                                                     2
class dberror;                                                      3

class dbstmt
{
  protected:
    typedef enum
    {
      NAME_LEN      =  20,
      STATE_LEN     =   6
    } _info;

    SQLHSTMT     _hstmt;     // SQL statement handle
    dbserver*    _dbserver;  // pointer to dbserver object
    SQLRETURN    _rc;        // for return code
```

Figure 99 (Part 1 of 3). dbstmt.h, dbstmt Class Declaration

```
public:

  dbstmt(dbserver *ds) throw(dberror);

  virtual ~dbstmt(void) throw(dberror);

  void SQLAllocStmt(void) throw(dberror);
  void SQLFreeStmt(SQLSMALLINT option=SQL_DROP) throw(dberror);

  void SQLPrepare(SQLCHAR* sql_string) throw(dberror);

  void SQLSetParam(SQLSMALLINT       par_no,
                   SQLSMALLINT       par_type,
                   SQLSMALLINT       col_type,
                   SQLINTEGER        col_length,
                   SQLSMALLINT       par_scale,
                   SQLPOINTER        data_ptr,
                   SQLINTEGER*       data_size) throw(dberror);

  void SQLExecute(void) throw(dberror);

  void SQLDescribeCol(SQLSMALLINT   col_no,
                      SQLCHAR*      col_name,
                      SQLSMALLINT   col_name_buffer_size,
                      SQLSMALLINT*  col_name_actual_size,
                      SQLSMALLINT*  col_type,
                      SQLINTEGER*   col_length,
                      SQLSMALLINT*  col_scale,
                      SQLSMALLINT*  col_nullable) throw(dberror);

  void SQLNumResultCols(SQLSMALLINT* num_col);

  void SQLBindCol(SQLSMALLINT       col_no,
                  SQLSMALLINT       col_type,
                  SQLPOINTER        col_data_buffer,
                  SQLINTEGER        col_data_buffer_size,
                  SQLINTEGER*       col_data_actual_size) throw(dberror);

  void SQLFetch(void) throw(dberror);
```

Figure 99 (Part 2 of 3). dbstmt.h, dbstmt Class Declaration

```
    void SQLGetData(SQLSMALLINT       col_no,
                    SQLSMALLINT       col_type,
                    SQLPOINTER        col_data_buffer,
                    SQLINTEGER        col_data_buffer_size,
                    SQLINTEGER*       col_data_actual_size) throw(dberror);

    void SQLRowCount(SQLINTEGER*      no_rows) throw(dberror);

};

#endif
```

Figure 99 (Part 3 of 3). dbstmt.h, dbstmt Class Declaration

We need to include the DB2 CLI include file (**1**), as well as forward declarations for the *dbserver* (**2**) and *dberror* (**3**) classes to be able to use them in the declaration.

Figure 100 shows the implementation for the *dbstmt* class. Due to the size, only a couple of selected functions are included here—enough to illustrate wrapping of DB2 CLI functions.

```
#include <stdio.h>
#include <string.h>

#include "dbstmt.h"
#include "dbserver.h"                                                  1
#include "dberror.h"                                                   2

dbstmt::dbstmt(dbserver* ds) throw(dberror) : _dbserver(ds)
{
  SQLAllocStmt();
}

dbstmt::~dbstmt(void) throw(dberror)
{
  SQLFreeStmt();
}
```

Figure 100 (Part 1 of 2). dbstmt.C, Selected Parts of dbstmt Class Implementation

```
void dbstmt::SQLAllocStmt(void) throw(dberror)
{
  _rc = ::SQLAllocStmt(_dbserver->get_hdbc(),&_hstmt);         3

  switch(_rc)
  {
    case SQL_SUCCESS:
      break;
    case SQL_SUCCESS_WITH_INFO:
      cout << dberror(_dbserver,"dbstmt::SQLAllocStmt info").msg();
      break;
    default:
      throw(dberror(_dbserver,"dbstmt::SQLAllocStmt"));
  }
}
:
:
void dbstmt::SQLBindCol(SQLSMALLINT       col_no,
                        SQLSMALLINT       col_type,
                        SQLPOINTER        col_data_buffer,
                        SQLINTEGER        col_data_buffer_size,
                        SQLINTEGER*       col_data_actual_size)
                        throw(dberror)
{
  _rc = ::SQLBindCol(_hstmt,col_no,col_type,col_data_buffer,    4
                            col_data_buffer_size,col_data_actual_size);

  switch(_rc)
  {
    case SQL_SUCCESS:
      break;
    case SQL_SUCCESS_WITH_INFO:
      cout << dberror(_dbserver,"dbstmt::SQLBindCol info").msg();
      break;
    default:
      throw(dberror(_dbserver,"dbstmt::SQLBindCol"));
  }
}
:
```

Figure 100 (Part 2 of 2). dbstmt.C, Selected Parts of dbstmt Class Implementation

Include statements (**1**, **2**) are needed because if we have forward declarations for classes *dbserver* and *dberror* in the `dbstmt.h` file, then full declarations are needed so objects of these classes can be accessed (**3**) and constructed (`throw(dberror)` statements). A call to `SQLAllocStmt()`(**3**) is preceded with a double colon (`::`) to ensure that the global DB2 CLI function will be called, rather then this member function where the call is made. This would result in an infinite recursion, causing the application to fail. Since C^{++} differentiates functions not only by name, but also by number and type of their parameter lists, in this particular case, the infinite recursion problem would not have occurred. However, such a practice does not pose any overhead for the application and makes the code more readable.

The reason *dbstmt* member functions do not need the SQL statement handle (*_hstmt*) to be passed as a parameter is that the SQL statement handle (*_hstmt*) is already part of the class and can simply be accessed by member functions that need access to it, as shown in line **4**.

8.4.4 Class dberror

Objects of the class *dberror* are created when an exception is triggered. The declaration is shown in Figure 101.

```
#ifndef dberror_h
#define dberror_h

extern "C"
{
#include "sqlcli.h"
}

class dbserver;

class dberror
{
  protected:
    typedef enum
    {
      MSG_ID_LEN    =  30,
      STATE_LEN     =  10
    } _info;

    dbserver*    _dbserver;
    SQLRETURN    _rc;                                            1

    SQLCHAR      _sqlstate[STATE_LEN];                           2
    char*        _errmsg[SQL_MAX_MESSAGE_LENGTH+MSG_ID_LEN];     3
    SQLINTEGER   _sqlcode;                                       4
    SQLSMALLINT  _errmsglen;


  public:

    dberror(dbserver* dbserver,const char* init_text);

    virtual ~dberror(void);

    const char*     msg(void);

};

#endif
```

Figure 101. dberror.h, dberror Class Declaration

The objects of the class *dberror* contain the following information returned by DB2/6000:

- Return code (**1**)
- SQL state code (**2**)

- Error message text (**3**)
- SQL code (**4**).

Figure 102 on page 166 contains the `dberror.C` file, where the *dberror* class is implemented.

```
#include <stdio.h>
#include <iostream.h>
#include <iomanip.h>
#include <string.h>

#include "dberror.h"
#include "dbserver.h"

dberror::dberror(dbserver* dbserver,const char* init_text) :
                 _dbserver(dbserver)
{
  static SQLCHAR  errmsg[SQL_MAX_MESSAGE_LENGTH+20];
  static SQLCHAR  tmpmsg[SQL_MAX_MESSAGE_LENGTH+20];
  SQLSMALLINT     errmsglen;
  SQLRETURN ret;

  sprintf((char *)_errmsg,"[%s]",init_text);                    1

  ret = SQLError(_dbserver->get_henv(),                         2
                 _dbserver->get_hdbc(),
                  SQL_NULL_HSTMT,
                 _sqlstate,&_sqlcode,errmsg,
                  SQL_MAX_MESSAGE_LENGTH-1,
                 &errmsglen);

  switch(ret)
  {
    case SQL_NO_DATA_FOUND:                                     3
      sprintf((char*)tmpmsg," SQLError - No error found\n");
      break;
    case SQL_SUCCESS:                                           4
    case SQL_SUCCESS_WITH_INFO:
      sprintf((char*)tmpmsg,"\nSQL state: %s SQL code: %d\n%s",_sqlstate,
                                                               _sqlcode,
                                                                errmsg);
      break;
    default:                                                    5

      sprintf((char*)tmpmsg,"SQLError failed with rc= %d\n",ret);
  }
  strcat((char*)_errmsg,(char*)tmpmsg);                         6
}

dberror::~dberror(void)
{
}

const char* dberror::msg(void)                                  7
{
  return (char*)_errmsg;
}
```

Figure 102. dberror.h, dberror Class Declaration

In the constructor, the message is constructed by concatenating (6) the initial text and (1) with the error codes and the text (3 , 4 , or 5) returned from the `SQLError()` (2) DB2 CLI function.

The initial text (1) is passed as a parameter when *dberror* is created (when an exception is thrown), to indicate in which module the exception has occurred. `SQLError()` is a function that returns additional information about the error that has occurred in one of the DB2 CLI functions. Being a DB2 CLI function itself, it can fail or succeed, just like any other DB2 CLI function; so to ensure validity of returned values its return code must be tested, and an appropriate message must be created (3 , 4 , or 5). `msg(void)` (7) is a public member function that will return the message.

8.5 Implementation Consideration and Organization

This section describes the various conventions we used for the names of directories, files, and functions.

8.5.1 Conventions and Files

We decided to put all the source files related to the C^{++} activity into a separate directory as we did for the *PortedGUI* and *ImprovedGUI* applications. We created a new directory called `C` in the `/ad/projectA_proto/source` directory for this purpose. (C^{++} source files have the extension of `.C`.)

Beneath `/ad/projectA_proto/source/C` we created a directory to hold the files related to the GUI that we would design and generate with AIC 1.2. This was named `OOGUI`. Beneath `OOGUI` we created one subdirectory for each interface. Following our design, we named the interfaces and the sudirectories that would contain their files:

CustomersGUI	The primary window—the Customers window. It showed search fields for Customer Name and Customer Number, a Search push button, and a list of Customers that matched the search criteria.
CustomerGUI	A secondary window—the Customer Details window. It showed all the fields for one customer.
PaymentsGUI	A primary window—the Payments window. It showed a list of Payments made by a particular customer.
PaymentGUI	A secondary window—the Payment Details window. It showed the detail fields for a particular payment.

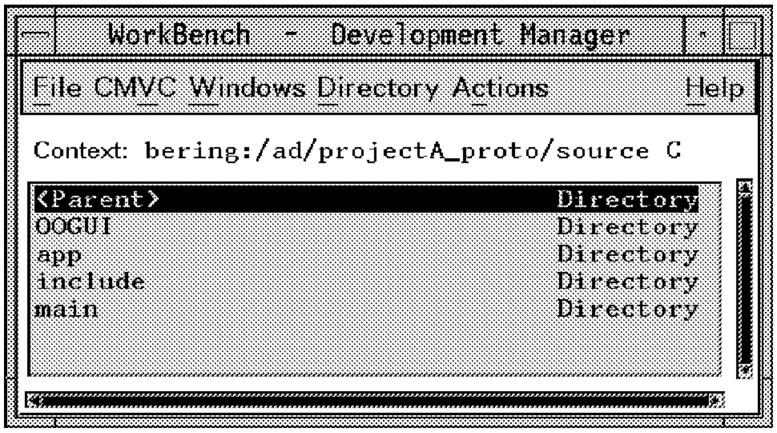
In the *PortedGUI* and the *ImprovedGUI* implementations, shown in 6.2.3, “Implementing the Callbacks” on page 80 and 7.3, “User Interface Implementation” on page 139, we implemented the callback code in a separate C source file so we could use Program Editor, rather than the AIC editor, to create and maintain the callback source code. Now, however, we wanted to generate C^{++} source code from this interface. We moved the callbacks back into the interface file, so they could be generated as member functions of the class that implemented this interface. Fortunately, the improved integration of AIC 1.2 with SDE WorkBench/6000 now enabled us to use Program Editor with AIC. This meant there would be no callback source files in the OOGUI subdirectories.

We had AIC generate a header file for each interface and place them in the interface directories under OOGUI.

Beneath /ad/projectA_proto/source/C we created an include to contain the links to all the include files that we created in any directory containing C++ source files.

Beneath /ad/projectA_proto/source/C we created a main directory in which to store the AIC project file, the generated C main program, and the associated makefile.

Finally, beneath /ad/projectA_proto/source/C we created an app directory to hold the source code files defining the application classes. Figure 103 shows the Development Manager window after creating these directories in /ad/projectA_proto/source/C/OOGUI.


WorkBench - Development Manager
File CMVC Windows Directory Actions Help

```
Context: bering:/ad/projectA_proto/source C

<Parent>                    Directory
OOGUI                       Directory
app                         Directory
include                     Directory
main                        Directory
```

Figure 103. The Subdirectories for the C++ Development Project

We should have gone on to create additional directories for the source implementing the application classes, and the component classes for database access. However, we were running out of time. Knowing that we would not be able to implement the entire application in C++, we redirected our goals to include proving the basic concepts of the design and exploring the GUI and application code integration issues. Under these circumstances, we decided to create the remaining C++ source files in the app directory.

Had we completed entire application, the following subdirectories containing these files would have existed:

app	The C++ source files, described in 8.7, "Implementation of C++ Application Code" on page 174, implementing the application classes and integrating the GUI with them:

- Customers.C
- Customers.h
- Payments.C.

- `Payments.h`
- `Customer.C`
- `Customer.h`
- `Payment.C.`
- `Payment.h`
- `aux.C`
- `aux.h`

app_logic

The C^{++} source files, described in 8.7, “Implementation of C^{++} Application Code” on page 174, implementing the application logic classes:

- `CustomersApp.C`
- `CustomersApp.h`
- `PaymentsApp.C`
- `PaymentsApp.h`
- `CustomerApp.C`
- `CustomerApp.h`
- `PaymentApp.C`
- `PaymentApp.h`

component

The C^{++} source files, described in 8.4, “Our DB2/6000 Object-Oriented Interface” on page 155, implementing the classes that provided database access:

- `dbserver.C`
- `dbserver.h`
- `dberror.C`
- `dberror.h`
- `dbstmt.C`
- `dbstmt.h`

8.6 Implementation of the GUI Using AIC 1.2

The following sections describes the effort to implement the design of the windows illustrated in Figure 90 on page 137 and Figure 91 on page 138 with C^{++}. It shows how we used AIC 1.2 to generate the GUI in C++.

8.6.1 Preparing AIC for C^{++} Code Generation

We started SDE WorkBench on the *yellow* host, because AIC 1.2 was installed and integrated with SDE WorkBench/6000 on that host. We then started AIC 1.2 using SDE WorkBench/6000.

To set up our AIC environment for C^{++} development, we selected **Code Generation ...** from the Options pull-down in the AIC main window. In the resulting dialog box, we selected **C^{++}** as the programming language, entered *C* for the C file suffix, and chose to not use the Ux convenience library. We pressed the **Apply** push button to activate these changes. We then selected **Save Options...** from the File pull-down of the AIC main window to save these settings for future invocations of AIC. For more information on code generation, refer to *AIXwindows Interface Composer Developer's Guide, Version 1.2*.

We had to make sure that the AIC interpreter used the path `/nfs/bering/ad/projectA_proto/source/C/include` when it looked for include files. We therefore added an entry to the AIC resource file that added this path name to the list of values associated with the variable *Aic12.cflags*. Modifying the AIC resource file is described in 9.5, "Tailoring AIC" on page 221 and illustrated in Figure 147 on page 222

8.6.2 Copying the OOGUI Interfaces from the Improved GUI Interfaces

For the *ImprovedGUI*, we had designed four interfaces:

- Customers
- CustomerDetails
- Payments
- PaymentDetails.

Because the callback and interface source files for these were stored in directories of the same names, we did not have to create a new AIC project and design four new interfaces to look exactly like those we had already designed for *ImprovedGUI*. Instead we reused these existing interfaces.

First, we loaded the existing interface for the Customers window of the *ImprovedGUI* into AIC. To do this, we selected **Open...** from the File pull-down. In the resulting dialog box, we entered the complete path name of this interface file: `/nfs/bering/ad/projectA_proto/source/c/ImprovedGUI/Customers/Customers.i`. To complete the action, we pressed the **OK** push button. We repeated this process, loading each of the remaining three interface files of the *ImprovedGUI* into AIC 1.2.

Each interface file that we loaded contained the path name used by AIC when saving the interface file. The interface file also contained the path name of the directory in which the generated code files should be placed. These directories were associated with the *ImprovedGUI* application. We, therefore, had to change these path names to point to correct directories, such as `nfs/bering/ad/projectA_proto/source/C/OOGUI/CustomersGUI`.

To do this, we performed the following steps for each of the interfaces:

1. Highlight the interface icon in the AIC Interfaces area.
2. Select **Save Interface As...** from the resulting pop-up menu.
3. Enter the new path name in the resulting dialog box.
4. Press the **OK** push button.
5. Press the right mouse button, again.
6. Select **Generate Interface Code As..** from the resulting pop-up menu.
7. Enter the new path name in the resulting dialog box.
8. Press the **OK** push button.

For more information about working with interfaces, refer to *AIXWindows Interface Composer Developer's Guide, Version 1.2*.

We next set the X Windows application class name of our project. The X Windows *application class* should not be confused with a C^{++} application class. In X Windows terminology, this is the name for a string that can be used to identify resource values of widgets in the GUI of a particular application. This string is prefixed to the X resource name when the resource's value is set in the `.Xdefaults` file. This string can also be used to name an application-specific resource file in which the resource values are set.

To do this, we selected **Program Layout...** from the Edit pull-down of the AIC main window and entered *OOGUI* in the Application Class text field. We then pressed the **Apply** push button.

We set the name of the AIC project to *OOGUI* by selecting **Save Project As...** from the File pull-down of the AIC main window. In the resulting dialog box we entered the name `OOGUI.prj`. We pressed the **OK** push button to complete this action.

The resulting AIC windows are shown in Figure 104. The name of the project *OOGUI* is reflected in the dialog title of the AIC main project window, which is the lowest window in the figure. The default widget palette is being composed of all OSF/Motif widgets as shown in the window at the right of the figure. The four windows Customers, Payments, Customer Details, and Payment Details, as shown in this figure, as drawn from the information in the interface files loaded into AIC. They look just as they did in the *ImprovedGUI* version.

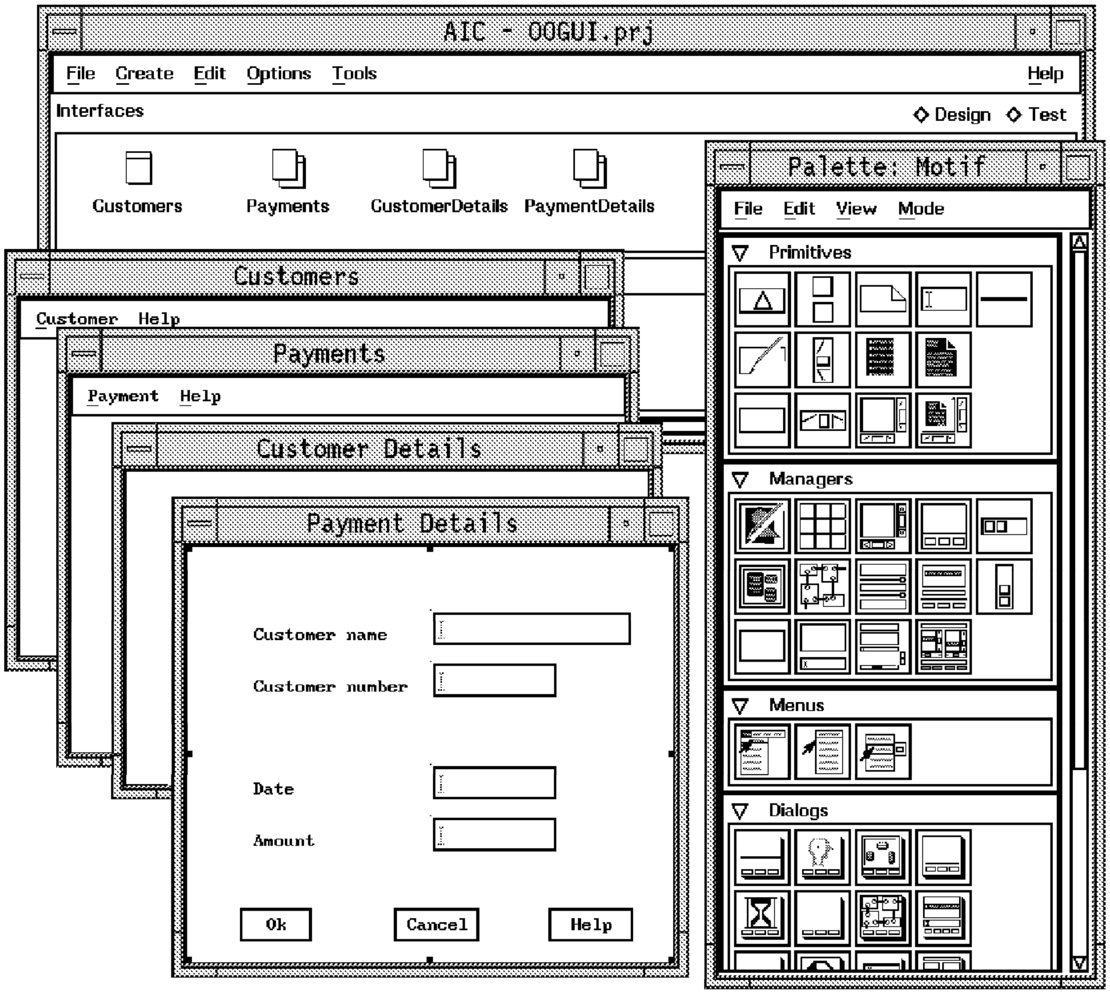


Figure 104. AIC 1.2 with the OOGUI Project

8.6.3 Modifying the Callbacks

We now had to implement the callbacks of the four interfaces. As is shown in Figure 89 on page 135, the callbacks for the *ImprovedGUI* version of our application were implemented as C functions in external source files. Only a call to each function was entered in the interface directly. Entering this call required our using the AIC widget Property Editor. To write the actual function in C, we edited the separate source file using Program Editor. This was illustrated in Figure 89 on page 135.

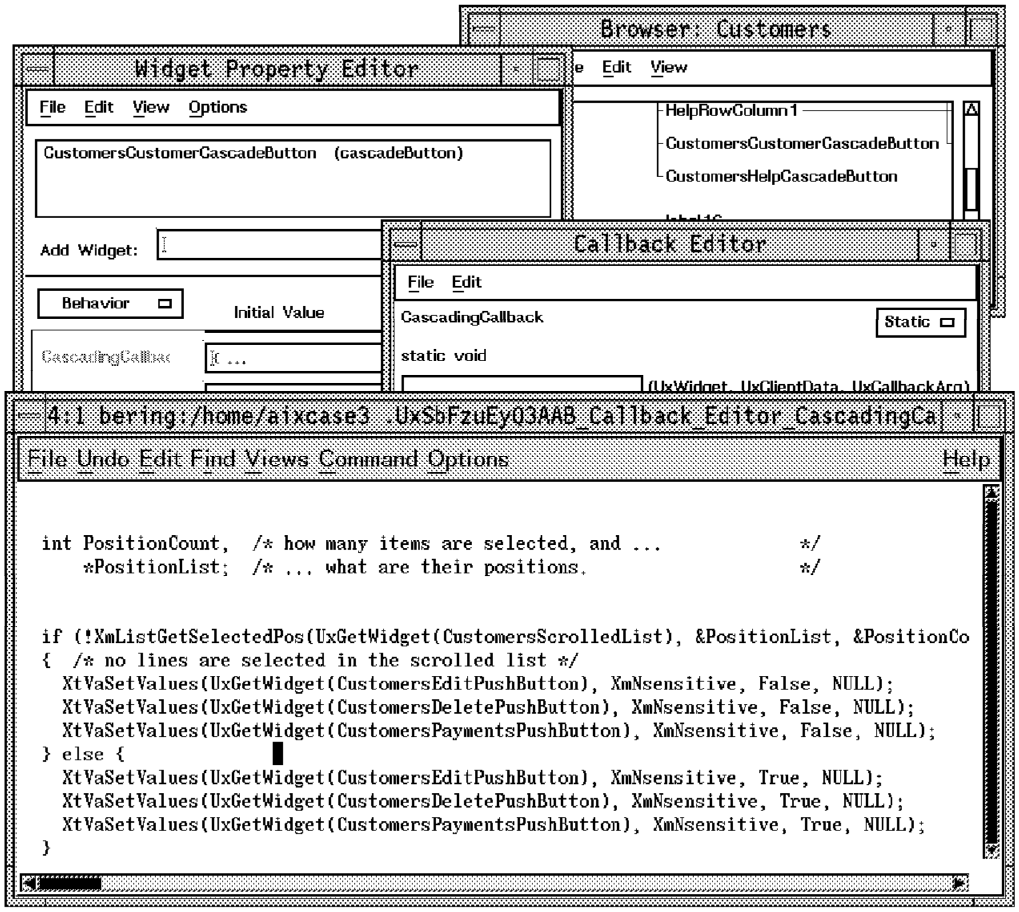
We could reuse these C functions with the *OOGUI* version of our application if we moved the source code back into the interface. AIC 1.2 would generate C++ wrapper code for these C language functions, turning them into private C++ function members of the GUI classes.

We reused the C source code for the callback functions in the Customers window by using these steps (see Figure 105 on page 173):

1. Highlight the Customers icon in the Interfaces area of the AIC main window with the left mouse button.
2. Select **Browser** from the Edit pull-down.
3. Select the *CustomerCustomersCascadeButton* widget in the Browser window.
4. Select **Property Editor...** from the Edit pull-down of the Browser window.
5. Change the toggle button in the Widget Property Editor window to *Behavior* rather than *Core*.
6. The Widget Property Editor displays the list of callbacks along with the corresponding code. Scroll to the entry for the *Cascading Callback*, and press the **...** button right to the text field of that callback entry.
7. After the Callback Editor window is displayed, select **Modify Code** from the Edit pull-down to invoke Program Editor.
8. Program Editor displays a source that line looks like this:

```
CustomerCustomersCascadingCallback(UxWidget);
```

9. Set the context of Development Manager to the source directory of the code that was implemented in the separate `CustomersCallbacks.c` source file for the *ImprovedGUI*.
10. Select **Edit** from the Actions pull-down. A second instance of the Program Editor window is displayed containing this source file.
11. Replace this call in the one file with the code from the second file, using the Program Editor's cut-and-paste mechanism.
12. Close the second editor session by selecting **Close** from the File pull-down. Figure 105 on page 173 shows the AIC and Program Editor windows that are used at this point.
13. Save the changes to the first file and close Program Editor.


Browser: Customers
Edit View
HelpRowColumn1
CustomersCustomerCascadeButton
CustomersHelpCascadeButton
Widget Property Editor
File Edit View Options
CustomersCustomerCascadeButton (cascadeButton)
Add Widget:
Behavior
Initial Value
CascadingCallbac
Callback Editor
File Edit
CascadingCallback
Static
static void
(UxWidget, UxClientData, UxCallbackArg)
4:1 bering:/home/aixcase3 .UxSbFzuEyO3AAB_Callback_Editor_CascadingCa
File Undo Edit Find Views Command Options Help

```
int PositionCount,  /* how many items are selected, and ...            */
    *PositionList;  /* ... what are their positions.                   */


if (!XmListGetSelectedPos(UxGetWidget(CustomersScrolledList), &PositionList, &PositionCo
{  /* no lines are selected in the scrolled list */
  XtVaSetValues(UxGetWidget(CustomersEditPushButton), XmNsensitive, False, NULL);
  XtVaSetValues(UxGetWidget(CustomersDeletePushButton), XmNsensitive, False, NULL);
  XtVaSetValues(UxGetWidget(CustomersPaymentsPushButton), XmNsensitive, False, NULL);
} else {
  XtVaSetValues(UxGetWidget(CustomersEditPushButton), XmNsensitive, True, NULL);
  XtVaSetValues(UxGetWidget(CustomersDeletePushButton), XmNsensitive, True, NULL);
  XtVaSetValues(UxGetWidget(CustomersPaymentsPushButton), XmNsensitive, True, NULL);
}
```

Figure 105. Editing Callbacks for the OOGUI Project

We repeated these steps for each of the callbacks in the interfaces. For more details on editing widget properties, refer to *AIXwindows Interface Composer Developer's Guide, Version 1.2*. For more information on specifying callback behavior, refer to *AIXwindows Interface Composer Developer's Guide, Version 1.2*.

We now had an AIC project identifying four interfaces each of which contained its own callback logic. The source code that was generated from these interfaces had no dependency on externally compiled functions. These callbacks did not contain any application logic, however. They contained the functionality necessary to set the sensitivity for push buttons in pull-down menus, and make calls to *UxPopupInterface* and *UxPopdownInterface* in the activate callbacks for the corresponding push buttons. This enabled testing the GUI without related application logic or its access to the DB2/6000 database.

8.6.4 Generating C++ GUI and Building the Executable GUI

To test the GUI stand-alone, we had to generate C++ code, generate a makefile, and run that makefile. We decided to use the AIC code generation utilities for this task.

We selected **Current Directory...** from the Options pull-down of the AIC main project window. In the resulting dialog box we set the current directory of AIC to `/ad/projectA_proto/source/C/main`. We pressed the **OK** push button to complete this action. We did not set the path to start with `/nfs/bering`. This was to circumvent the problems with network file resolution that appeared when we used AIC 1.2 together with SDE WorkBench/6000 as described in 9.4.6, "Integration Restrictions and Their Circumventions" on page 220. For more information about setting the current directory, refer to "Setting Default Options" in *AIXwindows Interface Composer Developer's Guide, Version 1.2*.

We selected **Generate Project As...** from the File pull-down of the AIC main project window. In the resulting dialog box, we entered the project code directory name. We set the options to generate the modified interfaces, the main file, and the makefile. We also set the option to run the makefile. We pressed the **OK** push button to complete this action.

AIC 1.2 started the Program Builder to run the generated makefile. For more information about this, refer to Chapter 21, "Generating Code" in *AIXwindows Interface Composer Developer's Guide, Version 1.2*.

The generated `OOGUI` executable file was written to the `/ad/projectA_proto/source/C/main` directory, and could be executed from there.

8.6.5 Saving the New AIC Interfaces

After we had tested the application code, we saved the project by selecting **Save** from the File pull-down of the AIC main project window. We then quit AIC by selecting **Exit** from the File pull-down.

8.7 Implementation of C++ Application Code

This section shows how the base classes, implementing application logic, and DB2/6000 services were used as building blocks for *Customer* and *Payment* application classes. This section includes sample code that is important for further understanding of the integration process.

8.7.1 Class CustomerApp

Class *CustomerApp* provides the application logic for class *Customer*. This application logic results in the appropriate data for the Customer Details window being updated to or extracted from the application's database. Figure 106 on page 175 shows the contents of `CustomerApp.h` file that contains the declaration of *CustomerApp* class.

```
#ifndef CustomerApp_h
#define CustomerApp_h

#include "dbstmt.h"

class CustomerApp
{
  protected:
    typedef enum
    {
      NUMBER_SIZE     =  8,
      FIRST_NAME_SIZE = 13,
      LAST_NAME_SIZE  = 18,
      STREET_SIZE     = 26,
      ZIP_CODE_SIZE   =  5,
      CITY_SIZE       = 20
    } _info;

    SQLINTEGER    _customer_no;                             1
    SQLINTEGER    _reference_no;
    SQLCHAR       _first_name[FIRST_NAME_SIZE+1];
    SQLCHAR       _last_name[LAST_NAME_SIZE+1];
    SQLCHAR       _street[STREET_SIZE+1];
    SQLINTEGER    _zip_code;
    SQLCHAR       _city[CITY_SIZE+1];
    SQLINTEGER    _address_change;

    SQLINTEGER    _mail_id;
    SQLINTEGER    _source_code;
    SQLINTEGER    _last_activity;

    dbstmt        _cust;                                    2
    dbstmt        _name;
    dbstmt        _zip;

  public:

    CustomerApp(dbserver *dbsrv);
    virtual ~CustomerApp(void);

    void get_by_custno(int custno);                         3
    SQLINTEGER reference_no(void) const;
    friend ostream& operator<<(ostream& x, const CustomerApp& a);

};

#endif
```

Figure 106. CustomerApp.h, CustomerApp Class Declaration

In this declaration, storage is reserved for all data fields related to the Customer Details window (1). This storage is made accessible only to the *CustomerApp* and its derived classes. For each of the database tables we need to access there is a corresponding *dbstmt* variable (2), which provides the database interface.

Public interface consists of three functions (3), in addition to the constructor and the destructor:

`void get_by_custno(int custno)`
: Initiates the retrieval of data about the person, whose customer number is `custno`, from the database.

`SQLINTEGER reference_no(void) const`
: Permits information about the reference number to be available outside the class in Read Only mode.

`friend ostream& operator<<(ostream& x, const CustomerApp& a)`
C++ output stream function for *CustomerApp* class. Strictly speaking this is not a member function of a *CustomerApp* class, but a `friend` operator function. Nevertheless it is logically part of public interface for *CustomerApp* class, and therefore mentioned here.

File `CustomerApp.C` contains implementation of *CustomerApp* class as shown in Figure 107.

```
#include <stdio.h>
#include <iostream.h>
#include <iomanip.h>
#include <string.h>
#include "CustomerApp.h"

CustomerApp::CustomerApp(dbserver *dbsrv):                                    1
                         _cust(dbsrv),_name(dbsrv),_zip(dbsrv)
{
  static SQLINTEGER   actsize;

  _cust.SQLBindCol(1,SQL_C_LONG,&_customer_no,sizeof(_customer_no),&actsize);
  _cust.SQLBindCol(2,SQL_C_LONG,&_reference_no,sizeof(_reference_no),&actsize);
  _cust.SQLBindCol(3,SQL_C_LONG,&_last_activity,sizeof(_last_activity),&actsize);
  _cust.SQLBindCol(4,SQL_C_LONG,&_address_change,sizeof(_address_change),&actsize);
  _cust.SQLBindCol(6,SQL_C_LONG,&_mail_id,sizeof(_mail_id),&actsize);
  _cust.SQLBindCol(7,SQL_C_LONG,&_source_code,sizeof(_source_code),&actsize);

  _name.SQLBindCol(2,SQL_C_DEFAULT,_first_name,sizeof(_first_name),&actsize);
  _name.SQLBindCol(3,SQL_C_DEFAULT,_last_name,sizeof(_last_name),&actsize);
  _name.SQLBindCol(4,SQL_C_DEFAULT,_street,sizeof(_street),&actsize);
  _name.SQLBindCol(5,SQL_C_LONG,&_zip_code,sizeof(_zip_code),&actsize);

  _zip.SQLBindCol(2,SQL_C_DEFAULT,_city,sizeof(_city),&actsize);
}

CustomerApp::~CustomerApp(void)
 2
{
}
```

Figure 107 (Part 1 of 2). CustomerApp.C, CustomerApp Class Implementation

```
void CustomerApp::get_by_custno(int custno)                                           3
{
  static SQLCHAR  tmp[SQL_MAX_MESSAGE_LENGTH+20];

  sprintf((char *)tmp,"select * from name where custno=%d",custno);

  _name.SQLPrepare(tmp);
  _name.SQLExecute();
  _name.SQLFetch();
  _name.SQLFreeStmt(SQL_CLOSE);

  sprintf((char *)tmp,"select * from cust where custno=%d",custno);

  _cust.SQLPrepare(tmp);
  _cust.SQLExecute();
  _cust.SQLFetch();
  _cust.SQLFreeStmt(SQL_CLOSE);

  sprintf((char *)tmp,"select * from zip where zipcode=%d",_zip_code);

  _zip.SQLPrepare(tmp);
  _zip.SQLExecute();
  _zip.SQLFetch();
  _zip.SQLFreeStmt(SQL_CLOSE);
}

SQLINTEGER CustomerApp::reference_no(void) const
{
  return _reference_no;
}

ostream& operator<<(ostream& x, const CustomerApp& a)                                 4
{
  x << setfill('0')
    << setw(CustomerApp::NUMBER_SIZE)     << a._customer_no <<
" "
    << setfill(' ') << setiosflags(ios::left)
    << setw(CustomerApp::FIRST_NAME_SIZE) << (char *)
a._first_name  << " "
    << setw(CustomerApp::LAST_NAME_SIZE)  << (char *)
a._last_name   << " "
    << setw(CustomerApp::STREET_SIZE)     << (char *)
a._street      << " "
    << setfill('0') << setiosflags(ios::right)
    << setw(CustomerApp::ZIP_CODE_SIZE)   << a._zip_code    <<
" "
    << setfill(' ') << setiosflags(ios::left)
    << setw(CustomerApp::CITY_SIZE)       << (char *)
a._city        << " "
    << a._address_change << " ";
  return x;
}
```

Figure 107 (Part 2 of 2). CustomerApp.C, CustomerApp Class Implementation

In the constructor (**1**), instances of the *dbstmt* class are initialized with a pointer to the instance of the *dbserver* class that was created earlier. The constructor also uses the `SQLBindCol()` member function in three instances of the `dbstmt` class to bind appropriate data members of the *CustomerApp* class to specific data fields in the customer, name, and zip code tables. This is necessary to transfer data from the database to the class data members.

If some resources, like database locks, were obtained during the lifetime of the object, they would typically be released in the destructor (**2**). Since this was not the case in our example, the destructor simply does nothing.

Function `void CustomerApp::get_by_custno(int custno)` (**3**) issues SQL requests to retrieve data from three different tables (name, cust, and zip) and stores the results in C++ variables.

The operator << (**4**) produces a line of output. Typically it would be used for debugging or producing tables.

8.7.2 CustomerGUI

The *CustomerGUI* class was generated by AIC. Since it is one of the direct base classes for *Customer* class, we needed to examine the generated source code and determine how we would integrate it with the application class we would derive from it. Figure 108 contains the `CustomerGUI.h` file—the declaration of *CustomerGUI* class.

```
#ifndef     _CUSTOMERGUI_INCLUDED
#define     _CUSTOMERGUI_INCLUDED

#include <stdio.h>
#include <Xm/Xm.h>
#include <Xm/MwmUtil.h>
#include <Xm/MenuShell.h>
#include "UxXt.h"

#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/TextF.h>
#include <Xm/Label.h>
#include <Xm/Form.h>
#include <X11/Shell.h>
```

Figure 108 (Part 1 of 2). CustomerGUI.h, CustomerGUI Class Declaration

```
class _UxCCustomerGUI: public _UxCInterface                      1
{
public:

     _UxCCustomerGUI();

     Widget     _create_CustomerGUI( void );

protected:

     Widget     CustomerGUI;                                      2
     Widget     form1;
     Widget     label1;
     Widget     label2;
     Widget     label3;
     Widget     CustomerDetailsFirstName;
     Widget     CustomerDetailsLastName;
     Widget     CustomerDetailsStreet;
     Widget     CustomerDetailsZIPCode;
     Widget     CustomerDetailsCustomerNo;
     Widget     CustomerDetailsMailId;
     Widget     label14;
     Widget     CustomerDetailsCity;
     Widget     label15;
     Widget     CustomerDetailsReferenceNo;
     Widget     label18;
     Widget     rowColumn1;
     Widget     CustomerDetailsOkPushButton;
     Widget     CustomerDetailsCancelPushButton;
     Widget     CustomerDetailsHelpPushButton;
     Widget     CustomerDetailsSourceCode;
     Widget     label20;
     Widget     CustomerLastActivity;
     Widget     label21;
     Widget     CustomerLastAddressChange;
     Widget     label4;
     Widget     label5;
     Widget     label12;
                                                                  3
     void      activateCB_CustomerDetailsOkPushButton(Widget, XtPointer, XtPointer);
                                                                  4
     static void Wrap_activateCB_CustomerDetailsOkPushButton(Widget,
                                                             XtPointer,
XtPointer);
     void      activateCB_CustomerDetailsCancelPushButton(Widget,
                                                          XtPointer,
XtPointer);
     static void Wrap_activateCB_CustomerDetailsCancelPushButton(Widget,

XtPointer, XtPointer);

     static void      UxDestroyContextCB(Widget, XtPointer, XtPointer);

private:
     Widget _build();
     CPLUS_ADAPT_CONTEXT(_UxCCustomerGUI)
} ;

Widget     create_CustomerGUI( void );

#endif
```

Figure 108 (Part 2 of 2). CustomerGUI.h, CustomerGUI Class Declaration

The first thing we noted was that the actual name for the *CustomerGUI* class is *_UxCCustomerGUI* (1). The prefix *_UxC* is automatically attached by AIC. It is important to use the actual name in C++ code, but for conceptual discussion it is less confusing to use the name we have used so far—*CustomerGUI* and we will continue to do so.

Another thing worth noting is the naming convention for members of the *CustomerGUI*—it is just the opposite of the naming convention we used for the *CustomerApp* class. In this code, names of public members start with an underscore (_) (so does the actual class name), while names of protected and private members start with an alphanumeric character. This AIC convention contradicts the convention widely encountered in the authors' experience. We followed the more common C++ naming convention, using the leading underscore to indicate a private and protected label in all the code we wrote ourselves.

All GUI elements, the X widgets, are declared (**2**), and so are the callback functions (**3** and **4**).
Notice that `Wrap_activateCBCustomerDetailsOkPushButton()` (**4**) is declared as `static`, while `activateCB_CustomerDetailsOkPushButton()` (**3**) is not. The significance of this is made clear when we examine the selected details of implementation of the *CustomerGUI* class, shown in Figure 109.

```
#include <stdio.h>
#include <Xm/Xm.h>
#include <Xm/MwmUtil.h>
#include <Xm/MenuShell.h>
#include "UxXt.h"

#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/TextF.h>
#include <Xm/Label.h>
#include <Xm/Form.h>
#include <X11/Shell.h>

#include "CustomerGUI.h"

void _UxCCustomerGUI::activateCB_CustomerDetailsOkPushButton(      1
                         Widget wgt,
                         XtPointer cd,
                         XtPointer cb)
{
     Widget                 UxWidget = wgt;
     XtPointer              UxClientData = cd;
     XtPointer              UxCallbackArg = cb;
     {
     UxPopdownInterface(UxThisWidget);
     }
}
```

Figure 109 (Part 1 of 3). CustomerGUI.C, Selected Parts of CustomerGUI Class Implementation

```
void _UxCCustomerGUI::Wrap_activateCB_CustomerDetailsOkPushButton( 2
                         Widget wgt,
                         XtPointer cd,
                         XtPointer cb)
{
     _UxCCustomerGUI         *UxContext;
     Widget                  UxWidget = wgt;
     XtPointer               UxClientData = cd;
     XtPointer               UxCallbackArg = cb;
     UxContext = (_UxCCustomerGUI *) UxGetContext(UxWidget);      3
     UxContext->activateCB_CustomerDetailsOkPushButton(UxWidget,   4
                                                        UxClientData,
                                                        UxCallbackArg);
}

⋮

Widget _UxCCustomerGUI::_build()
{
     // Creation of CustomerGUI
     CustomerGUI = XtVaCreatePopupShell( "CustomerGUI",
               topLevelShellWidgetClass,
               UxTopLevel,
               XmNx, 718,
               XmNy, 120,
               XmNwidth, 470,
               XmNheight, 590,
               XmNiconName, "Customer ",
               XmNtitle, "Customer",
               NULL );
     UxPutContext( CustomerGUI, (char *) this );

⋮
     // Creation of CustomerGUIOkPushButton
     CustomerDetailsOkPushButton =
     XtVaCreateManagedWidget( "CustomerDetailsOkPushButton",
                               xmPushButtonWidgetClass,
                               rowColumn1,
                               XmNx, -40,
                               XmNy, 0,
                               XmNwidth, 76,
                               XmNheight, 47,
                               RES_CONVERT( XmNlabelString, "   Ok   " ),
                               XmNfontList, UxConvertFontList( "rom10" ),
                               XmNnavigationType, XmTAB_GROUP,
                               NULL );

     XtAddCallback( CustomerDetailsOkPushButton, XmNactivateCallback,
     (XtCallbackProc) &_UxCCustomerGUI::Wrap_activateCB_CustomerDetailsOkPushButton,
     (XtPointer) NULL );

     UxPutContext( CustomerDetailsOkPushButton,(char *) this);     5

⋮
     return ( CustomerGUI );
}
```

Figure 109 (Part 2 of 3). CustomerGUI.C, Selected Parts of CustomerGUI Class Implementation

```
swidget _UxCCustomerGUI::_create_CustomerGUI(void)
{
     Widget                  rtrn;
     UxThis = rtrn = _build();

     // Final Code from declarations editor
     return(rtrn);
}

_UxCCustomerGUI::_UxCCustomerGUI(void)
{
}

swidget create_CustomerGUI(void)
{
     _UxCCustomerGUI *theInterface =
               new _UxCCustomerGUI();
     return (theInterface->_create_CustomerGUI());
}
```

Figure 109 (Part 3 of 3). CustomerGUI.C, Selected Parts of CustomerGUI Class Implementation

Of particular interest are the two class member functions generated for every callback function. One function, whose name begins with `Wrap_`, is used to register the function (**2**) as a callback function for the widget. Because of how X Windows is implemented, a registered function must have external linkage. This means its address must be resolvable at link-edit time. However, C++ member functions do not have external linkage, unless they are declared as `static`, as this one was.

However, being static, this function does not have access to any class data members that are created dynamically at execution. To circumvent the problem of access, *this* pointer (a class internal pointer pointing to itself) is stored in the X context when the callback is registered (**5**) and retrieved in the callback function (**3**). This `static` wrap function also cannot be inherited by nor overridden in derived classes. Introduction of the second function (**1**) that is called from the static callback function through *this* pointer (**4**) should solve this problem because this second function is a “proper” member function. It is not declared as `static`. This member function can be inherited by a derived class. However, because AIC fails to explicitly declare it `virtual`, it cannot be overridden in the derived class.

This fact seemed to undermine one of our fundamental design decisions. Our design implied the GUI and application logic classes for a given interface are both free of code which “had knowledge” of the other. The derived class, having access to member data and functions in both, should be the only place where knowledge of their interrelationships resides. If a callback in a GUI class needs to display information retrieved from the database by a function in the application logic class, it would be best if that callback function were overridden in the application class with a function that accesses both the application logic class member function and performs the callback actions using GUI class resources.

We researched this problem and postulated three alternatives that would allow us to continue with our basic design:

1. Embed calls to functions of the application logic classes directly in the callback functions of the GUI classes when they are needed. It is wise to group these calls together in a single subroutine and make the one call in the callback. Then define that subroutine in a separately compiled module, an auxiliary

module. When the application is built, the auxiliary module would be compiled and linked with it. During test and debug of the GUI code, a dummy version of this module could be substituted for the actual one, which does only what is necessary to test the GUI.

This approach does not undermine the basic design principles. There is still value in having the derived classes, of course, because routines that are unique to the application can be placed in these classes. Routines like `Customer::fill_GUI_fields()` illustrate this point. But, where a callback must retrieve and then display data from the database, that code which would have been in the derived class is simply placed in subroutines that are part of this external module.

2. Create AIC methods in the GUI class. AIC methods are a new feature of AIC 1.2. AIC methods are generated as virtual member functions in the class representing the interface. As such, AIC methods are polymorhpic—they can be overridden in derived classes.

   While they are not specifically associated with any callback function, AIC methods could be written to contain the code necessary to debug the GUI stand-alone. They would also be overridden in the derived application classes. The AIC methods would have to be called from the second member function associated with the callback (not the function whose name begins with `Wrap_`). AIC methods are described in *AIXwindows Interface Composer Developer's Guide, Version 1.2*.

3. Modify the code after generation by AIC to insert the word `virtual` in front of the appropriate callback function. We decided that this idea had merit, if it could be done algorithmically, without manual intervention during the build process, and if we could confirm that AIC would in later releases correct this problem, which we thought of as an oversight. Subsequent discussions with Visual Edge who produce UIM/X on which AIC is based, indicated that this choice had merit, but did not produce a certain resolution. AIC 1.2 is based on UIM/X 2.5. After this project was completed, but before a later release of AIC came on the market, Visual Edge released UIM/X 2.6. Visual Edge advised the authors that when UIM/X 2.6 generates C^{++} the second member function associated with each callback was declared `virtual`. However, they also indicated that users could only add to the callback lists, not override callbacks. Until a new AIC release picks up this change, we cannot evaluate it firsthand.

8.7.3 Customer

Our project did not have the time to evaluate all three options, though we determined that the second and third were probably the most promising in the long run. However, we pursued the first option with the goal to confirm a minimal integration of the C^{++} code generated by AIC with the C^{++} code that we wrote. We illustrate this approach in our implementation of the *Customer* class and **Search**: push button callback of the *Customers* class.

The *Customer* class is derived from the *CustomerGUI* class that provides it with knowledge about how to interact with users, and from *CustomerApp* that provides it with knowledge about how to interact with the DB2/6000. This class demonstrates the power of inheritance in integrating those qualities together in a single object.

The declaration for *Customer* class is shown in Figure 110 on page 184.

```
#ifndef Customer_h
#define Customer_h

#include "UxXt.h"
#include "CustomerApp.h"
#include "CustomerGUI.h"
class dbserver;

class Customer : public CustomerApp, public _UxCCustomerGUI
{
  public:

    Customer(dbserver* dserv);

    virtual ~Customer(void);

    swidget get_topLevelswidget(void) const;
    void fill_GUI_fields(void);
    friend ostream& operator<<(ostream& x, const Customer& a);
};

#endif
```

Figure 110. Customer.h, Customer Class Declaration

Implementation of the *Customer* class is shown in Figure 111.

```
#include <iostream.h>
#include <iomanip.h>
#include <string.h>
#include "Customer.h"
#include "dbserver.h"

Customer::Customer(dbserver* dserv):                            1
                   CustomerApp(dserv),
                   _UxCCustomerGUI()
{
  _create_CustomerGUI();
}

Customer::~Customer()
{
}

swidget Customer::get_topLevelswidget(void) const
{
  return UxThis;
}

ostream& operator<<(ostream& x, const Customer& a)              2
{
  x << (CustomerApp ) a;
  return x;
}
```

Figure 111 (Part 1 of 2). Customer.C: Customer Class Implementation

```
void Customer::fill_GUI_fields(void)                         3
{
  XmTextFieldSetString(CustomerDetailsFirstName,(char *)_first_name);
  XmTextFieldSetString(CustomerDetailsLastName,(char *)_last_name);
  XmTextFieldSetString(CustomerDetailsStreet,(char *)_street);

  char tmp[NUMBER_SIZE+1];
  sprintf(tmp,"%d",_zip_code);

  XmTextFieldSetString(CustomerDetailsZIPCode,tmp);

  sprintf(tmp,"%d",_customer_no);
  XmTextFieldSetString(CustomerDetailsCustomerNo,tmp);

  sprintf(tmp,"%d",_reference_no);
  XmTextFieldSetString(CustomerDetailsReferenceNo,tmp);

  XmTextFieldSetString(CustomerDetailsCity,(char *)_city);

  sprintf(tmp,"%d",_address_change);
  XmTextFieldSetString(CustomerLastAddressChange,tmp);

  sprintf(tmp,"%d",_last_activity);
  XmTextFieldSetString(CustomerLastActivity,tmp);

  sprintf(tmp,"%d",_mail_id);
  XmTextFieldSetString(CustomerDetailsMailId,tmp);

  sprintf(tmp,"%d",_source_code);
  XmTextFieldSetString(CustomerDetailsSourceCode,tmp);
}
```

Figure 111 (Part 2 of 2). Customer.C: Customer Class Implementation

In the constructor, we can see that *Customer* objects are constructed from two pieces—*CustomerApp* and *CustomerGUI*(**1**).

Customer operator `<<()` (**2**) in our case only calls operator `<<()` from *CustomerApp* class, but if *CustomerGUI* had such an operator, it would typically be called here as well.

Connecting the data belonging to the two base classes occurs in function `void Customer::fill_GUI_fields(void)` (**3**). Here results of database query that are stored in data members of the CustomerApp class are transferred to data members in the CustomerGUI class.

8.8 Integration and Test

This section discusses a test scenario that demonstrates all the integration issues we needed to examine and prove successful during this phase of the project. This section also address how AIC 1.2 is used to make the changes necessary to integrate and test the C^{++} code.

8.8.1 Defining a Useful Test Scenario

Before we explain the AIC mechanics of integration, we need to define a test scenario and explore the programming issues it raises. While we have been talking about a derived class inheriting member functions and data from its base classes, we have not addressed how these are actually accessed during execution on an application. Public member data and functions are available to the main program and to member functions of other classes. Having defined our base and derived classes, we now needed to examine what belonged in the main executable program and look at how instances of one class caused the creation of instances of other independent classes. In other words, we needed to examine how to start the program going and tie the different windows together logically.

The program must have a the main module with an entry point defining where execution begins. After defining global data, the executing code must create its first instance of a class; then it can access the public member data and functions of that instance . Member functions of this object can in turn create other instances of classes and then access their public data and member functions. Iterations and recursions on this theme are how the execution of an application flows and grows.

A complete test scenario for our application would be complex. Consider, for instance a sequence of events that parallels one we demonstrated with the earlier versions of our application:

1. This main routine begins to execute when the application is started. It creates an instance of the *Customers* class, causing the Customers window to be displayed. This window is associated with the *Customers* class, which is derived jointly from the *CustomersGUI* and *CustomersApp* class. This instance inherits member data and functions from its base classes. The callback functionality associated with menus and push buttons on this window are member functions of this *Customers* class.
2. The user enters either a customer name or a customer number in one of the data entry fields on this window and then presses the **Search** push button.
3. The callback associated with this push button calls a member function, inherited from its base class *CustomerApp*, to query the database and extract all relevant records. This process makes use of member functions from the component classes providing DB2/6000 access: *dbserver*, *dbstmt*, and *dberror*.
4. Specific fields from these records are copied from data members inherited from the *CustomersApp* class, to data members inherited from the *CustomersGUI* class, by means of a member function of the *Customers* class. This member function is similar to `Customer::fill_GUI_fields(void)` shown at (**3**) in Figure 111 on page 184.
5. The callback function causes this data to be displayed.
6. Subsequently, the user highlights a specific customer record from the display, and selects **Show Details** from the Customer pulldown menu.
7. Another callback is triggered. This callback must create an instance of a new class, the *Customer* class. This class is associated with the Customer Details window. Now member data and functions from this class are available to the callback.
8. This same callback calls `Customer::get_by_custno()`, a member function of the *Customer* class. This function queries the database for information about this customer, using the customer number the callback passes it. This function has been inherited by the *Customer* class from its base class *CustomerApp*. This process indirectly makes use of the resources of the component classes providing DB2/6000 access: *dbserver*, *dbstmt*, and *dberror*.
9. The callback then calls `Customer::fill_GUI_fields(void)`, a member function in the *Customer* class. This function transfers data from the data members it inherits from the *CustomerApp* class, to data members it inherits from the *CustomerGUI* class.

10. Finally, the second window is displayed and the callback is completed.

We determined it was not necessary to implement this entire scenario to discuss the relevant test and integration issues in this book. We settled on an abbreviated version in which we do step 1 and then skip immediately to step 6 in the scenario. The user enters a customer number, instead of highlighting a customer record. We then execute steps 7 through 10.

8.8.2 Linking the Classes Together—Our Auxiliary Source Code Module

The piece of code that we have not examined yet is the auxiliary source module defining the subroutines executed by the callbacks in the GUI classes that provide linkage to the application classes. This module was introduced in 8.7.2, "CustomerGUI" on page 178 as the first of three options we envisioned employing to get around the problem that with the code generated by AIC 1.2 we could not actually override base class member functions for callbacks in the application classes. The subroutines in this module contain the code that would have been placed in the callback subroutine in the derived class. Therefore, this code, probably even stored as it is in an auxiliary module, would be required in our application regardless of which option we employed.

We named the auxiliary module `aux.C`. We named the header file for this module `aux.h`. A subroutine named `create_Customer` implements steps 7 through 9 of our test scenario. It is shown in Figure 112.

```
#include <iostream.h>
#include "aux.h"

swidget create_Customer(int CustomerNumber)
{
        theCustomer = new Customer(db2);                    1
        theCustomer->get_by_custno(CustomerNumber);         2
        theCustomer->fill_GUI_fields();                     3
        return  theCustomer->get_topLevelswidget();         4
}
```

Figure 112. Portion of aux.C.

This subroutine first creates a new instance of the *Customer* class (**1**), which implicitly creates the new widget for the Customer window. Then, it calls the member function *get_by_custno*, which it inherited from its base class, *CustomerApp*. The *get_by_custno*function issues the database query(**2**). Then *create_Customer* calls *fill_GUI_fields*, a member function that it inherited from its other base class, *CustomerGUI* class. This function fills the display fields with data just retrieved from the database(**3**) by *get_by_customerno*. Finally, *create_customer*, returns the top level widget of the window it has just created (**4**).

The code as shown carries a restriction,
because we have illustrated the most simple case only.This restriction is seen in the fact that there is only one (global) declaration of a variable for an instance of the *Customer* class. This sample code works, therefore, with no more than one customer detail window being shown at a time. In the proper application, this would be changed so multiple windows could be displayed at any given time, showing the details from different customers. Figure 113 on page 188 shows the file `aux.h`.

```
#ifndef aux_h
#define aux_h

#ifdef __cplusplus

#include "UxXt.h"

#endif

extern swidget create_Customers(void);
extern swidget create_Payments(void);                      1
extern swidget create_Customer(int CustomerNumber);
extern swidget create_Payment(void);

#ifdef __cplusplus

#include "dbserver.h"
#include "dberror.h"
#include "Customer.h"
#include "Payment.h"

extern dbserver* db2;
extern Customer* theCustomer;                              2
extern Payment* thePayment;

#endif

#endif
```

Figure 113. The aux.h File

The declarations of the function prototypes (**1**), as well as external declarations for the class instance variables used in the `aux.C` file (**2**), are contained in the `aux.h` file.

8.8.3 Integrating Our C^{++} Code with the Callbacks

The integration of the GUI and application code required interacting with AIC 1.2 to accomplish these three things:

- Modifications of the various callbacks of the interfaces to integrate the GUI code with the application logic we implemented in C^{++}.
- Modification of the main program, named `OOGUI`, to declare some global variables.
- Additions to the *makefile* file, `OOGUI.mk`, generated by AIC to include compile and link steps for the handwritten C^{++} application code.

The following sections elaborate on these steps.

8.8.4 Modifying the Callback Code

We needed to ensure that the call to the `create_Customer` would be included in the callback code generated by AIC at the appropriate place. To include this we did the following.

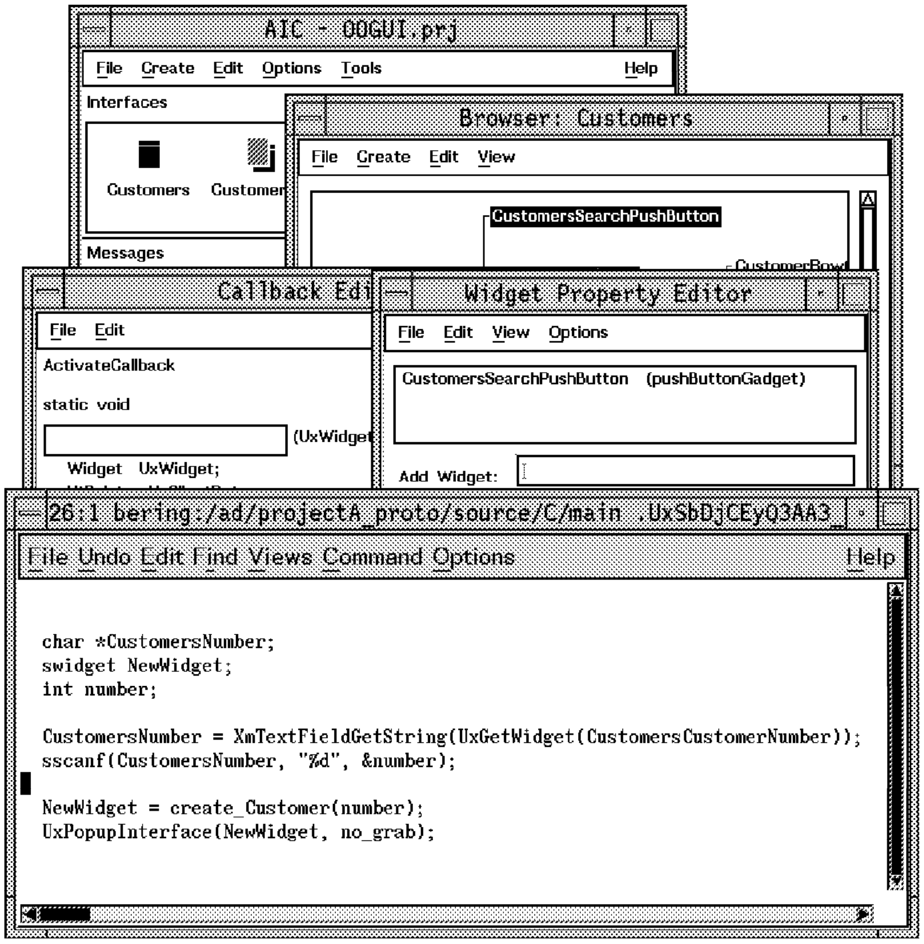
First, we ensured that `aux.h` would be included in the declarations portion of the source file that would be generated for the interface. We selected the Customers icon in the AIC main project window, pressed the right mouse button, and selected **Declarations...** from the pop-up menu. We pressed the **...** push button to the right of the *Includes, defines, and global variables* label and Program Editor was invoked. We added the line shown in Figure 114 on page 189 and selected **Save** from the File pull-down of Program Editor. For more about using the declarations file, refer to *AIXwindows Interface Composer Developer's Guide, Version 1.2*.

```
#include "aux.h"
```

Figure 114. Modification to the List of Include Files

Next we needed to ensure that the call to `create_Customer` would be included in the generated code for the callback. We selected **Close** from the File pull-down of Program Editor and pressed the **OK** button in the Declaration Editor window. We selected **Browser** from the Edit pull-down of the AIC project main window and selected **CustomersSearchPushButton** in the Browser window for the Customers widget. We selected **Property Editor** from the Edit pull-down and changed the toggle button from *Core* to *Behavior* in the Widget Property Editor window. We selected the **...** push button to the right of the *Activate callback* label, and selected **Modify code...** from the Edit pull-down of the Callback Editor window. This invoked Program Editor, which enabled us to edit the activate callback code for this widget. We added the call to *create_Customer*. At this point, we saw the overlapping windows shown in Figure 115 on page 190.

* 5.0x6.0



```
char *CustomersNumber;
swidget NewWidget;
int number;

CustomersNumber = XmTextFieldGetString(UxGetWidget(CustomersCustomerNumber));
sscanf(CustomersNumber, "%d", &number);

NewWidget = create_Customer(number);
UxPopupInterface(NewWidget, no_grab);
```

Figure 115. The Interface from the Callback to the Application Code

We selected **Save** from the File pull-down of Program Editor, closed Program Editor and pressed the **OK** button in the Callback Editor window. We then pressed the **Apply** push button in the Widget Property Editor window to activate the changes. We selected the **Save Project** push button of the File pull-down of the AIC main project window to save these changes.

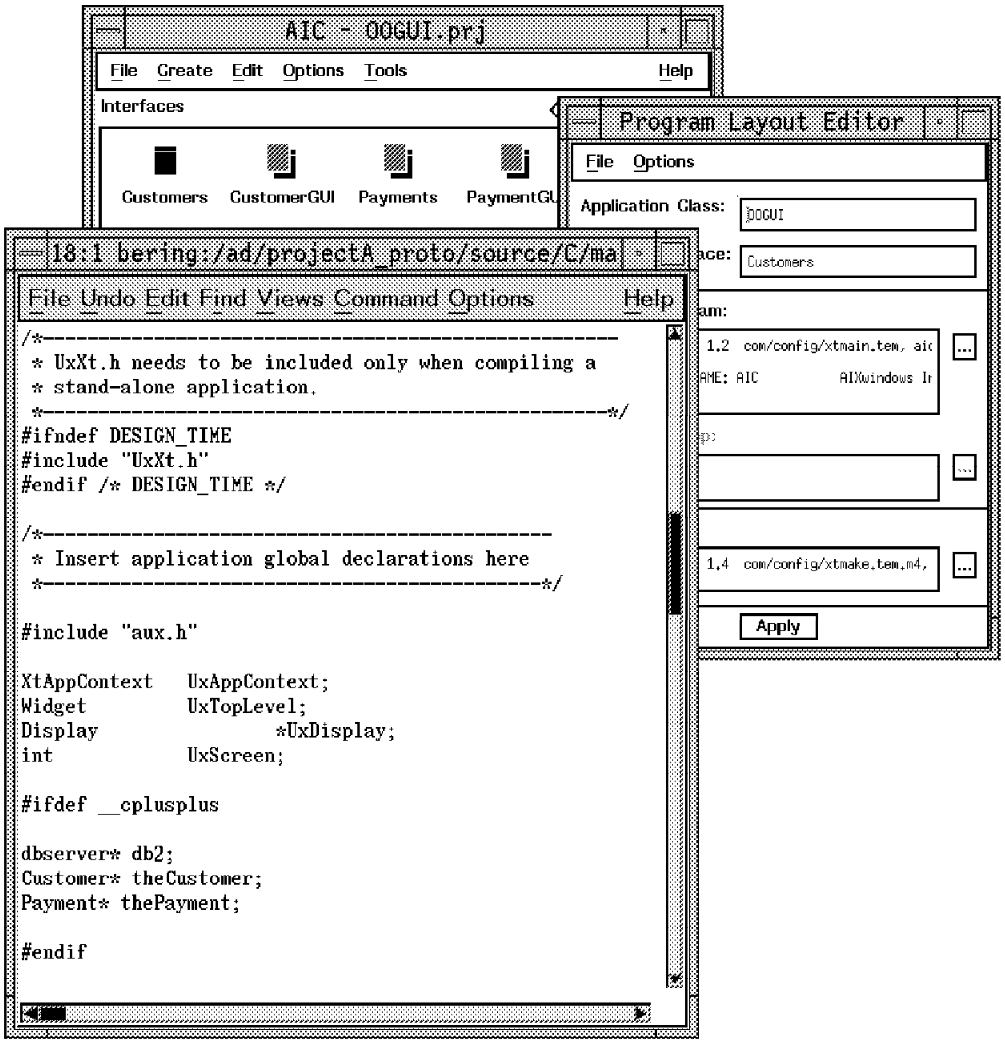
8.8.5 Modifying the Main Program

The main program required three modifications:

1. The `aux.h` file needed to be included at compile time.
2. Global variables for the *Customer*, *Payment*, and *dbserver* class instances needed to be declared.
3. Code needed to be added to ensure that database access errors would be handled properly.

The aux.h file is written in C^{++}, which is a superset of C. We had to make sure that the AIC 1.2 interpreter, which is based on a pure C parser, would not complain about any constructs which are unique to C^{++} when we tested the application. *AIXwindows Interface Composer Developer's Guide, Version 1.2* discusses the interpreter and suggests surrounding any pure C^{++} constructs with *#ifdef __cplusplus* and *#endif* directives. The *__cplusplus* macro variable is automatically set to a true Boolean value, and this tells the AIC interpreter to ignore them.

We selected **Program Layout ...** from the Edit pull-down of the AIC project window, and pressed the **...** button to the right of the Xt Main Program label. Program Editor was invoked, and we entered changes that would affect the main program code generated by AIC 1.2. Figure 116 shows some of these modifications being entered in the declaration part of the main program. For more information on modifying the AIC main program, see the discussion of generating main programs and makefiles in *AIXwindows Interface Composer Developer's Guide, Version 1.2*.



Application Class: OOGUI
ace: Customers

```
1.2  com/config/xtmain.tem, aic
AME: AIC          AIXwindows Ir
```

```
1.4  com/config/xtmake.tem.m4,
```

```
/*-----------------------------------------------------
 * UxXt.h needs to be included only when compiling a
 * stand-alone application.
 *-----------------------------------------------------*/
#ifndef DESIGN_TIME
#include "UxXt.h"
#endif /* DESIGN_TIME */

/*----------------------------------------------
 * Insert application global declarations here
 *----------------------------------------------*/

#include "aux.h"

XtAppContext   UxAppContext;
Widget         UxTopLevel;
Display                *UxDisplay;
int            UxScreen;

#ifdef __cplusplus

dbserver* db2;
Customer* theCustomer;
Payment* thePayment;

#endif
```

Figure 116. Modified Xt Main Program With Declarations for C^{++} Variables

Figure 117 on page 193 shows the code generated by AIC 1. 2 for the main program. It shows additional modifications we made in the area of the event loop. We added a call to the constructor of the *db2server* class (2), and the entire main program was embraced with a *try* construct (1) to catch C^{++} events in the corresponding *catch* directive (3). The *catch* part just printed an error message to standard output, but a more sophisticated exception routine could have been written and added here.

```
#ifdef _NO_PROTO
main(argc,argv)
        int     argc;
        char    *argv[];
#else
main( int argc, char *argv[])
#endif /* _NO_PROTO */
{
#ifdef __cplusplus
  try                                                     1
  {
#endif
        /*------------------------------------------------------------
         * Declarations.
         * The default identifier - mainIface will only be declared
         * if the interface function is global and of type swidget.
         * To change the identifier to a different name, modify the
         * string mainIface in the file "xtmain.dat". If "mainIface"
         * is declared, it will be used below where the return value
         * of  PJ_INTERFACE_FUNCTION_CALL will be assigned to it.
         *-----------------------------------------------------------*/

        $PJ_INTERFACE_RETVAL_TYPE

        /*---------------------------------
         * Interface function declaration
         *--------------------------------*/

        $PJ_INTERFACE_FUNCTION_DECL
        $PJ_INTERFACE_FUNCTION_ARG_DECL

        /*---------------------
         * Initialize program
         *--------------------*/
#ifdef __cplusplus
       db2 = new dbserver("customer","root","lorna");     2
#endif

#ifdef XOPEN_CATALOG
        XtSetLanguageProc(NULL,(XtLanguageProc)NULL,NULL);
#endif

        UxTopLevel = XtAppInitialize(&UxAppContext, "$PJ_APP_CLASS_NAME",
                             NULL, 0, &argc, argv, NULL, NULL, 0);

        UxDisplay = XtDisplay(UxTopLevel);
        UxScreen = XDefaultScreen(UxDisplay);

        /* We set the geometry of UxTopLevel so that dialogShells
           that are parented on it will get centered on the screen
           (if defaultPosition is true). */

        XtVaSetValues(UxTopLevel,
                        XtNx, 0,
                        XtNy, 0,
                        XtNwidth, DisplayWidth(UxDisplay, UxScreen),
                        XtNheight, DisplayHeight(UxDisplay, UxScreen),
                        NULL);

        /*-------------------------------------------------------
         * Insert initialization code for your application here
         *------------------------------------------------------*/


        /*----------------------------------------------------------------
         * Create and popup the first window of the interface.  The
         * return value can be used in the popdown or destroy functions.
         * The Widget return value of  PJ_INTERFACE_FUNCTION_CALL will
         * be assigned to "mainIface" from  PJ_INTERFACE_RETVAL_TYPE.
         *---------------------------------------------------------------*/

        $PJ_INTERFACE_FUNCTION_CALL
        $PJ_POPUP_CALL

        /*-----------------------
         * Enter the event loop
         *----------------------*/

        $PJ_EVENT_LOOP

#ifdef __cplusplus
  }
  catch(dberror x)                                   3
  {
    cout << x.msg() << endl;
  }
#endif
}
```

Figure 117. Modified Event Loop of Xt Main Program

After the modification were entered we selected **Save** from the File pull-down of Program Editor to save the changes. We selected **Close Document** to leave the Program Editor and pressed the **Apply** push button to activate the changes in the Program Layout Editor window.

8.8.6 Modifying the Makefile File

After all the changes had been coded, we had to adjust the*makefile* file generated by AIC to reflect our file structure and to add the various object modules that had been implemented manually. To do this, we selected **Program Layout...** from the Edit pull-down in the AIC project main window and pressed the **...** button to the right of the Xt Makefile label. Program Editor was started, showing us a template for the *makefile* file. When we were finished making our changes, we selected **Save** from the File pull-down of Program Editor to save the changes and closed Program Editor. We selected **Apply** in the Program Layout Editor window to activate the changes in AIC.

Figure 118 on page 195 shows the *makefile* file that AIC 1.2 later generated reflecting the edits we made at this time. These changes were:

1. We added the file names of the additional object modules that were implemented manually (**1**). Refer to 8.5.1, “Conventions and Files” on page 167 for a description of the additional source files external to AIC.

2. We added the DB2 libraries and the corresponding library path to the compile options (**2**).

3. We added the -I flag along with a file name to point to our include directory `/ad/projectA_proto/source/C/include` and to the DB2 include directory (**3**).

4. We added a rule to compile the additional source files (**4**).

For more about modifying the *makefile* file that AIC generates, see the discussion of generating main programs and makefiles in *AIXwindows Interface Composer Developer's Guide, Version 1.2*

```
EXECUTABLE      = $PJ_EXECUTABLE
MAIN            = $PJ_MAIN_SRC
INTERFACES      = $PJ_INTERFACES_SRC
LANGUAGE        = $PJ_LANGUAGE_OPTION
APPL_OBJS       = $PJ_SPECIAL_MODULES

EXTERNAL_OBJS  = /nfs/bering/ad/projectA_proto/source/C/app/aux.o \
                 /nfs/bering/ad/projectA_proto/source/C/app/CustomerApp.o \
                 /nfs/bering/ad/projectA_proto/source/C/app/Customer.o \
                 /nfs/bering/ad/projectA_proto/source/C/app/PaymentApp.o \                  1
                 /nfs/bering/ad/projectA_proto/source/C/app/Payment.o \
                 /nfs/bering/ad/projectA_proto/source/C/app/dberror.o \
                 /nfs/bering/ad/projectA_proto/source/C/app/dbstmt.o \
                 /nfs/bering/ad/projectA_proto/source/C/app/dbserver.o

UX_DIR          = /usr/lpp/aic12
UX_LIBPATH      = $(UX_DIR)/lib
X_LIBS          = -lXm -lXt -lX11

X_LIBPATH       =
MOTIF_LIBPATH   =
X_CFLAGS        =
MOTIF_CFLAGS    =

DB2_LIBPATH    = -L/usr/db2/lib                                                             2
DB2_LIBS       = -ldb2

KR_CC           = cc
ANSI_CC         = xlc
CPLUS_CC        = xlC
KR_CFLAGS       = -D_NO_PROTO
ANSI_CFLAGS     =
CPLUS_CFLAGS    = -+ -I/usr/lpp/xlC/include -I/usr/include \
                      -I/usr/db2/include -I/nfs/bering/ad/projectA_proto/source/C/include \    3
CFLAGS          = -D_BSD  -DXT_CODE -DXOPEN_CATALOG -DAIXV3 \
                  $(X_CFLAGS) $(MOTIF_CFLAGS)
LIBPATH         = $(X_LIBPATH) $(MOTIF_LIBPATH) $(DB2_LIBPATH)
LIBS            = $(DB2_LIBS) $(X_LIBS) -lm

OBJS = $(MAIN:$PJ_SOURCE_SUFFIX=.o) $(INTERFACES:$PJ_SOURCE_SUFFIX=.o) $(APPL_OBJS) \
       $(EXTERNAL_OBJS)                                                                     4

$(EXECUTABLE): $(OBJS)
        @echo Linking    $(EXECUTABLE)
        $(CC) $(OBJS) $(LIBPATH) $(LIBS) -o $(EXECUTABLE)
        @echo "Done"

.SUFFIXES:
.SUFFIXES: .o $PJ_SOURCE_SUFFIX .c

.c.o:
        @echo Compiling $< [$(LANGUAGE)] [XT-CODE]
        $(CC) -c $(CFLAGS) $< -o $@
$PJ_SOURCE_SUFFIX.o:
        @echo Compiling $< [$(LANGUAGE)] [XT-CODE]
        $(CC) -c $(CFLAGS) $< -o $@

CC = \
@ if [ "$(LANGUAGE)" = "C++" ]; then echo $(CPLUS_CC) $(CPLUS_CFLAGS);fi  \
 if [ "$(LANGUAGE)" = "ANSI C" ]; then echo $(ANSI_CC) $(ANSI_CFLAGS); fi \
 if [ "$(LANGUAGE)" = "KR-C" ]; then echo $(KR_CC) $(KR_CFLAGS); fi
```

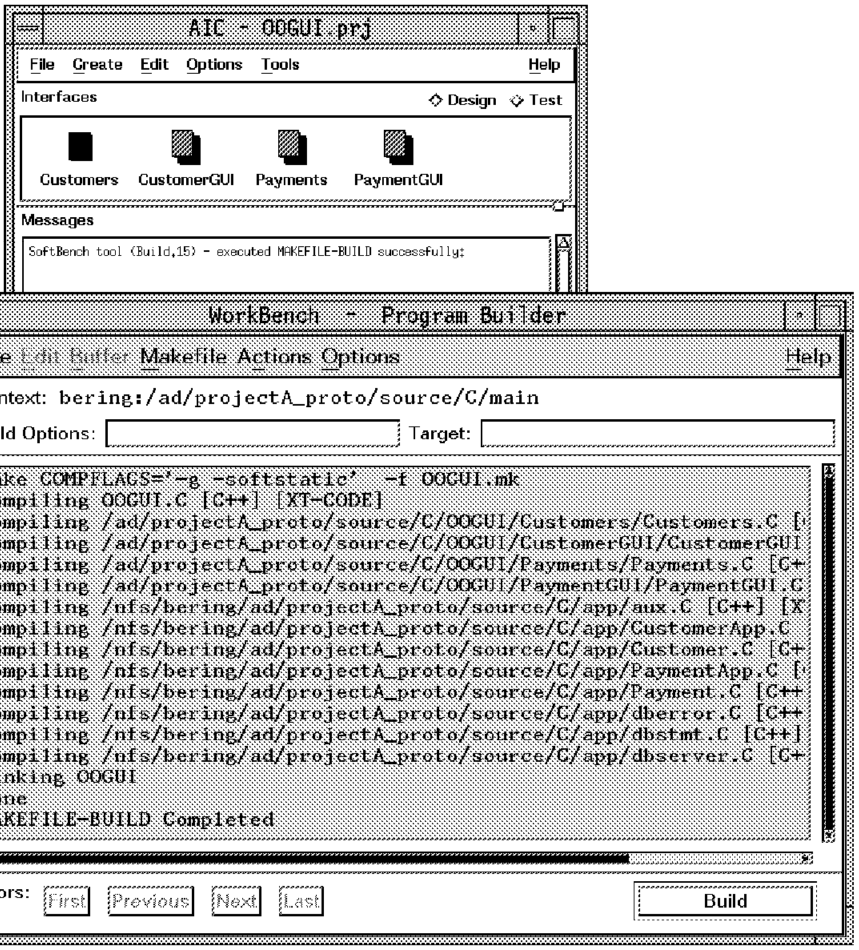
Figure 118. Modifications to the Makefile File Used for Our C^{++} Application

We now had implemented all the changes that were required to implement the interface to the C^{++} application code.

8.8.7 Generating Code and Test

Once all the modifications had been implemented, we were able to run the *makefile* file as it was generated by AIC. This *makefile* file included the modifications we had made to the *makefile* file template, and all the interfaces to the C^{++} application code had been implemented in the AIC callbacks.

We selected **Generate project Code...** from the File pull-down in the AIC main project window. AIC then generated the various source files and the *makefile* file, and would then started Program Builder to execute the *makefile* file. The discussion of generating code in *AIXwindows Interface Composer Developer's Guide, Version 1.2* explains this aspect of AIC 1.2. The Workbench Program Builder window is shown as it looked at the completion of this process in Figure 119.



```
SoftBench tool (Build,15) - executed MAKEFILE-BUILD successfully;
```

Context: bering:/ad/projectA_proto/source/C/main

```
make COMPFLAGS='-g -softstatic'  -f OOGUI.mk
Compiling OOGUI.C [C++] [XT-CODE]
Compiling /ad/projectA_proto/source/C/OOGUI/Customers/Customers.C [
Compiling /ad/projectA_proto/source/C/OOGUI/CustomerGUI/CustomerGUI
Compiling /ad/projectA_proto/source/C/OOGUI/Payments/Payments.C [C+
Compiling /ad/projectA_proto/source/C/OOGUI/PaymentGUI/PaymentGUI.C
Compiling /nfs/bering/ad/projectA_proto/source/C/app/aux.C [C++] [X
Compiling /nfs/bering/ad/projectA_proto/source/C/app/CustomerApp.C
Compiling /nfs/bering/ad/projectA_proto/source/C/app/Customer.C [C+
Compiling /nfs/bering/ad/projectA_proto/source/C/app/PaymentApp.C [
Compiling /nfs/bering/ad/projectA_proto/source/C/app/Payment.C [C++
Compiling /nfs/bering/ad/projectA_proto/source/C/app/dberror.C [C++
Compiling /nfs/bering/ad/projectA_proto/source/C/app/dbstmt.C [C++]
Compiling /nfs/bering/ad/projectA_proto/source/C/app/dbserver.C [C+
Linking OOGUI
Done
MAKEFILE-BUILD Completed
```

Figure 119. Generating the Final C^{++} Application From AIC

From either a terminal emulation, or the WorkBench Development Manager window, executing on *yellow*, we could now invoke the application and test it. The windows displayed would look very similar to those shown earlier in Figure 87 on page 132.

8.8.8 Overriding Callbacks in Application Classes

As indicated in 8.7.2, “CustomerGUI” on page 178 we would have preferred to implement the code found in the auxiliary source module in the context of the application class by overriding the callback function provided by the GUI class. This would require that the callback member function in the base class to be declared *virtual*.

If this were the case, the callback in the derived class would call a subroutine in our auxiliary source module, very similar to *create_Customer*. By overriding the GUI class callback, this member function of the derived class would also have to handle any GUI related functionality that might be required. However, it would be best if that functionality remained in the GUI class to maintain design integrity and prevent duplication of that code in the application class. Therefore, additional, general purpose member functions might be needed in the base class, which would be used in the derived class, for this purpose.

Part 3. Customizing and Tailoring AIX AD Products

Chapter 9. Tailoring SDE WorkBench/6000, Integrated Tools, and DB2/6000

This chapter describes how SDE WorkBench/6000 was tailored for the particular convenience of each of our developers. Customization for COBOL, C^{++}, and AIC are addressed. It also discusses how SDE WorkBench/6000 was tailored for our GUI developer. This chapter also contains a brief description of how AIC itself was tailored by our GUI developer.

9.1 DB2/6000 Installation, Initialization, and Shutdown

Before you install DB2/6000, we recommend that you read *DATABASE 2 AIX/6000 Information and Planning Guide*. The DB2/6000 release we installed, was a June 23, 1993 Beta release, and we used the *Beta Driver #2 Release Notes* as installation instructions.

We used the following procedure to install our DB2/6000 system:

1. Transferred the installation file onto our system (*bengal*).
2. Created a separate File System with mount point **/u/db2**.
3. Created a group by the name of **sysadm**.
4. Created an instance owner by the name of **db2**, with the home directory of **/u/db2**, and the primary group of **sysadm**.
5. Used SMIT to install DB2/6000 from the installation file.
6. Created symbolic links as per the *Beta Driver #2 Release Notes*.
7. Run the **db2instance** as root.
8. Included the environment variable definitions as defined in the **/u/db2/sqllib/db2profile** in the users **$HOME/.profile**.
9. Updated the **root** and the **aixcase2** users with **sysadm** group authority, as these users are our system administrators.
10. Started the database.
11. Tested the database.

After this we also updated the **aixcase2** users **.xsession**[2] file with the following line:

```
xset fp+ /u/db2/sqllib/dbat/fonts
```

This enabled the database administrator to use the OSF/Motif-based DBA tool. The DBA tool is invoked in a **aixterm** window or from the OSF/Motif Window Manager (MWM) by issuing the command **db2adm**. To enable the user to start the DBA tool from the MWM-menu, we include the following line into the **$HOME/.mwmrc** file:

```
"DB2ADM"    f.exec "db2adm &"
```

[2] The update to the **.xsession** file is because the user aixcase2 is using the xdm environment. Normally you would put the same line into the **.xinit** file.

9.2 Tailoring SDE WorkBench/6000 for COBOL Programmers

In our project we decided to make full use of SDE WorkBench/6000, and do all development in SDE WorkBench/6000. So even if Micro Focus COBOL Toolbox comes as a stand-alone environment, we decided to integrate it with SDE WorkBench/6000. The default tool started by the Animate item in the Development Manager Actions pull-down menu for a **.int** file type is **anim**. This starts the nonencapsulated version of the Micro Focus COBOL Animator debug tool. Version 3.0 of Micro Focus COBOL Toolbox includes a version of the Animator that has been integrated with SDE WorkBench/6000. It is called **softanim**. So we wanted to create a separate pull-down menu for Micro Focus COBOL in the WorkBench Development Manager Action pull-down menu, and create a new selection in the Actions pull-down menu labelled **softanim**. The following paragraphs take you through the steps of integrating the above items with SDE WorkBench/6000.

9.2.1 Creating a Separate Micro Focus COBOL Menu Item

There are two ways of creating new menu items in the Development Manager.

- SDE WorkBench/6000 provides a Menu tree, **/usr/softbench/menus**, where additional menu directories can be created and menu files can be placed. If the new menu files are put in the correct place, they are picked up by the hosting tool automatically. Files and directories are added to the menu directory tree by the system administrators when they integrate additional tools with SDE WorkBench/6000.
- The other choice is to place the file in an arbitrary directory created by the user, and direct the hosting tool to pick up the menu using the **userMenus** resource. User-defined menus are typically handled in this manner.

We chose to do the latter, as our programmers need to access different tools in their development. So our COBOL programmer created a new directory in his home directory called **menus**, and in this directory he created a new directory called **mf**. In the **mf** directory we created two files, the **$LANG** and **MDF.m** files, as shown in Figure 120 and Figure 121 on page 203, where the **$LANG** represents our language locale (in our case En_US). To accommodate for different language locales, there should be one file for every locale used on the system. The **$LANG** language environment variable controls which resource file is read. If **$LANG** is not set, the **C** resource file, which contains American English text, is used as the default.

```
! Add a menu for MicroFocus tools to the DM Menu bar

Softdm*menuBarButtonList:                 DM_mf_menu

Softdm*DM_mf_menu.labelString:            MicroFocus
Softdm*DM_mf_menu.mnemonic:               F

Softdm*DM_mftde.labelString:              Micro Focus Toolbox
Softdm*DM_mftde.mnemonic:                 x

Softdm*DM_mfds.labelString:               Micro Focus Dialog System
Softdm*DM_mfds.mnemonic:                  D
```

Figure 120. The En_US Resource File

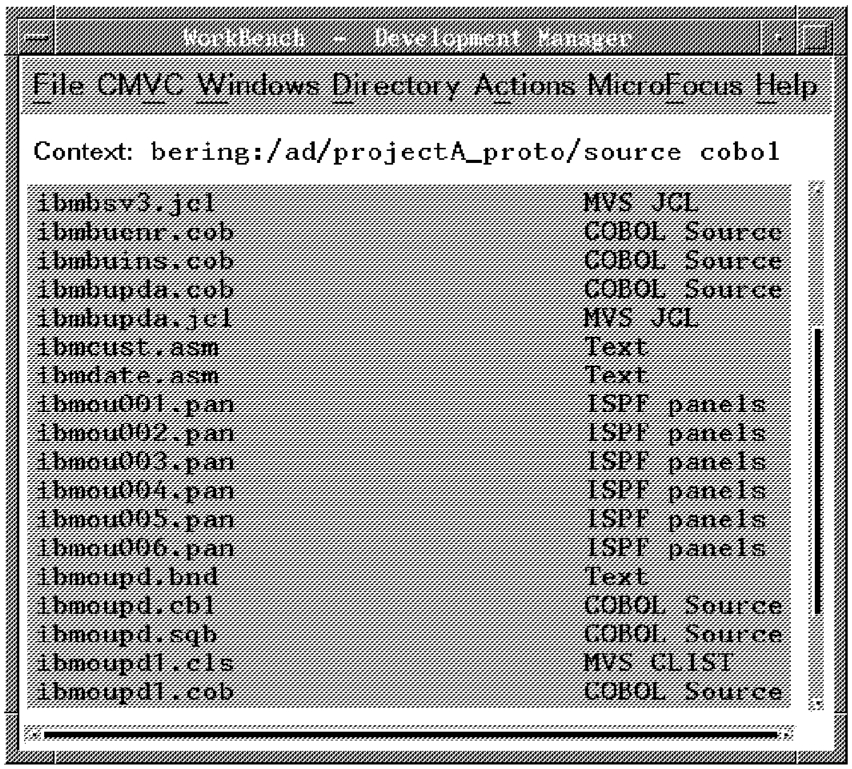
```
# Add items for MicroFocus tools menu in DM window

Menu DM_mf_menu
 {
  DM_mftde f_exec MenuMsgObjectFunc "TERM TERMINAL \
   %Context_Host% 2 %Context_Dir% 2 - - %Context_Host% 2 tbox"
  DM_mfds f_exec MenuMsgObjectFunc "TERM TERMINAL \
   %Context_Host% 2 %Context_Dir% 2 - - %Context_Host% 2 ds"
 }
```

Figure 121. The MDF.m Menu Description File

By setting the application resource in **$HOME/.Xdefaults** to:

```
Softdm.userMenus: /home/aixcase2/menus/mf
```

our menu files can be read by the hosting tool, in this case the Development Manager. After having restarted SDE WorkBench/6000, our *new* Development Manager looks like Figure 122.


WorkBench - Development Manager
File CMVC Windows Directory Actions MicroFocus Help

Context: bering:/ad/projectA_proto/source cobol

```
ibmbsv3.jcl                  MVS JCL
ibmbuenr.cob                 COBOL Source
ibmbuins.cob                 COBOL Source
ibmbupda.cob                 COBOL Source
ibmbupda.jcl                 MVS JCL
ibmcust.asm                  Text
ibmdate.asm                  Text
ibmou001.pan                 ISPF panels
ibmou002.pan                 ISPF panels
ibmou003.pan                 ISPF panels
ibmou004.pan                 ISPF panels
ibmou005.pan                 ISPF panels
ibmou006.pan                 ISPF panels
ibmoupd.bnd                  Text
ibmoupd.cbl                  COBOL Source
ibmoupd.sqb                  COBOL Source
ibmoupd1.cls                 MVS CLIST
ibmoupd1.cob                 COBOL Source
```

Figure 122. Our New Development Manager

We also changed the file **/usr/softbench/config/termsrv.config** and used the file **/usr/softbench/config/termsrv.general**, instead, to get the right apearance of the Toolbox window as shown in Figure 64 on page 107.

To read more about adding choices to existing menus, refer to *IBM AIX SDE Integrator/6000 Distributing and Encapsulation*.

9.2.2 Integrating Softanim Tool

Softanim is an encapsulation of the Micro Focus Animator debugger for use in the SDE WorkBench/6000. The Animator screen is displayed in a window, with a menu bar to access SDE WorkBench/6000 functions and a label describing the current context file. To integrate the Animator with SDE Workbench/6000, you can either modify the **/usr/softbench/config/softinit** file, or copy this file to your home directory, and rename it to **.softinit**. In our case we choose the latter.

The file **/usr/softbench/config/softinit** provides SDE WorkBench/6000 with the information necessary to start applications properly. If you do not have the environment variable **SOFTINIT** defined, SDE WorkBench/6000 reads the **softinit** files in the following order:

```
/usr/softbench/config/softinit
$HOME/.softinit
```

Therefore, you only need to specify the addition or modifications to the system default entries in `$HOME/.softinit`.

If the **SOFTINIT** environment variable is defined, SDE WorkBench/6000 reads the files specified by the content of that variable for softinit information. You should normally have **/usr/softbench/config/softinit** as the first file in this list. The files listed in **SOFTINIT** are read in order. If there are multiple entries of the same tool class operating on the same file types, the last one read takes precedence. Thus, you can modify any tool initiation specification by having a local version of **softinit** in your home directory and specifying it last in the **SOFTINIT** list, as in the following **Korn** shell example:

```
SOFTINIT="/usr/softbench/config/softinit $HOME/mysoftinit"
export SOFTINIT
```

After having copied the **/usr/softbench/config/softinit** to our home directory, and having renamed it **.softinit**, we added the lines as shown in Figure 123.

```
# Micro Focus COBOL Animator as Debugger for int files
DEBUG TOOL FILE COBOL_INTER %Local% softanim -host %Host% -dir %Directory% -file %File%
```

Figure 123. The New Entries in the .softinit File

9.2.3 Creating the Softanim Item

Having done this, we have to create an Actions Menu item. The **Actions** menu is located in menu directories below **/usr/softbench/menus/Softdm/ACTIONS**. The key difference between the **ACTIONS** menu directories and the other application menu directories is that *all* the **ACTIONS** menu directories are read whereas only a single application menu directory per tool class is read. The Development Manager uses this capability to define panes that are dynamically attached to the **Actions** menu bar button. All **ACTIONS** menus must be available simultaneously, since the user is free to select any file type in the Development Manager directory list at any time.

In our case, we created a subdirectory called **mfcobol**. In this directory we created one resource file and one menu description file. As we have explained earlier, the **$LANG** environment variable controls which resource file is read, and if **$LANG** is not set, the **C** resource file is used. So we created just a **C** resource file. The contents of our resource and menu description files are shown in Figure 124 on page 205, and in Figure 125 on page 205.

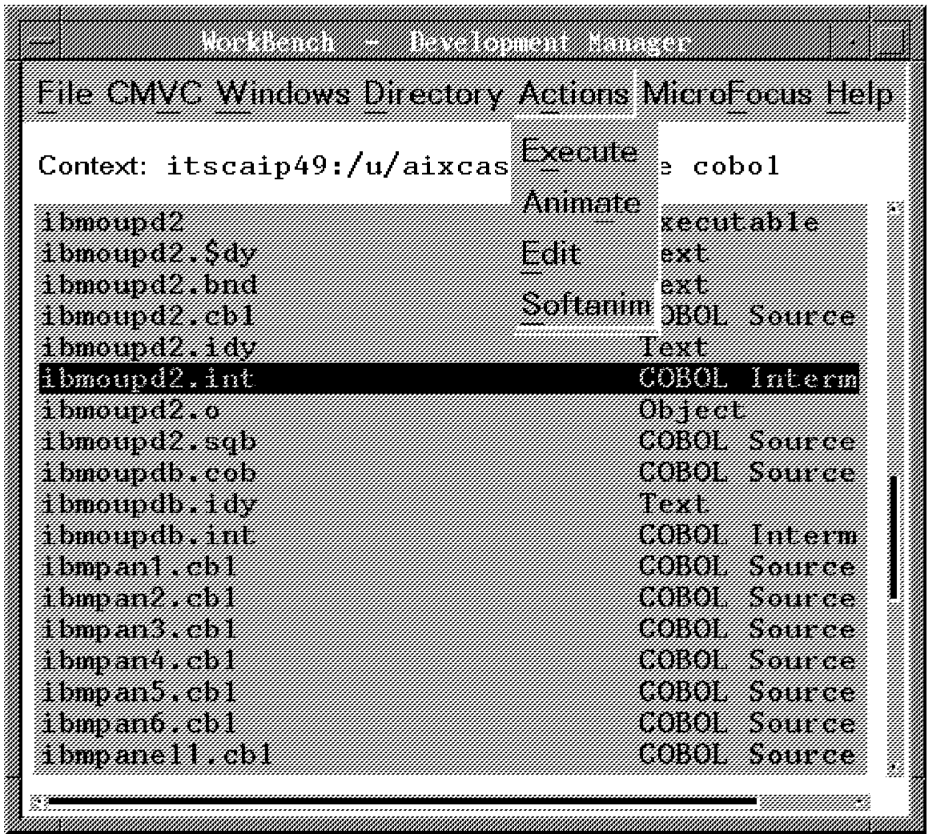
```
! Add action to softtype cobolinter for MF integrated animator
Softdm*DM_cobolinter_filesoftanim.labelString: Softanim
Softdm*DM_cobolinter_filesoftanim.mnemonic:    S
```

Figure 124. The Resource File for the Softanim Item

```
# Add new item to cobolinter softtype for MF integrated animator
Menu cobolinter
 {
  DM_cobolinter_filesoftanim f_exec MenuMsgSelectFunc "DEBUG START"
 }
```

Figure 125. The Menu Description File for the Softanim Item

Figure 126 shows the new **Softanim** item in the Actions pull-down menu. For more information about adding items to the **Actions** menu, and integrating new tools with Development Manager, refer to *IBM AIX SDE Integrator/6000 Distributing and Encapsulation* and *IBM AIX SDE Integrator/6000 Programmer's Guide*.


WorkBench - Development Manager
File CMVC Windows Directory Actions MicroFocus Help
Execute
Animate
Edit
Softanim

```
Context: itscaip49:/u/aixcas        e cobol
ibmoupd2                            xecutable
ibmoupd2.$dy                        ext
ibmoupd2.bnd                        ext
ibmoupd2.cbl                        OBOL Source
ibmoupd2.idy                        Text
ibmoupd2.int                        COBOL Interm
ibmoupd2.o                          Object
ibmoupd2.sqb                        COBOL Source
ibmoupdb.cob                        COBOL Source
ibmoupdb.idy                        Text
ibmoupdb.int                        COBOL Interm
ibmpan1.cbl                         COBOL Source
ibmpan2.cbl                         COBOL Source
ibmpan3.cbl                         COBOL Source
ibmpan4.cbl                         COBOL Source
ibmpan5.cbl                         COBOL Source
ibmpan6.cbl                         COBOL Source
ibmpanel1.cbl                       COBOL Source
```

Figure 126. The Softanim Item in the Actions Pull-Down Menu

9.2.4 Tailoring Program Editor

Program Editor is a *live parsing* editor, which uses different color schemes to aid the programmer in the editing. Tailored macros can be built, and the support for various programming languages can be accommodated. Program Editor comes bundled with support for C++, C, COBOL, FORTRAN, and Data Composition Language (DCF).

If you want to tailor Program Editor to your own needs, without affecting other users, you should create a directory in your home directory, like:

```
mkdir lpex_dir
```

and place your own macros in this directory.

After you have done this, and created your own macros in this directory either by copying existing ones or by writing new ones, you have to tell Program Editor where to look for tailored macros. You do this by setting the environment variable `MY_LPEX_DIR` to the directory just created, like:

```
export MY_LPEX_DIR=$HOME/lpex_dir
```

You should also put this line in your **.profile**.

In our case, we wanted support for COBOL files with the extension of **.sqb**, because our DB2/6000 precompiler required this naming convention. To enable Program Editor to recognize any file with the extension of **.sqb** as a COBOL file, we copied the file `cob.LXL` from the `/usr/softbench/lpex/lpex/macros` directory to our `$HOME/lpex_dir`, and renamed it to `sqb.LXL`. We also needed to tell the Development Manager that all files with the extensions of **.sqb** are COBOL files, so they show up correctly in the WorkBench Development Manager window. To do this, we had to log in as `root`, and then do the following:

```
cd /usr/softbench/config/softtypes/config/$LANG
```

In this directory we modified the file called `ibm_cobol`. After the modification, the file looked like Figure 127.

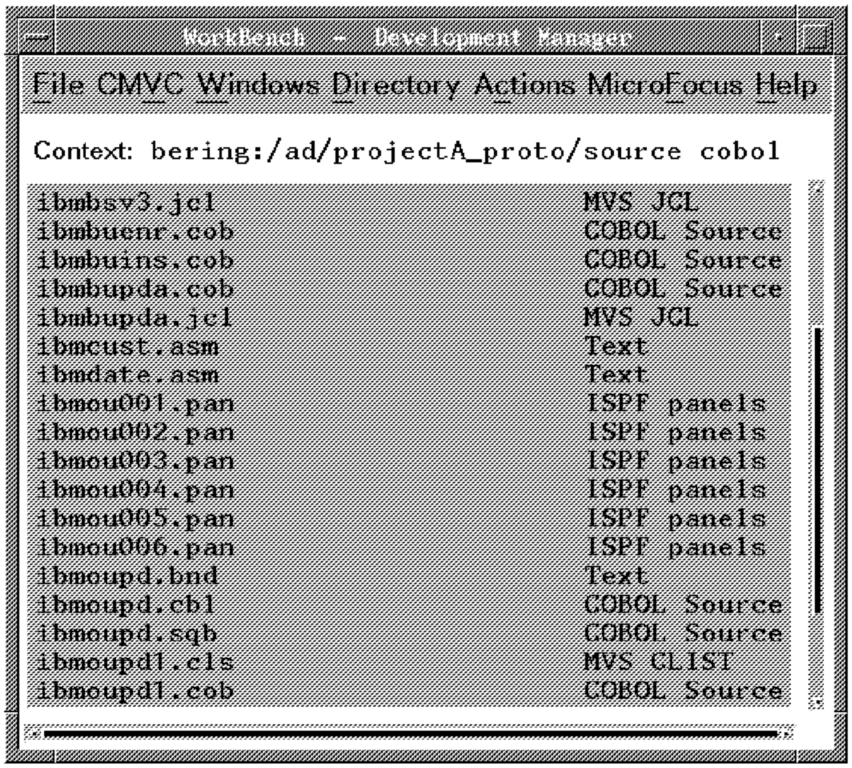
```
COBOL Intermediate|    .int      COBOL_INTER    cobolinter
COBOL Source|          .sqb      COBOLSOURCE    source
COBOL Native|          .gnt      COBOL_NATIVE   cobolnative
Build|                 .mk       BUILD          make
ISPF panels|           .pan      MVS_ISPF       generic
MVS CLIST|             .cls      MVS_CLIST      generic
MVS JCL|               .jcl      MVS_JCL        generic
```

Figure 127. The ibm_cobol File

As you notice, we have also added entries for our non-AIX-files, like the ISPF panel-files that have the extension `.pan`. When we are finished editing this file, we have to merge our new **softtypes** with the existing ones. This merging is done by:

```
/usr/softbench/etc/merge-types
```

After having done this, we have to restart SDE WorkBench/6000. The effect of the change is shown in Figure 128 on page 207.


WorkBench - Development Manager
File CMVC Windows Directory Actions MicroFocus Help

```
Context: bering:/ad/projectA_proto/source cobol

ibmbsv3.jcl                        MVS JCL
ibmbucnr.cob                       COBOL Source
ibmbuins.cob                       COBOL Source
ibmbupda.cob                       COBOL Source
ibmbupda.jcl                       MVS JCL
ibmcust.asm                        Text
ibmdate.asm                        Text
ibmou001.pan                       ISPF panels
ibmou002.pan                       ISPF panels
ibmou003.pan                       ISPF panels
ibmou004.pan                       ISPF panels
ibmou005.pan                       ISPF panels
ibmou006.pan                       ISPF panels
ibmoupd.bnd                        Text
ibmoupd.cbl                        COBOL Source
ibmoupd.sqb                        COBOL Source
ibmoupd1.cls                       MVS CLIST
ibmoupd1.cob                       COBOL Source
```

Figure 128. The Appearance of New softtypes

For more information about **softtypes**, refer to the *IBM AIX SDE Integrator/6000 Distributing and Encapsulation*.

9.3 Tailoring SDE WorkBench/6000 for C++ Programmers

Although SDE WorkBench/6000 supports C++ development, we customized the following tools to tailor SDE WorkBench/6000 for our project:

- Program Editor
- Development Manager
- Program Builder.

Customization of Program Editor was done on a global basis, to accommodate all C++ users. Customization of Development Manager and Program Builder was done on a local per user basis.

9.3.1 Customization of Program Editor

We customized Program Editor to change the default parsing based on file naming convention. By default, Program Editor treats files with extension **.H** as C++ header files, and files with extension **.h** as C header files. We wanted C++ header files to have extension **.h**, because all XL C++ system header files have this extension (for example, `iostream.h`).

To instruct Program Editor to treat *.h* files as C++ files we created our own version of the h.LXL load macro file, as shown in Figure 129 on page 208, and made it available to Program Editor by following the instructions in *IBM AIX SDE WorkBench/6000 Program Editor*.

```
/****************************************************
*                                                   *
* h.LXL - load macro for .h (C++ header) files      *
*                                                   *
* This macro is invoked by Program Editor when a    *
* file with an extension of .h is edited, unless    *
* the /asis or /nopro options are used. It sets up  *
* the C++ emphasis parser and fonts for a color     *
* display.                                          *
*                                                   *
****************************************************/

'lxr macro C.LXL'
```

Figure 129. Contents of the h.LXL File

Our customization consisted of the following steps:

1. Create a /usr/local/SDE_WorkBench_Cust directory.
2. Create the h.LXL file in that directory.
3. Set up the environment variable **LPEXDIR** adding the lines shown in Figure 130 to the /etc/environment file.
4. Log off and then log in again, to enable the changes to take effect.
5. Restart the SDE WorkBench/6000.

Note that only users with system administration authority (*root*) can perform steps 1 to 3, because those steps are executed on files and directories for which ordinary users do not have write authority.

```
#################################################################
# Environment variables for SDE WorkBench Program Editor
#################################################################

LPEXPATH=/usr/local/SDE_WorkBench_Cust /usr/softbench/lpex/lpex/macros
```

Figure 130. Lines Added to the /etc/environment File

9.3.2 Customization of Development Manager

We customized Development Manager by specifying the default file selection filter for our C++ development directories. We created a filter, as shown in Figure 131 on page 209, to display only C++ source files and *makefile* files and to make it easier to spot them among other files created dynamically during the development process (for example, files with extension **.o** or **.u**).

```
Softdm*Filter_label_0       : *.C *.h
Softdm*Filter_expression_0  : ^(.*)[.]((C)|(h))$
Softdm*Filter_IsRegex_0     : TRUE
Softdm*Filter_activate_0    : TRUE
Softdm*Filter_ByExclusion_0 : FALSE

Softdm*Filter_label_1       : Makefile
Softdm*Filter_expression_1  : ^((m)|(M)akefile)
Softdm*Filter_IsRegex_1     : TRUE
Softdm*Filter_activate_1    : TRUE
Softdm*Filter_ByExclusion_1 : FALSE
```

Figure 131. Lines Added to the .softbenchrc File

The detailed syntax for writing filter resources is explained in *IBM AIX SDE WorkBench/6000 Development Manager:Managing Files and Directories*.

9.3.3 Customization of Program Builder

We customized the *makefile* template for automatic generation of *makefile* files, as shown in Figure 132, modifying the copy of the `/usr/lpp/workbench/config/buildt.p` file. We removed lines not relevant to C^{++}, and added some, including comments, that we needed for our project.

```
##-----------------------------------------------------------------+
##
## makefile for SDE WorkBench
##
## This file has been generated automatically                  1
##
##-----------------------------------------------------------------+

COMPFLAGS    = -c $(DEBUG)

CXXFLAGS     = $(COMPFLAGS)

COMP         = $(CCC)

LD           = $(CCC)

LDFLAGS      = $(DEBUG)

PROGRAM      =
```

Figure 132 (Part 1 of 3). Modified Makefile Template for Program Builder

```
##----------------------------------------------------------------+
##
## Uncomment the following lines and run Makefile/Update dependencies
## to include:
##
##             system header files
##SYSHDRS      =                                                  2
##
##             XL C++ header files
##EXTHDRS      =                                                  3
##
##             application header files
##HDRS         =                                                  4
##
##             application source files
##SRCS         =                                                  5
##
##----------------------------------------------------------------+

##----------------------------------------------------------------+
## LIST OF OBJECT FILES AND LIBRARIES                             6
##----------------------------------------------------------------+

OBJS         =

LIBS         =

##----------------------------------------------------------------+
## GLOBAL DEFINITIONS
##----------------------------------------------------------------+

.C.o:
        $(COMP) $(CXXFLAGS) $<
```

Figure 132 (Part 2 of 3). Modified Makefile Template for Program Builder

```
##-----------------------------------------------------------------+
## LINK LINE
##-----------------------------------------------------------------+

$(PROGRAM): $(OBJS) $(LIBS)
        $(LD) $(LDFLAGS) $(OBJS) $(LIBS) -o $@

##-----------------------------------------------------------------+
## UTILITY LINES
##-----------------------------------------------------------------+

depend:;     @mkmf -f $(MAKEFILE) ROOT=$(ROOT)

clean:;      @rm -f $(OBJS) core

clobber:;    @rm -f $(OBJS) $(PROGRAM) core

##-----------------------------------------------------------------+
## DEPENDENCIES                                                  7
##-----------------------------------------------------------------+
```

Figure 132 (Part 3 of 3). Modified Makefile Template for Program Builder

The *makefile* template is used for automatic generation of *makefile* files. Some lines have a specific meaning for `mkmf`, the program invoked by Program Builder to generate *makefile* files:

1 This is just a comment line. It appears as written. Note that the text is true for the generated *makefile* file, rather than for the *makefile* template where it appears.

2 Uncommenting this line causes system header files (for example, `stdio.h`) to be included in the dependency lines.

3 Uncommenting this line causes XL C++ header files (for example, `iostream.h`) to be included in dependency lines. To do so, put the `/usr/lpp/xlC/include` option on the *DIRS* text field of the Program Builder window (the top window in Figure 40 on page 84). If both system and XL C++ header files are to be included, the *DIRS* text field must contain `/usr/include /usr/lpp/xlC/include` in that order.

4 Uncommenting this line causes application header files to be listed in the *makefile HDRS* variable. The files will, however, appear on dependency lines, whether or not the *HDRS* variable was commented out.

5 Same as **4**, with respect to the *SRCS* variable.

6 The *OBJS* and *LIBS* variables contain the application's object files and libraries.

7 *mkmf* places dependency lines at the end of the generated *makefile* file.

To instruct Program Builder to use the customized version of the `makefile` template, we added a line (Figure 133 on page 212) to the `.softinit` file.

```
Build*progTemplateFile : /home/aixcase1/SDE_WorkBench_Cust/project_Makefile
```

Figure 133. Line Added to .softbenchrc File

9.4 Tailoring SDE WorkBench/6000 for AIC Programmers

Some of the customization instructions in this chapter are based on the IBM publication *IBM AIX SDE Integrator/6000 Distributing and Encapsulation*; so make sure you have this manual even if you do not intend to order the product IBM AIX SDE Integrator/6000.

In this chapter we explain the various levels of integrating AIC with the SDE WorkBench/6000.We applied the following steps to customize SDE WorkBench/6000 we used throughout our development:

1. We added a new entry to the tools list of SDE WorkBench/6000 so we could start AIC from the tools list of the Tool Manager Start window.
2. We added new entries for the extensions of the AIC interface and project files so Development Manager would display a proper descriptive text for these AIC files.
3. We added new pull-down menus to the Actions pull-down of Development Manager so AIC-specific functions would be called for AIC interface, project, or palette files.
4. We changed the default action triggered from Development Manager so a double click on an AIC interface, project, or palette file would invoke AIC.
5. We also customized AIC so SDE WorkBench/6000 tools such as Program Editor or CMVC would be invoked directly from AIC. This integration step is supported in AIC Release 1.2 and above, and not in the preceding releases of AIC. This customization offered a much better integration.

There is more on this subject in the discussion of running AIC under WorkBench in *AIXwindows Interface Composer Developer's Guide, Version 1.2*.

9.4.1 Modify the Tools List of Tool Manager

To start AIC as a user interface builder tool from the Tool Manager Start window in our environment, we considered AIC as a tool from the new class,*UIBUILD*. We wanted to add a new entry for this tool class to the list of tools available to SDE WorkBench/6000.

By default Tool Manager uses the `/usr/softbench/config/softinit` file followed by the user's `$HOME/.softinit` file to show the list of tools. Each user can, however, use the *SOFTINIT* environment variable to change this order or even the files SDE WorkBench/6000 looks for. An additional entry for a new tool can therefore be made to:

- The system-wide softinit file

  or
- A private softinit file (like the `.softinit` in the `$HOME` directory).

To add a new tool as a default tool for a new class, follow these steps exemplified with the integration of AIC:

1. Log in as *root*, edit the `/usr/softbench/config/softinitsrc/tool-override/ui` file, and add the lines shown in Figure 134 to the end of the file when running AIC Release 1.2.

```
UIBUILD TOOL DIR * %Host% /usr/lpp/aic12/bin/aic12 -scope \
    dir -dir %Directory%
```

Figure 134. Entry to Define AIC Rel. 1.2 as a New Tool to Tool Manager

   When running AIC releases previous to 1. 2 add the line as shown in Figure 135 to the end of the file.

```
UIBUILD   TOOL    FILE   *      %Host%      aic
```

Figure 135. Entry to Define AIC as a New Tool to Tool Manager

   This line defines a new tool class UIBUILD on the local host, and SDE starts this tool using the **aic** command.

2. Run the **/usr/softbench/etc/merge-init** utility.

   This utility reads the modified file and updates the system-wide `/usr/softbench/config/softinit` file. We do not recommend editing this file directly, as various tool vendors and system integrators also contribute to this file.

To add a new tool for an individual user, edit the user's private`.softinit` file and add an entry like that in Figure 134 or Figure 135. You do not need to run a utility to activate the changes. SDE WorkBench/6000 reads the modified file the next time you start it.

Once you have modified the tools list and started SDE WorkBench/6000 again, you can select **Start** from the Tool menu of the Tool Manager window and get to the Tool Manager Start window as shown in Figure 136 on page 214, which includes an entry for the tool class UIBUILD.

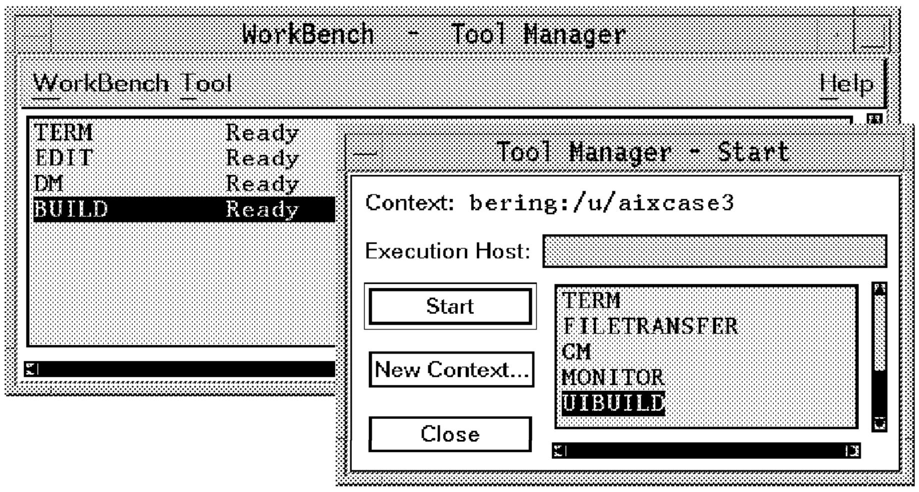


Figure 136. A Tool Manager Start Window with an Entry for the UIBUILD Tool Class

9.4.2 Modify Extension-Based File-Typing of Development Manager

The **Save** selection from the File menu of AIC saves the data being edited into *interface*, *palette*, or *project* files. These files store the window definitions in a format that is specific to AIC so other AIC utilities like **uxcgen** for code generation or AIC itself can interpret the data. These files have the specific file type extensions of `.i`, `.pal`, or `.prj`.

We wanted these AIC-specific files to be identified in Development Manager so Development Manager could recognize a file with an extension of `i`, `.pal`, or `.prj` as an AIC interface, palette, or project file. Development Manager could then display a descriptive string along with the file name to further explain the contents of the file.

Development Manager uses the `/usr/softbench/config/softtypes/$LANG` file to recognize particular file types, where *$LANG* is the language environment variable of the user. There are different translated versions of the `softtypes` file, because Development Manager should display different strings for different languages. If the *$LANG* environment variable is not set, Development Manager looks for the `/usr/softbench/config/softtypes/C` file.

To add new file type extensions to Development Manager follow these steps as exemplified by adding AIC file type extensions:

1. Log in as *root* and edit the `/usr/softbench/config/softtypes/config/$LANG/softtypes` file, where *$LANG* is the language for which you want to modify the descriptive strings. In our case we modified the `/usr/softbench/config/softtypes/config/En_US/softtypes` file for users running with a US English locale (En_US). If, for example, you want to support the German locale De_DE, you need to put a translated `softtypes` file into the corresponding directory `/usr/softbench/config/softtypes/config/De_DE` for users running with a German locale (De_DE).

Add the lines shown in Figure 137 to the end of the `/usr/softbench/config/softtypes/config/En_US/softtypes` file.

```
AIC Interface File “   .i       UIAICINTERFACE     aicInterface
AIC Project File   “   .prj     UIAICPROJECT       aicProject
AIC Palette File   “   .pal     UIAICPALETTE       aicPalette
```

Figure 137. Entries to the softtypes File for Development Manager

Each line of the file specifies:

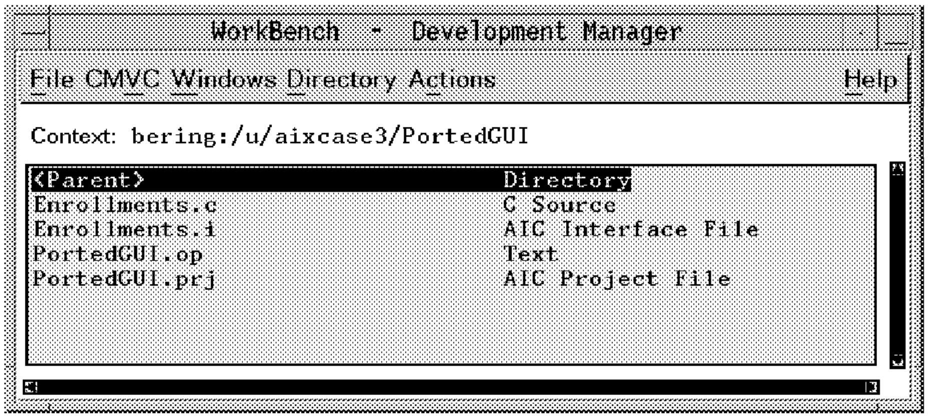
- The descriptive string to be displayed by Development Manager
- The file type extension of the file that matches to the description
- A unique identification of the file type extension
- A menu definition for a pull-down menu that is displayed when you select a file with this file type and pull down the **ACTIONS** menu.

Refer to 9.4.3, “Modify the Action Pull-Down of Development Manager” on page 216 for an explanation about how to define the additional menu. If you do not want to add new Action menus, specify `generic` instead of `aicInterface`, `aicProject`, and `aicPalette`. You then get the same **ACTIONS** pull-down menu for these file types as for h source files, and the offered selections are **Edit** and **Print**.

2. Run the **/usr/softbench/etc/merge-types** utility.

   This utility reads the modified file and updates the system-wide`/usr/softbench/config/softtypes/$LANG` files. For example, when you modified the `/usr/softbench/config/softtypes/config/En_US/softtypes` file as shown above and run **/usr/softbench/etc/merge-types**, the `/usr/softbench/config/softtypes/En_US` file gets updated. We do not recommend that you edit this file directly, as the next invocation of **merge-types** will overwrite the changes from the specified sources.

Once you have made the modifications and started SDE WorkBench/6000 again, Development Manager recognizes the AIC-specific file type extensions and displays the proper descriptive text as shown in Figure 138 on page 216.


WorkBench - Development Manager
File CMVC Windows Directory Actions Help

```
Context: bering:/u/aixcase3/PortedGUI

<Parent>                  Directory
Enrollments.c             C Source
Enrollments.i             AIC Interface File
PortedGUI.op              Text
PortedGUI.prj             AIC Project File
```

Figure 138. Development Manager Recognizing AIC-Specific File Type Extensions

9.4.3 Modify the Action Pull-Down of Development Manager

Development Manager attaches different Action pull-down menus for files with different file type extensions. For example, the Actions pull-down for C source files includes a push button to start a compile, whereas the Actions pull-down for *makefile* files contains a push button to run **make**.

Development Manager reads the menu definitions along with the label strings and mnemonics for the push buttons from customization files in the `/usr/softbench/menus/Softdm` directory. These customization files exist for each language that is supported, and Development Manager uses the $LANG environment variable to determine which files are to be used. To change the menu definitions you can either change the `/usr/softbench/menus/Softdm/DM/Softdm/$LANG` and `/usr/softbench/menus/Softdm/DM/Softdm/MDF.m` files directly, or create a new directory in `/usr/softbench/menus/Softdm/ACTIONS` and add the entries to the new `$LANG` and `MDF.m` files in that directory.

In our environment the English locale `En_US` was used, and we wanted to have an AIC-specific Action pull-down for the AIC-specific file type extensions. As explained in 9.4.2, "Modify Extension-Based File-Typing of Development Manager" on page 214 we modified the extension file typing of SDE WorkBench/6000 and defined new menus to SDE WorkBench/6000. Refer to Figure 137 on page 215.

Follow these steps to add the new menu definitions:

1. Log in as *root* and create a new `UIBUILD` directory in the `/usr/softbench/menus/Softdm/ACTIONS` directory.
2. Create a new file called `En_US`. To have different label strings for different user locales, copy the translated contents of the file to a file with the proper locale name. The file contains the line as shown in Figure 139 on page 217.

```
Softdm*aicInterface.labelString: Actions
Softdm*aicInterface.mnemonic: A
Softdm*aicProject.labelString: Actions
Softdm*aicProject.mnemonic: A
Softdm*aicPalette.labelString: Actions
Softdm*aicPalette.mnemonic: A

Softdm*ACTIONS_UIBUILD_aicProject_aic.labelString:    AIC
Softdm*ACTIONS_UIBUILD_aicProject_aic.mnemonic:    A
Softdm*ACTIONS_UIBUILD_aicProject_edit.labelString:    Edit
Softdm*ACTIONS_UIBUILD_aicProject_edit.mnemonic:    E

Softdm*ACTIONS_UIBUILD_aicInterface_aic.labelString:    AIC
Softdm*ACTIONS_UIBUILD_aicInterface_aic.mnemonic:    A
Softdm*ACTIONS_UIBUILD_aicInterface_edit.labelString:    Edit
Softdm*ACTIONS_UIBUILD_aicInterface_edit.mnemonic:    E

Softdm*ACTIONS_UIBUILD_aicPalette_aic.labelString:    AIC
Softdm*ACTIONS_UIBUILD_aicPalette_aic.mnemonic:    A
Softdm*ACTIONS_UIBUILD_aicPalette_edit.labelString:    Edit
Softdm*ACTIONS_UIBUILD_aicPalette_edit.mnemonic:    E
```

Figure 139. Entries to the En_US File Used for Menu Definition

3. Create a new file called MDF.m. Under AIC 1.2, the file contains the lines shown in Figure 140.

```
Menu aicProject {
   ACTIONS_UIBUILD_aicProject_aic  f_exec \
            MenuMsgSelectFunc "UIBUILD LOAD-UIFILE"
   ACTIONS_UIBUILD_aicProject_edit f_exec \
            MenuMsgSelectFunc "EDIT WINDOW"
}

Menu aicInterface {
   ACTIONS_UIBUILD_aicInterface_aic  f_exec \
            MenuMsgSelectFunc "UIBUILD LOAD-UIFILE"
   ACTIONS_UIBUILD_aicInterface_edit f_exec \
            MenuMsgSelectFunc "EDIT WINDOW"
}

Menu aicPalette {
   ACTIONS_UIBUILD_aicPalette_aic  f_exec \
            MenuMsgSelectFunc "UIBUILD   LOAD-UIFILE"
   ACTIONS_UIBUILD_aicPalette_edit f_exec \
            MenuMsgSelectFunc "EDIT WINDOW"
}
```

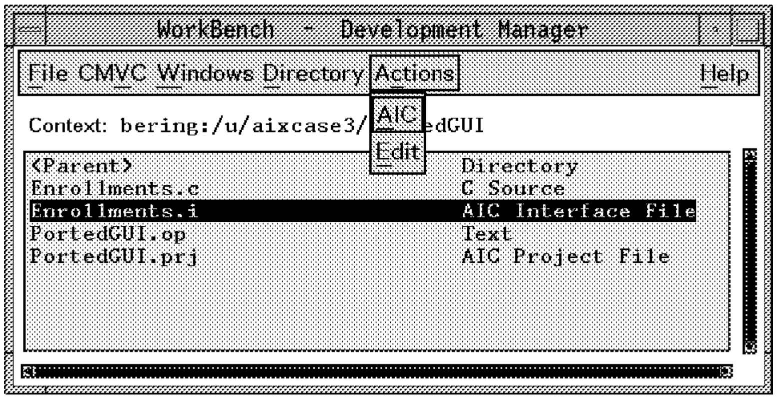
Figure 140. Entries to the MDF.m File Used for Menu Definition (AIC 1.2)

When using AIC releases previous to 1.2, the file contains the lines shown in Figure 141 on page 218.

```
Menu aicProject {
     ACTIONS_UIBUILD_aicProject_aic f_exec \
        MenuMsgObjectFunc "TERM NO-STDIO \
        %Context_Host% 2 %Context_Dir% 2 - - %Context_Host% 2 \
        aic %List% 1 %Select% %Warning% 1 %White_space% %Warning%"
     ACTIONS_UIBUILD_aicProject_edit f_exec \
        MenuMsgSelectFunc          "EDIT WINDOW"
}


Menu aicInterface {
     ACTIONS_UIBUILD_aicInterface_aic f_exec \
        MenuMsgObjectFunc "TERM NO-STDIO \
        %Context_Host% 2 %Context_Dir% 2 - - %Context_Host% 2 \
        aic %List% 1 %Select% %Warning% 1 %White_space% %Warning%"
     ACTIONS_UIBUILD_aicInterface_edit  f_exec \
        MenuMsgSelectFunc "EDIT WINDOW"
}


Menu aicPalette {
     ACTIONS_UIBUILD_aicPalette_aic  f_exec
        MenuMsgObjectFunc "TERM NO-STDIO \
        %Context_Host% 2 %Context_Dir% 2 - - %Context_Host% 2 \
        aic %List% 1 %Select% %Warning% 1 %White_space% %Warning%"
     ACTIONS_UIBUILD_aicPalette_edit  f_exec
        MenuMsgSelectFunc "EDIT WINDOW"

}
```

Figure 141. Entries to the MDF.m File for AIC Releases Previous to 1.2

The Action pull-down modifications are in effect the next time you start SDE WorkBench/6000. When you select an AIC interface, project, or palette file, and pull down the **ACTIONS** menu, the window looks like that shown in Figure 142 on page 219.


WorkBench - Development Manager
File CMVC Windows Directory Actions Help
AIC
Edit

Context: bering:/u/aixcase3/PortedGUI

```
<Parent>                          Directory
Enrollments.c                     C Source
Enrollments.i                     AIC Interface File
PortedGUI.op                      Text
PortedGUI.prj                     AIC Project File
```

Figure 142. Actions Pull-Down for AIC-Specific File Type Extensions

9.4.4 Modify the Default Action of Development Manager

Development Manager can assign different default actions to different file type extensions; so a double click triggers different actions for different file type extensions.

To set the default action for AIC interface, project, and palette files to invoke AIC directly for the selected file, follow these steps:

1. Either edit the `.Xdefaults` file in the *$HOME* directory

   or log in as root and edit the`/usr/softbench/menus/Softdm/ACTIONS/$LANG` file, where `$LANG` is the locale.

2. Add the lines shown in Figure 143 when using AIC 1.2.

```
  Softdm.actionPanelDoubleClick_aicInterface: MenuMsgSelectFunc \
     UIBUILD LOAD-UIFILE
  Softdm.actionPanelDoubleClick_aicProject: MenuMsgSelectFunc \
     UIBUILD LOAD-UIFILE
  Softdm.actionPanelDoubleClick_aicPalette: MenuMsgSelectFunc \
     UIBUILD LOAD-UIFILE
```

(AIC 1.2)

Figure 143. Modify the Default Action for AIC-Specific File Type Extensions

Or, add the lines shown in Figure 144 on page 220 when using releases previous to AIC 1.2.

```
   Softdm.
actionPaneDoubleClick_aicInterface: MenuMsgObjectFunc \
           TERM NO-STDIO \
           %Context_Host% 2 %Context_Dir% 2 - - %Context_Host% 2 \
           aic %List% 1 %Select% %Warning% 1 %White_space% %Warning%
   Softdm.actionPaneDoubleClick_aicProject:   MenuMsgObjectFunc \
           TERM NO-STDIO \
           %Context_Host% 2 %Context_Dir% 2 - - %Context_Host% 2 \
           aic %List% 1 %Select% %Warning% 1 %White_space% %Warning%
   Softdm.actionPaneDoubleClick_aicPalette:   MenuMsgObjectFunc \
           TERM NO-STDIO \
           %Context_Host% 2 %Context_Dir% 2 - - %Context_Host% 2 \
           aic %List% 1 %Select% %Warning% 1 %White_space% %Warning%
```

Figure 144. Modify the Default Action (AIC Releases Previous to 1.2)

These default action modifications are in effect the next time SDE WorkBench/6000 is started. When Development Manager displays an AIC interface, profile, or palette file, and you-double click that file, AIC starts and loads the selected file automatically.

9.4.5 Modify the Invocation of Other Tools from AIC

AIC 1.2 enables you to invoke other SDE WorkBench/6000 tools. For example, you can use Program Editor to edit AIC callbacks, *makefile* files, or resource properties, or use Program Builder for generating C code, or use CMVC for library control functions. In addition, AIC 1.2 responds to messages from other SDE WorkBench/6000 tools. For example, when you change the context in Development Manager and start AIC for a selected AIC project, palette, or interface file, the current directory as shown in AIC is also changed to the current context of Development Manager. To enable this behaviour, add the line shown in in Figure 145 to your `.Xdefaults` file. For more details, refer to the discussion of running AIC under SDE WorkBench/6000 in *AIXwindows Interface Composer Developer's Guide, Version 1.2*.

```
   Aic12*usingSoftbench: true
```

Figure 145. Enabling AIC to Use SDE WorkBench/6000 Tools

This setting becomes active the next time you start X, or explicitly enter the **xrdb .Xdefaults** command.

9.4.6 Integration Restrictions and Their Circumventions

We found that the AIC 1. 2 release we used had a restriction when AIC was integrated with SDE WorkBench/6000. The problem became obvious when the current directory set in AIC was in fact a remote file that was NFS-mounted to the system where AIC was executing. AIC would then fail to interface to other SDE WorkBench/6000 tools.

For example, invoking Program Builder from a current directory of`/nfs/bering/ad` (which is in fact remote and NFS-mounted) would result in an AIC error message. The error message would say that Program Builder could not change the current directory to `/nfs/bering/nfs/bering/ad`. For some reason, the remote path prefix `/ad/bering` got appended to the current directory name again.

In this case the circumvention we used was as follows. We created a directory `/ad` on the local system where AIC was executing, and linked that directory to `/nfs/bering/ad`. If we now change the current directory of AIC to `/ad`, AIC would then add the path name `/nfs/bering` to that name, and Program Builder and AIC would be able to access the same file.

AIC development has committed to correcting this restriction and providing the appropriate PTF for AIC.
Contact your IBM AIX service provider for details of the PTF order number and availability.

9.5 Tailoring AIC

You can customize AIC by modifying some of AIC's X11 resources. When you use AIC releases previous to 1.2 these modifications can be made either to the system-wide `/usr/lpp/aic/newconfig/app-defaults/Aic` resource file, or to your private `Aic` resource file that is addressed by the *XAPPLRESDIR* environment variable, or to your `.Xdefaults` resource file. When you run AIC 1.2, the corresponding file names are `/usr/lpp/aic12/newconfig/app-defaults/Aic12` and `Aic12`.

In our environment we wanted to make changes to the system-wide resource file, because all the members of our team had to use AIC in the same customized version. We set the following AIC resources for releases prior to AIC 1.2:

Aic.cflags	These are the flags AIC passes along to the interpreter.
Aic.includeFile	If set to True, AIC generates an include file for each AIC interface.
Aic.writeCCode	If set to Xt Code, AIC generates plain Xt calls rather than calls to the AIC Ux libraries.

The discussion of basic concepts of AIC in *User Interface Programming Concepts: AIXwindows Interface Composer, Volume 2*, gives more details on using the AIC resource file.

For AIC 1.2 we set the following options:

Aic12.cflags	These are the flags AIC passes along to the interpreter.
Aic12.includeFile	If set to True, AIC generates an include file for each AIC interface.
Aic12.cgUxLib	If set to false, AIC generates plain Xt calls rather than calls to the AIC Ux libraries.

These resources are described in *Installing and Configuring AIC, Version 1.2*.

In our project environment we wanted AIC to load the include files from our project include directory both during interpretation time and for compilations, we wanted to have an include file generated for each AIC interface, and we wanted to generate plain Xt code to achieve the highest degree of portability.

The modified resource file contains the lines shown in Figure 146 on page 222 for releases previous to AIC 1.2:

```
  Aic.
cflags:       -I/ad/projectA_proto/include
  Aic.includeFile:  true
  Aic.writeCCode:   XtCode
```

Figure 146. Modified AIC Resource File Previous to AIC 1.2

The modified resource file for AIC 1.2 contains the lines shown in Figure 147.

```
  Aic12.
cflags:       -I/ad/projectA_proto/include
  Aic12.includeFile:  true
  Aic12.cgUxLib:      false
```

Figure 147. Modified AIC Resource File for AIC 1.2

Chapter 10. Integrating User-Developed Utilities into SDE WorkBench/6000

This chapter describes how additional tools, utilities, or applications can be integrated with SDE WorkBench/6000. Integration in this sense means that the new tool would be sensitive to messages that are exchanged to and from the message bus, and would thus be able to interface with other tools by sending messages or reacting to messages from others. This process is called *encapsulation* of a tool. We show the basic principles of how this can be done and explain the various levels of encapsulation with or without modifications to the existing code of the tool.

10.1 Concepts

This section shows the rationale for encapsulating a tool and shows the benefits of doing so. It also discusses the different steps of the encapsulation process.

When encapsulating a tool for use with SDE WorkBench/6000, you are required to install an additional IBM LPP, SDE Integrator/6000.

10.1.1 Why Encapsulate a Tool?

When you have a set of tools to be used during your development process, these tools do not stand beside one another. Rather, they have to interact either directly (if the logical output of one tool is the input to another one), or they have to be used beside one another (if one tool invokes another and the called tool provides the output to the calling one). In addition, sometimes several tools have to be used in a coordinated fashion, like tools to support the edit, compile, and debug cycle.

One solution to manage this kind of interaction is to have the user do this. A developer would then have to take care of invoking one tool after the other, and would have to provide the required input manually. As an example, a developer would have to manually invoke the editor after an unsuccessful compilation (of course this does not happen with good programmers...). However, in an integrated environment the tools would be linked together so whenever a tool needs help from another tool they will communicate with one another accordingly.

This idea is exactly the concept of the message system of SDE WorkBench/6000. However, this system must also support a way to enable customers or vendors to plug in additional tools as they appear or are required along the development process. Integration implies using tools across the network, and it may also imply using a consistent end user interface.

When a requirement for a new tool arises, a development shop must be able to integrate this new tool with the existing set of tools.

For example consider a tool to support some kind of additional project control by gathering statistical data for source code like counting the number of executable source line instructions (we do not imply that this is a real requirement, but just use this as an example); so an integrated use of the tool would imply that each time a source file is modified in the library system, new statistics would be generated. So the counting tool would be encapsulated such that it would listen for Checkin messages from the Configuration Management tool. Each time such a message is received from the bus, the counting tool would start automatically and update the corresponding count. In addition, the count tool could send a START message to a TERM tool in order to display the generated number on a screen. The tool would also be added to the ACTIONS pull-down of Development Manager; so once the new option is selected for a given file, the tool would also be started.

10.1.2 Steps of an Encapsulation

You have to follow the steps as described in this section when you intend to encapsulate a tool.

1. Define the messages to which the tool will listen and react, and define the actions your tool is going to perform in these cases.
2. Define the messages that the tool is going to send out, and define which other tool is going to receive these messages.
3. If you want to integrate an existing tool with a standard input/output user interface or no user interface at all, you can implement a wrapper encapsulation without touching the existing code.
4. If you want to integrate an existing tool with an existing event driven user interface, you may need to merge the event loops of your existing code with the new one as required by the encapsulation.
5. If you want to integrate and develop a new tool, you need to decide whether to use the SDE WorkBench/6000 development tools for user interface and message- and help-text handling.
6. Choose the implementation language for the encapsulation. SDE Integrator/6000 supports EDL (a high level meta language), C, or C^{++}.
7. Implement the encapsulation and create the corresponding binary executable file.
8. Install the required programs and data files in corresponding directories.
9. Customize SDE WorkBench/6000 to reflect the addition of the tool by modifying the softtypes and softinit files or the menu definitions.

Have a look at *IBM AIX SDE Integrator/6000 Programmer's Guide* for more details about how to implement a tool encapsulation.

10.2 Encapsulation Approaches

This section describes the various encapsulation approaches that are possible. It explains the various types of events to which a tool reacts, and explains the different ways of encapsulating tools with or without source code modifications. Finally an example is described that shows the interaction between new tools and existing ones.

10.2.1 Events a Tool May React to

SDE WorkBench/6000 differentiates between the following event types:

Application Events	These are events that are triggered when the tool sends output to stdout or stderr

User Events	These are events that are triggered when a user interacts with the encapsulated application and triggers a X event (such as clicking a button)
Message Events	These are events triggered when the encapsulated tool receives a message from the message server.
System Events	These are system signals from the operating system.

The major tasks during design time of the encapsulation is to decide, how (if at all) the encapsulated application should react if one of the events as described above is received. This reaction can either be the invocation of some application function, or sending a message to another tool, or both. Of course, events can also just be ignored by the application.

The main effort when implementing an encapsulation is to implement the programs that handle those events that are important for the application. Once the event was handled by the program, the control is simply returned to the main event loop until another event occurs. In that aspect, writing an encapsulation is very similar to implementing an AIXwindows application.

10.2.2 Encapsulations with No Code Modifications

The easiest way of encapsulating an application is to build some kind of a shell around the existing application. This shell defines the events the tool would react to, and would call the existing application for corresponding events. The existing code of the application is not touched; instead the encapsulation is like an additional wrapper layer around the application itself. As explained in 10.1.2, "Steps of an Encapsulation" on page 224 the encapsulation can be written in EDL, C^{++}, or C.

This wrapper approach for encapsulation can be chosen if the application uses only standard input and output rather than a graphical user interface, or if the application does not have a user interface at all.

In this case the encapsulation code would define the events the tool reacts to and would code the calls to invoke the original application executable in the corresponding event definition.

10.2.3 Encapsulations with Code Modifications

If the application to be encapsulated is already based on an event-driven user interface system, such as AIXwindows, there is more effort involved. In this case, it may be necessary to merge the event loops of the original application with the event loop of the encapsulation so when certain events occur in the *old* application actions for the encapsulation are triggered.

This effort is more sophisticated and is described in great detail in *IBM AIX SDE Integrator/6000 Programmer's Guide*.

10.3 Integrated Utility Example

A common requirement in a development organization environment is to count the number of source lines for modules. This number can be used for planning purposes, such as estimating project schedules, upcoming test effort, or resource assignments for maintenance. In our project we decided to use an IBM internal tool that counts the number of source lines for a given source file. This section does not intend to describe the algorithm used to calculate that number. Rather than that, we want to show how this IBM internal tool was used as a base, and how we wrote an encapsulation around the tool. This encapsulated tool was then integrated with Development Manager so the tool could be started from the Actions pull-down of Development Manager once a source file has been selected, or from the Tool Manager Start window.

10.3.1 The Original Counting Tool

As mentioned the encapsulation was based on an existing program, for which we did not have the source code. The basic tool was program that could be invoked from the command line and would accept a number of options along with parameters. The name of the file to be counted was prompted from standard input, and the output result was written to standard output. Additional flags could define whether to produce verbose output or not, and what kind of language definition profile was to be used.

To invoke the tool without having any more input required, we could use the `echo` command along with the file name of the source file to be counted, and pipe the output to the counting tool.

10.3.2 Design of the New Tool

We decided to write a tool that would be a member of the *COUNT* tool class. The tool would be invoked either from the Tool Manager Start window, or from the Actions pull-down of Development Manager once a source file was selected. The message sent to the *COUNT* tool would be *RUN-COUNT*, and the definition of the tool in the `.softinit` file would map the name of the tool to the proper invocation command of the encapsulated tool.

The tool itself would show a window along with the standard SDE WorkBench/6000 menu bar entries for *File* and *Help*. The window would show the name of the file to be counted and a scrolled text area where the output from the counting process would be displayed.

The window would look like Figure 148 on page 227.

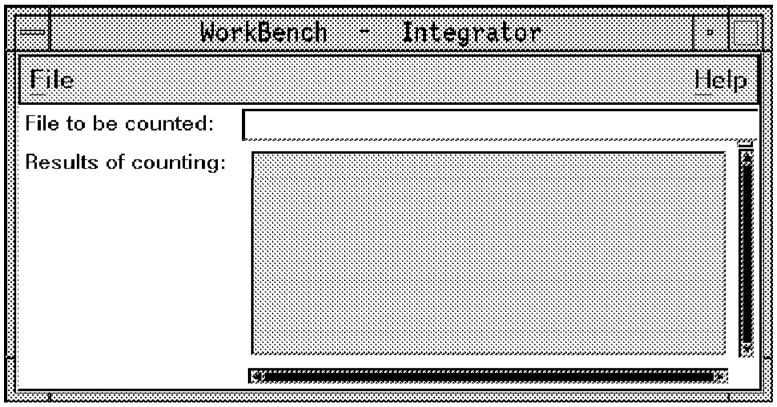


Figure 148. User Interface of the New Tool

10.3.3 Implementation of the New Tool

We decided to implement the new tool using the C language. Figure 149 on page 228 shows the source code of the encapsulated tool.

```
 #include <edl/edl_names.
h>

/* global definitions for the objects as part of the COUNT tool window */
object top, panel, pane2, number, file;
/* global definitions for the name of the file to be counted           */
string filepath;

void run_count_first_time(void)
{
  /* called, if the main program is invoked from Execution manager    */
  string part1, command, result;

  busy(number);
  busy(file);

  /* build the command to be executed                                 */
  part1 = string_concat("echo ", filepath);
  command = string_concat(part1, "|/home/aixcase3/slocc/slocc -p /home/aixcase3/slocc/slocc.pro -d -vq");
  /* issue command and put output out to the screen                   */
  result = system(command, NULL);
  append(number, result);

  unbusy(number);
  unbusy(file);
}


void run_count(void)
{
  /* called, if the RUN-COUNT message is received                     */
  string part1, command;

  busy(number);
  busy(file);

  clear(file, 0);
  clear(number, 0);

  /* get the name of the file to be counted from message parameters   */
  filepath = make_filename(message_host(), message_directory(), message_file(),
              False);
  append(file, filepath);
  /* issue command and put output out to the screen                   */
  append_to(number, False);
  part1 = string_concat("echo ", filepath);
  command = string_concat(part1, "|/home/aixcase3/slocc/slocc -p /home/aixcase3/slocc/slocc.pro -d -vq");
  send_command(command, True );                                     1

  unbusy(number);
  unbusy(file);
  send_message(Notify, 0, "RUN-COUNT", message_id(), 0, 0);
}


/* Function to announce operations that COUNT tool supports           */
void establish_message_interface(void)
{
  /* post to pay attention to the RUN-COUNT message                   */
  add_event(make_event(Message,                                     2
                      make_message_pattern(Request, "COUNT",
"RUN-COUNT",
                                          0,0,0,0,0,0,0),
                      run_count
                     ));


}
```

Figure 149 (Part 1 of 2). The Implemented Source Code for the Encapsulation

```
main (int argc, char *argv[])
{
  attribute temp;
  attribute att;

  /* Init the tool encapsulation                                         */
  init(&argc, argv);
  tool_class("COUNT");                                              3


  /* Define the window layout                                   */  4
  top   = make_manager(NULL, Toplevel, "Count");

  pane1 = make_manager(top,  Pane, "CountPane1");
  make_object(pane1, "contextLabel1", Label, "File to be counted:   ",
READONLY, 0);
  att = merge_attribute(XOFFSET(0), SINGLELINE);
  temp = merge_attribute(att, READONLY) ;
  att = merge_attribute(temp, COLUMNS(50) );
  file = make_object(pane1, "file", Edit, " ", att, 0);

  pane2 = make_manager(top,  Pane, "CountPane2");
  make_object(pane2, "contextLabel2", Label, "Results of counting:  ",
READONLY, 0);
  att =  merge_attribute(ROWS(8) , XOFFSET(0)) ;
  temp = merge_attribute(att, READONLY);
  number = make_object(pane2,
```

Note: umber

```
, Edit, " ", temp, 0);

  /* display the window                                                  */
  display(top, 0);

  /* define the messages the tool listens to                             */
  establish_message_interface();

  clear(file, 0);

  if (strcmp(argv[5], "RUN-COUNT")==0)
{                         5
    /* invocation done from Actions pull-down                            */
    if (strcmp(argv[7],"/")==0)
      filepath = print_to_string("/%s", argv[8]);
    else
      filepath = print_to_string("%s/%s", argv[7],
argv[8]);
    append(file, filepath);
    run_count_first_time();
    send_message(Notify, 0, "RUN-COUNT", message_id(), 0, 0);
  }  else {                                                         6
    /* called from the Tool Start window                                 */
    /* just put the context directory info the 'File name' field        */
    filepath = print_to_string("%s", argv[7]);
    append(file, filepath);
    send_message(Notify, 0, "START", message_id(), 0, 0);
 }

  /* and start the event loop with a sub process                         */
  start("sh", 0,0,0);
  /* cleanup (called once the event loop was broken)                     */
  finish();
}
```

Figure 149 (Part 2 of 2). The Implemented Source Code for the Encapsulation

The encapsulation as shown in Figure 149 on page 228 defines the new tool class *COUNT* (**3**) and defines the window layout at the beginning of the main program (**4**). The event monitored is the *RUN-COUNT* message (**2**), and the corresponding routine to be triggered is the *run_count* function with the final call to the existing IBM internal tool (**1**). The main program, if invoked for a *START* request, would not start counting but just display the window (**6**). This would be the case if we invoked *COUNT* from the Tool Manager Start window rather than from the Actions pull-down of Development Manager. To find out how *COUNT* was called, the parameters as passed to the program were used. As defined in the `.softinit` file, shown in Figure 151 on page 230, the parameter being passed to *COUNT* is the entire message string. The fifth parameter of the message (which is the sixth command line argument, as the first one is the name of the called program itself) is either the string *START* or *RUN-COUNT*. Thus the variable *argv[5]* can be used to determine whether *COUNT* was invoked from the Tool

Manager Start window or not. This is shown at the lines indicated by **5** in Figure 149 on page 228.

The source file as implemented was then compiled using the SDE Integrator/6000 include files and linked using the corresponding libraries. The command used to do this is shown in Figure 150.

```
 cc -I/usr/softbench/include \
   -L/usr/softbench/lib     \
   -lencapinit              \
   -lencap                  \
   -lXe                     \
   -lXeTest                 \
   -lbms                    \
   -lXm                     \
   -lXt                     \
   -lX11                    \
   -lsoftlib -o kloc kloc.
c
```

Figure 150. Compile Command to Build the Encapsulation

After we compile and link, the system would build the encapsulated executable `kloc` file.

10.3.4 Integration of the New Tool with SDE WorkBench/6000

Once the encapsulation was built, we had to integrate the new tool with SDE WorkBench/6000. This included defining the new *COUNT* entry in either the system wide `softinit` fil or in the user's private `.softinit` file. For test purposes we decided to modify the private `.softinit` file, and we added the line as shown in Figure 151.

```
COUNT   TOOL HOST * %Host% /home/aixcase3/ToolIntegration/kloc %Message%
```

Figure 151. Softinit Entry to Define the Class of the Encapsulated Tool

We then modified the Actions pull-down of Development Manager. We logged in as *root* and edited the file `/usr/softbench/menus/Softdm/DM/Softdm/MDF.m`. We added the line indicated by **1** in Figure 152 of the definition of the *source* pull-down menu.

```
Menu source {
        DM_source_fileedit            f_exec MenuMsgSelectFunc "EDIT WINDOW"
        DM_source_filecompile         f_exec MenuMsgSelectFunc "BUILD COMPILE-FILE - - -"
        DM_source_filelistfuncs       f_exec MenuMsgCurrentDirFunc "STATIC SHOW-FUNCTIONS"
        DM_source_fileprint           f_exec fileprint       NULL
        DM_source_filecount           f_exec MenuMsgSelectFunc "COUNT RUN-COUNT"        1
}
```

Figure 152. Modifying the Actions Menu Definition for the New Tool

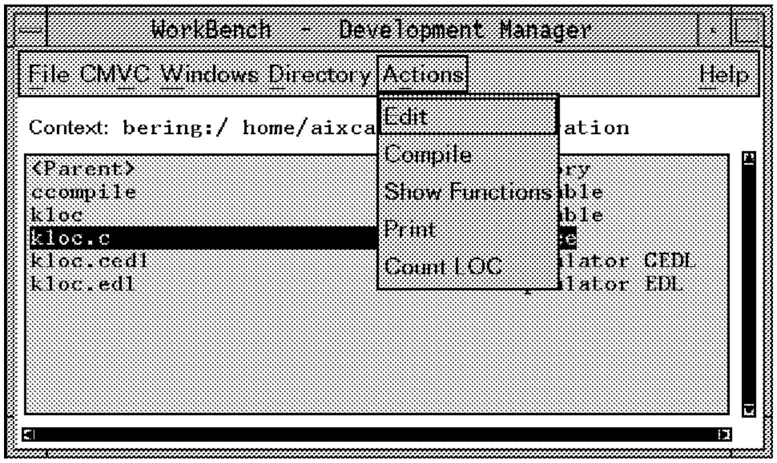
We then added the label text definition of the new push button of the menu. To do this, we edited the file `/usr/softbench/menus/Softdm/DM/Softdm/C`. We added the line shown in Figure 153.

```
Softdm*DM_source_filecount.labelString:         Count LOC
```

Figure 153. Modifying the Labels of the Actions Pull-Down for the New Tool

When we started SDE WorkBench/6000 the next time, Development Manager would show the pull-down as shown in Figure 154 once a C source file is selected.

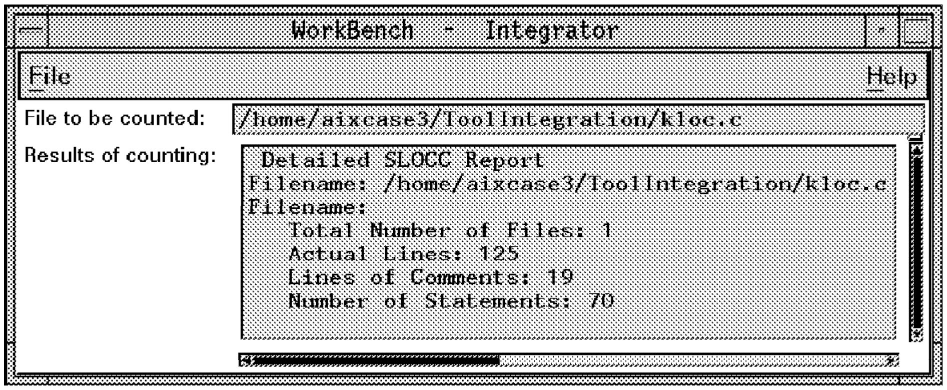
* 4i*3i


WorkBench - Development Manager
File CMVC Windows Directory Actions Help
Edit
Compile
Show Functions
Print
Count LOC

```
Context: bering:/ home/aixca          ation
<Parent>                               ry
ccompile                               ble
kloc                                   ble
kloc.c
kloc.cedl                              lator CEDL
kloc.edl                               lator EDL
```

Figure 154. Modified Actions Pull-Down for Encapsulated Tool

Once **Count LOC** is selected from the Actions menu, the window as shown in Figure 155 would be displayed when the tool was invoked for the source of the tool encapsulation program itself.



File to be counted: /home/aixcase3/ToolIntegration/kloc.c

Results of counting:

```
 Detailed SLOCC Report
Filename: /home/aixcase3/ToolIntegration/kloc.c
Filename:
    Total Number of Files: 1
    Actual Lines: 125
    Lines of Comments: 19
    Number of Statements: 70
```

Figure 155. The Window after Invoking the Encapsulated Tool

Appendix A. Third-Party SDE Integrated AD Tools

IBM provides various products integrated with the SDE WorkBench/6000, most of which were described in this book. Here is a list of these:

Table 5. List of IBM Products in the SDE Environment

IBM product name	**Product number**
SDE WorkBench/6000	5696-524
SDE Integrator/6000	5696-523
C^{++} POWERbench	5696-733
COBOL POWERbench	5696-761
FORTRAN POWERbench	5696-551
AIXwindows Interface Composer	5756-027
CMVC/6000	5765-207
CMVC for Sun	5622-063
CMVC for Solaris Systems	5765-397
CMVC for HP	5765-202

Following is a list of tools supplied by various other software vendors and also integrated with SDE WorkBench/6000. These products supplement the tools as supplied by IBM. This list does not imply a recommendation to use one of these tools, nor is it considered to be complete. To the best of our knowledge, it represents the set of tools that are integrated with SDE Workbench/6000 at control integration level at the time of writing this book .

Table 6 (Page 1 of 2). List of Vendor Products Integrated with SDE WorkBench/6000

Product name	**Vendor name**
SEDIT**	BENAROYA
Teamwork**	Cadre
Process WEAVER**	CAP Gemini Sogeti (CGS)
CaseWare/CM**	CaseWare
Software Backplane**	CRI (Atherton)
INFORMIX/4GL for ToolBus**	Informix
StP-ISE**	IDE
Interleaf 5**	Interleaf
PVCS CB and VM**	INTERSOLV
KeyOne**	LPS
COBOL, COBOL with Toolbox**	Micro Focus
Innovator**	MID
SMARTsystem**	PROCASE
ROSE**	Rational
Systemator**	Sysdeco

Table 6 (Page 2 of 2). List of Vendor Products Integrated with SDE WorkBench/6000

Product name	**Vendor name**
SDT**	TeleLOGIC
Emacs editors**	Unipress Software
ASE/ASA, AGE/GEODE, LOGISCOPE**	VERILOG
VADS**	Verdix
ViSTA**	VERITAS Software
View, Vutil**	Versant
UIM/X (IBM logo: AIC)	Visual Edge
I-CASE**	Westmount

In addition to the list above, several vendors had announced support for integration with SDE WorkBench/6000 in their products and was in the process of implementing it.

Table 7 contains a list of these products, many of which are already available in beta version. Check the corresponding vendor for details about the availability date.

Table 7. List of Vendor Products Integrated with SDE WorkBench/6000 (Planned or Announced)

Product name	**Vendor name**
AdaWorld**	Alsys
TRITON Tools**	Baan International B.V.
Prolog by BIM**	BIM
ProMod-PLUS**	CAP debis GEI
DECADE**	Delaware Computing
APPLIDUAL**	DUAL
ENFIN/3**	Easel
FrameMaker**	Frame Technology
XRunner**	Mercury Interactive Corp., Israel
Micro Focus Dialog System**	Micro Focus
Open Interface**	Neuron Data
OBJECTORY**	Objective Systems (OS)
REFINE/FORTRAN**	Reasoning Systems
HyperWork**	PBS
Objecteering**	Softeam
Software TestWorks (STW)**	Software Research
Uniface WB**	UnifAce
VIEWS/VSF**	Virtual Software Factory

Appendix B. Sample Panel Test Program

Figure 156 on page 236 shows a C program that was written to test the MVS ISPF panel definitions with the IBM Internal Tool **AIXISPF**. This tool enables the testing of the ISPF panels written for MVS on the AIX development platform. This tool made it possible to test not only the COBOL and DB2 code, but also the MVS user interface itself, all on AIX. This tool consists of a library of C language calls that the developer uses in writing a panel test program.

```
/*********************************************************************/
/*    testpan.c                                                      */
/*    Program to test ISPF panels on AIX.                            */
/*    Makes use of the IBM Internal Tool AIXISPF.                    */
/*    Author: Leif Trulsson, IBM Sweden                              */
/*                                                                   */
/*********************************************************************/
#include <stdio.h>
#define PANEL_LIB "./panels"

/* these global variables are vdefined to match ISPF variables. */
char ZCMD[255];
char ERRMSG[70];
char PFK[4];
char ACT[2];
char ARG[18];
char PFNAME[13];
char PLNAME[18];
char PSTREET[26];
char PZIPCODE[5];
char PCITY[20];
char PCUSTNO[10];
char PREFNO[10];
char PMAILID[2];
char PSRC[3&rbracket;
char PACTDATE[6];
char PADDRCHG[6];
char PPROFIT[8];
char PDCODE[1];
char PCOLLC[1];
char PPAYDATE[6];
char PAYDAT1[6];
char PAYDAT2[6&4bracket.;
char PAYDAT3[6];
char PAYDAT4[6];
char PAYDAT5[6];
char PAMOUNT[8];
char AMOUNT1[8];
char AMOUNT2[8];
char AMOUNT3[8];
char AMOUNT4[8];
char AMOUNT5[8];
char VAL[2];

struct rad {
      char nr[2];
      char fill1;
      char custno[10];
      char fill2;
      char firstname[13];
      char fill3;
      char lastname[18];
      char fill4;
      char street[26];
      char fill5;
      char zipcode[5]“;
      char null;
   } ;
```

Figure 156 (Part 1 of 5). ISPF Panel Test Program

```
struct rad R1;
struct rad R2;
struct rad R3;
struct rad R4;
struct rad R5;
struct rad R6;
struct rad R7;
struct rad R8;
struct rad R9;
struct rad R10;
struct rad R11;
struct rad R12;
struct rad R13;
struct rad R14;
struct rad R15;



/* link variables to ISPF variables */
int vdef_vars()
{
  char com[255];
  int rc;

  ZCMD[sizeof(ZCMD)-1] = '\0';
  rc = isplink("VDEFINE", "ZCMD", ZCMD, "CHAR", sizeof(ZCMD)-1);
  if (rc) return rc;
  ERRMSG[sizeof(ERRMSG)-1] = '\0';
  rc = isplink("VDEFINE", "ERRMSG", ERRMSG, "CHAR", sizeof(ERRMSG)-1);
  if (rc) return rc;
  PFK[sizeof(PFK)-1] = '\0';
  rc = isplink("VDEFINE", "PFK", PFK, "CHAR", sizeof(PFK)-1);
  if (rc) return rc;
  ACT[sizeof(ACT)-1] = '\0';
  rc = isplink("VDEFINE", "ACT", ACT, "CHAR", sizeof(ACT)-1);
  if (rc) return rc;
  ARG[sizeof(ARG)-1] = '\0';
  rc = isplink("VDEFINE", "ARG", ARG, "CHAR", sizeof(ARG)-1);
  if (rc) return rc;
  PFNAME[sizeof(PFNAME)-1] = '\0';
  rc = isplink("VDEFINE", "PFNAME", PFNAME, "CHAR", sizeof(PFNAME)-1);
  if (rc) return rc;
  PLNAME[sizeof(PLNAME)-1] = '\0';
  rc = isplink("VDEFINE", "PLNAME", PLNAME, "CHAR", sizeof(PLNAME)-1);
  if (rc) return rc;
  PSTREET[sizeof(PSTREET)-1] = '\0';
  rc = isplink("VDEFINE", "PSTREET", PSTREET, "CHAR", sizeof(PSTREET)-1);
  if (rc) return rc;
  PZIPCODE[sizeof(PZIPCODE)-1] = '\0';
  rc = isplink("VDEFINE", "PZIPCODE", PZIPCODE, "CHAR", sizeof(PZIPCODE)-1);
  if (rc) return rc;
  PCITY[sizeof(PCITY)-1] = '\0';
  rc = isplink("VDEFINE", "PCITY", PCITY, "CHAR", sizeof(PCITY)-1);
  if (rc) return rc;
  PCUSTNO[sizeof(PCUSTNO)-1] = '\0';
  rc = isplink("VDEFINE", "PCUSTNO", PCUSTNO, "CHAR", sizeof(PCUSTNO)-1);
  if (rc) return rc;
  PREFNO[sizeof(PREFNO)-1] = '\0';
  rc = isplink("VDEFINE", "PREFNO", PREFNO, "CHAR", sizeof(PREFNO)-1);
  if (rc) return rc;
  PMAILID[sizeof(PMAILID)-1] = '\0';
  rc = isplink("VDEFINE", "PMAILID", PMAILID, "CHAR", sizeof(PMAILID)-1);
  if (rc) return rc;
  PSRC[sizeof(PSRC)-1] = '\0';
  rc = isplink("VDEFINE", "PSRC", PSRC, "CHAR", sizeof(PSRC)-1);
```

Figure 156 (Part 2 of 5). ISPF Panel Test Program

```
    if (rc) return rc;
    PACTDATE[sizeof(PACTDATE)-1] = '\0';
    rc = isplink("VDEFINE", "PACTDATE", PACTDATE, "CHAR", sizeof(PACTDATE)-1);
    if (rc) return rc;
    PADDRCHG[sizeof(PADDRCHG)-1] = '\0';
    rc = isplink("VDEFINE", "PADDRCHG", PADDRCHG, "CHAR", sizeof(PADDRCHG)-1);
    if (rc) return rc;
    PPROFIT[sizeof(PPROFIT)-1] = '\0';
    rc = isplink("VDEFINE", "PPROFIT", PPROFIT, "CHAR", sizeof(PPROFIT)-1);
    if (rc) return rc;
    PDCODE[sizeof(PDCODE)-1] = '\0';
    rc = isplink("VDEFINE", "PDCODE", PDCODE, "CHAR", sizeof(PDCODE)-1);
    if (rc) return rc;
    PCOLLC[sizeof(PCOLLC)-1] = '\0';
    rc = isplink("VDEFINE", "PCOLC", PCOLLC, "CHAR", sizeof(PCOLLC)-1);
    if (rc) return rc;
    PPAYDATE[sizeof(PPAYDATE)-1] = '\0';
    rc = isplink("VDEFINE", "PPAYDATE", PPAYDATE, "CHAR", sizeof(PPAYDATE)-1);
    if (rc) return rc;
    PAYDAT1[sizeof(PAYDAT1)-1] = '\0';
    rc = isplink("VDEFINE", "PAYDAT1", PAYDAT1, "CHAR", sizeof(PAYDAT1)-1);
    if (rc) return rc;
    PAYDAT2[sizeof(PAYDAT2)-1] = '\0';
    rc = isplink("VDEFINE", "PAYDAT2", PAYDAT2, "CHAR", sizeof(PAYDAT2)-1);
    if (rc) return rc;
    PAYDAT3[sizeof(PAYDAT3)-1] = '\0';
    rc = isplink("VDEFINE", "PAYDAT3", PAYDAT3, "CHAR", sizeof(PAYDAT3)-1);
    if (rc) return rc;
    PAYDAT4[sizeof(PAYDAT4)-1] = '\0';
    rc = isplink("VDEFINE", "PAYDAT4", PAYDAT4, "CHAR", sizeof(PAYDAT4)-1);
    if (rc) return rc;
    PAYDAT5[sizeof(PAYDAT5)-1] = '\0';
    rc = isplink("VDEFINE", "PAYDAT5", PAYDAT5, "CHAR", sizeof(PAYDAT5)-1);
    if (rc) return rc;
    PAMOUNT[sizeof(PAMOUNT)-1] = '\0';
    rc = isplink("VDEFINE", "PAMOUNT", PAMOUNT, "CHAR", sizeof(PAMOUNT)-1);
    if (rc) return rc;
    AMOUNT1[sizeof(AMOUNT1)-1] = '\0';
    rc = isplink("VDEFINE", "AMOUNT1", AMOUNT1, "CHAR", sizeof(AMOUNT1)-1);
    if (rc) return rc;
    AMOUNT2[sizeof(AMOUNT2)-1] = '\0';
    rc = isplink("VDEFINE", "AMOUNT2", AMOUNT2, "CHAR", sizeof(AMOUNT2)-1);
    if (rc) return rc;
    AMOUNT3[sizeof(AMOUNT3)-1] = '\0';
    rc = isplink("VDEFINE", "AMOUNT3", AMOUNT3, "CHAR", sizeof(AMOUNT3)-1);
    if (rc) return rc;
    AMOUNT4[sizeof(AMOUNT4)-1] = '\0';
    rc = isplink("VDEFINE", "AMOUNT4", AMOUNT4, "CHAR", sizeof(AMOUNT4)-1);
    if (rc) return rc;
    AMOUNT5[sizeof(AMOUNT5)-1] = '\0';
    rc = isplink("VDEFINE", "AMOUNT5", AMOUNT5, "CHAR", sizeof(AMOUNT5)-1);
    if (rc) return rc;
    VAL[sizeof(VAL)-1] = '\0';
    rc = isplink("VDEFINE", "VAL", VAL, "CHAR", sizeof(VAL)-1);
    if (rc) return rc;
    return 0;
  }
```

Figure 156 (Part 3 of 5). ISPF Panel Test Program

```
/* Process our panels                                     */
int primary_cmd()
{
  char com[255];
  int rc;
  char c;

    do {                                 /* Do unitl PF3 is pressed     */

       strcpy(com, "VPUT (ACT ARG ERRMSG) SHARED");
       rc = ispexec(strlen(com), com);

       strcpy(com, "DISPLAY PANEL (IBMOU001)");
       rc = ispexec(strlen(com), com);
       if (rc == 8)
          break;

       ERRMSGﬄ0“ = '\0';

       strcpy(com, "VGET (ACT ARG) SHARED");
       rc = ispexec(strlen(com), com);

       if (!strcmp(ACT, "01"))
       {
          ACT[0] = '\0';
          strcpy(com, "VPUT (PFNAME PLNAME PSTREET PZIPCODE PCITY PCUSTNO PREFNO PMAILID PSRC) SHARED");
          rc = ispexec(strlen(com), com);
          do {
            strcpy(com, "DISPLAY PANEL (IBMOU002)");
            rc = ispexec(strlen(com), com);
          } while (rc != 8);

          ERRMSG[0&rbraket. = '\0';
          strcpy(com, "VGET (PFNAME PLNAME PSTREET PZIPCODE PCITY PCUSTNO PREFNO PMAILID PSRC) SHARED");
          rc = ispexec(strlen(com), com);
          rc = 0;
       }
       if (!strcmp(ACT, "23"))
       {
          do {
          ACTﬄ0“ = '\0';
            strcpy(com, "DISPLAY PANEL (IBMOU003)");
            rc = ispexec(strlen(com), com);
          } while (rc != 8);
          ERRMSG[0] = '\0';
          rc = 0;
       }
       if (!strcmp(ACT, "60"))
       {
          do {
          ACT[0] = '\0';
            strcpy(com, "DISPLAY PANEL (IBMOU004)");
            rc = ispexec(strlen(com), com);
          } while (rc != 8);
          ERRMSGYlbracket.0] = '\0';
          rc = 0;
       }
       if (!strcmp(ACT, "99"))
       {
          do {
            ACT[0] = "\0";
            strcpy(com, "DISPLAY PANEL (IBMOU005)");
            rc = ispexec(strlen(com), com);
          } while (rc != 8);
          ERRMSG[0] = '\0';
          rc = 0;
       }

    } while (rc != 8);
```

Figure 156 (Part 4 of 5). ISPF Panel Test Program

```
    if (!strcmp(ZCMD, "END")) {          /* Clean up                   */
    }
    strcpy(com, "CONTROL DISPLAY RESTORE");
    rc = ispexec(strlen(com), com);
}

/* Take care of any Errors                              */
void errexit (int rc, char *msg)
{
  printf("demo1: error: command returned nonzero code %d\n",rc);
  printf("demo1: command was: %s\n",msg);

  exit_services();
  exit (1);
}

main(int argc, char **argv)
{
  char com[255];
  int rc;

  /* start ISPF services. */
  init_services();

  /* vdefine variables */
  if (rc = vdef_vars()) {
    fprintf(stderr, "vdefine: returned %d\n", rc);
    exit(1);
  }

  /* define libraries */
  sprintf(com, "LIBDEF ISPPLIB FILE ID(%s)", PANEL_LIB);
  rc = ispexec(strlen(com), com);


  rc = 0;
  primary_cmd();

  /* make sure to call this before exiting */
  exit_services();
}
```

Figure 156 (Part 5 of 5). ISPF Panel Test Program

Appendix C. Panel Definitions

To be able to test the MVS releases on AIX without the limitation of the ISPF panel interface, we decided to write a character-based screen handler for AIX. This character-based screen handler uses the Micro Focus COBOL SPECIAL-NAMES and SCREEN-SECTIONs. Figure 157 shows the new panel definitions.

```
      * PANEL VARIABLES
       77  LASTCC                 PIC 9(4).
       01  EXIT-FLAG              PIC 9(2) COMP-X VALUE 0.
       01  ERRMSG                 PIC X(70) VALUE SPACE.
       01  PFK                    PIC X(4) VALUE SPACE.
       01  ACT                    PIC X(2) VALUE SPACE.
       01  ARG                    PIC X(18) VALUE SPACE.
       01  NARG REDEFINES ARG.
           05   CNUM              PIC 9(10).
           05   FILLER            PIC X(8).
       01  PFNAME                 PIC X(13) VALUE SPACE.
       01  PLNAME                 PIC X(18) VALUE SPACE.
       01  PSTREET                PIC X(26) VALUE SPACE.
       01  PZIPCODE               PIC X(5) VALUE SPACE.
       01  NZIPCODE REDEFINES PZIPCODE   PIC 9(5).
       01  PCITY                  PIC X(20) VALUE SPACE.
       01  PCUSTNO                PIC X(10) VALUE SPACE.
       01  NCUSTNO REDEFINES PCUSTNO     PIC 9(10).
       01  PREFNO                 PIC X(10) VALUE SPACE.
       01  NREFNO  REDEFINES PREFNO      PIC 9(10).
       01  PMAILID                PIC X(2) VALUE SPACE.
       01  NMAILID REDEFINES PMAILID     PIC 9(2).
       01  PSRC                   PIC X(3) VALUE SPACE.
       01  NSRC REDEFINES PSRC           PIC 9(3).
       01  PACTDATE               PIC X(6) VALUE SPACE.
       01  NACTDATE REDEFINES PACTDATE   PIC 9(6).
       01  PADDRCHG               PIC X(6) VALUE SPACE.
       01  NADDRCHG REDEFINES PADDRCHG   PIC 9(6).
       01  PPROFIT                PIC X(8) VALUE SPACE.
       01  EPROFIT                PIC ZZZZ9.99 VALUE ZERO.
       01  PDCODE                 PIC X(1) VALUE SPACE.
       01  NDCODE REDEFINES PDCODE        PIC 9(1).
       01  PCOLLC                 PIC X(1) VALUE SPACE.
       01  NCOLLC REDEFINES PCOLLC        PIC 9(1).
       01  PPAYDATE               PIC X(6) VALUE SPACE.
       01  NPAYDATE REDEFINES PPAYDATE    PIC 9(6).
       01  PAYDAT1                PIC X(6) VALUE SPACE.
       01  PAYDAT2                PIC X(6) VALUE SPACE.
       01  PAYDAT3                PIC X(6) VALUE SPACE.
       01  PAYDAT4                PIC X(6) VALUE SPACE.
       01  PAYDAT5                PIC X(6) VALUE SPACE.
       01  PAMOUNT                PIC X(8) VALUE SPACE.
       01  EAMOUNT                PIC ZZZZ9.99 VALUE ZERO.
       01  XAMOUNT.
          05   HTAL               PIC 9(5) VALUE ZERO.
          05   FILLER             PIC X VALUE ".".
          05   DEC                PIC 9(2) VALUE ZERO.
       01  AMOUNT1                PIC X(8) VALUE SPACE.
       01  AMOUNT2                PIC X(8) VALUE SPACE.
       01  AMOUNT3                PIC X(8) VALUE SPACE.
       01  AMOUNT4                PIC X(8) VALUE SPACE.
       01  AMOUNT5                PIC X(8) VALUE SPACE.
       01  VAL                    PIC X(2) VALUE SPACE.
       01  VAL-NUM REDEFINES VAL   PIC 9(2).
```

Figure 157 (Part 1 of 11). New Panel Definitions

```
       01  PANEL-RADER.
          05   R1.
             10   NR                  PIC X(2) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   CUSTNO              PIC X(10) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   FIRSTNAME           PIC X(13) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   LASTNAME            PIC X(18) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   STREET              PIC X(26) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   ZIPCODE             PIC X(5) VALUE SPACE.
          05   R2.
             10   NR                  PIC X(2) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   CUSTNO              PIC X(10) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   FIRSTNAME           PIC X(13) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   LASTNAME            PIC X(18) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   STREET              PIC X(26) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   ZIPCODE             PIC X(5) VALUE SPACE.
          05   R3.
             10   NR                  PIC X(2) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   CUSTNO              PIC X(10) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   FIRSTNAME           PIC X(13) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   LASTNAME            PIC X(18) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   STREET              PIC X(26) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   ZIPCODE             PIC X(5) VALUE SPACE.
          05   R4.
             10   NR                  PIC X(2) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   CUSTNO              PIC X(10) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   FIRSTNAME           PIC X(13) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   LASTNAME            PIC X(18) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   STREET              PIC X(26) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   ZIPCODE             PIC X(5) VALUE SPACE.
          05   R5.
             10   NR                  PIC X(2) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   CUSTNO              PIC X(10) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   FIRSTNAME           PIC X(13) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   LASTNAME            PIC X(18) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   STREET              PIC X(26) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   ZIPCODE             PIC X(5) VALUE SPACE.
```

Figure 157 (Part 2 of 11). New Panel Definitions

```
        05  R6.
            10  NR                PIC X(2) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  CUSTNO            PIC X(10) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  FIRSTNAME         PIC X(13) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  LASTNAME          PIC X(18) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  STREET            PIC X(26) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  ZIPCODE           PIC X(5) VALUE SPACE.
        05  R7.
            10  NR                PIC X(2) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  CUSTNO            PIC X(10) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  FIRSTNAME         PIC X(13) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  LASTNAME          PIC X(18) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  STREET            PIC X(26) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  ZIPCODE           PIC X(5) VALUE SPACE.
        05  R8.
            10  NR                PIC X(2) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  CUSTNO            PIC X(10) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  FIRSTNAME         PIC X(13) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  LASTNAME          PIC X(18) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  STREET            PIC X(26) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  ZIPCODE           PIC X(5) VALUE SPACE.
        05  R9.
            10  NR                PIC X(2) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  CUSTNO            PIC X(10) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  FIRSTNAME         PIC X(13) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  LASTNAME          PIC X(18) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  STREET            PIC X(26) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  ZIPCODE           PIC X(5) VALUE SPACE.
        05  R10.
            10  NR                PIC X(2) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  CUSTNO            PIC X(10) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  FIRSTNAME         PIC X(13) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  LASTNAME          PIC X(18) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  STREET            PIC X(26) VALUE SPACE.
            10  FILLER            PIC X(1) VALUE SPACE.
            10  ZIPCODE           PIC X(5) VALUE SPACE.
```

Figure 157 (Part 3 of 11). New Panel Definitions

```
          05   R11.
             10   NR                  PIC X(2) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   CUSTNO              PIC X(10) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   FIRSTNAME           PIC X(13) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   LASTNAME            PIC X(18) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   STREET              PIC X(26) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   ZIPCODE             PIC X(5) VALUE SPACE.
          05   R12.
             10   NR                  PIC X(2) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   CUSTNO              PIC X(10) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   FIRSTNAME           PIC X(13) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   LASTNAME            PIC X(18) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   STREET              PIC X(26) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   ZIPCODE             PIC X(5) VALUE SPACE.
          05   R13.
             10   NR                  PIC X(2) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   CUSTNO              PIC X(10) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   FIRSTNAME           PIC X(13) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   LASTNAME            PIC X(18) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   STREET              PIC X(26) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   ZIPCODE             PIC X(5) VALUE SPACE.
          05   R14.
             10   NR                  PIC X(2) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   CUSTNO              PIC X(10) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   FIRSTNAME           PIC X(13) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   LASTNAME            PIC X(18) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   STREET              PIC X(26) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   ZIPCODE             PIC X(5) VALUE SPACE.
          05   R15.
             10   NR                  PIC X(2) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   CUSTNO              PIC X(10) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   FIRSTNAME           PIC X(13) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   LASTNAME            PIC X(18) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   STREET              PIC X(26) VALUE SPACE.
             10   FILLER              PIC X(1) VALUE SPACE.
             10   ZIPCODE             PIC X(5) VALUE SPACE.
```

Figure 157 (Part 4 of 11). New Panel Definitions

```
*
       01  BLANK-P-RADER.
          05   R1                    PIC X(76) VALUE SPACE.
          05   R2                    PIC X(76) VALUE SPACE.
          05   R3                    PIC X(76) VALUE SPACE.
          05   R4                    PIC X(76) VALUE SPACE.
          05   R5                    PIC X(76) VALUE SPACE.
          05   R6                    PIC X(76) VALUE SPACE.
          05   R7                    PIC X(76) VALUE SPACE.
          05   R8                    PIC X(76) VALUE SPACE.
          05   R9                    PIC X(76) VALUE SPACE.
          05   R10                   PIC X(76) VALUE SPACE.
          05   R11                   PIC X(76) VALUE SPACE.
          05   R12                   PIC X(76) VALUE SPACE.
          05   R13                   PIC X(76) VALUE SPACE.
          05   R14                   PIC X(76) VALUE SPACE.
          05   R15                   PIC X(76) VALUE SPACE.

      *******************************************************************
      *   CURSOR-POSITION is returned by ADIS containing the position   *                 *
      *   of the cursor when the ACCEPT was terminated.                 *                 *
      *******************************************************************

       01  CURSOR-POSITION.
           03  CURSOR-ROW            PIC 99.
           03  CURSOR-COLUMN         PIC 99.

      **********************************************************
      *   Parameters to be used for the X"AF" call             *                          *
      **********************************************************

       01  SET-BIT-PAIRS             PIC 9(2) COMP-X VALUE 1.
       01  GET-SINGLE-CHARACTER      PIC 9(2) COMP-X VALUE 26.

       01  KEY-STATUS.
           03  KEY-TYPE              PIC X.
           03  KEY-CODE-1            PIC 9(2) COMP-X.
           03  KEY-CODE-2            PIC 9(2) COMP-X.

       01  USER-KEY-CONTROL.
           03  USER-KEY-SETTING      PIC 9(2) COMP-X.
           03  FILLER                PIC X VALUE "1".
           03  FIRST-USER-KEY        PIC 9(2) COMP-X.
           03  NUMBER-OF-KEYS        PIC 9(2) COMP-X.

       01  ENABLE-F3.
           03  FILLER         PIC 9(2) COMP-X VALUE 1.
           03  FILLER         PIC X VALUE "1".
           03  FILLER         PIC 9(2) COMP-X VALUE 3.
           03  FILLER         PIC 9(2) COMP-X VALUE 1.

       01  ENABLE-F10.
           03  FILLER         PIC 9(2) COMP-X VALUE 1.
           03  FILLER         PIC X VALUE "1".
           03  FILLER         PIC 9(2) COMP-X VALUE 10.
           03  FILLER         PIC 9(2) COMP-X VALUE 1.

       01  DISABLE-ALL-OTHER-KEYS.
           03  FILLER         PIC 9(2) COMP-X VALUE 0.
           03  FILLER         PIC X VALUE "1".
           03  FILLER         PIC 9(2) COMP-X VALUE 0.
           03  FILLER         PIC 9(2) COMP-X VALUE 126.
```
Figure 157 (Part 5 of 11). New Panel Definitions

```
      *********************************************************
      *   Screen Section.                                     *                            *
      *********************************************************
       SCREEN SECTION.

       01  PANEL1.
           03  blank screen.
           03  line 1 column 28
                     value "Samlaren AB" highlight.
           03  line 2 column 28
                     value "On-line update" highlight.
           03  line 9 column 18
                     value "01" highlight.
           03  line 9 column 22
                     value "= Enrollment".
           03  line 10 column 18
                     value "23" highlight.
           03  line 10 column 22
                     value "= Address change".
           03  line 11 column 18
                     value "60" highlight.
           03  line 11 column 22
                     value "= Payment".
           03  line 12 column 18
                     value "99" highlight.
           03  line 12 column 22
                     value "= Delete".
           03  line 13 column 18
                     value "S" highlight.
           03  line 13 column 22
                     value "= Search".
           03  line 17 column 5
                     value "ACTION : " highlight.
           03  pic X(2) using ACT reverse-video prompt " ".
           03  line 17 column 23
                     value "ARGUMENT : " highlight.
           03  pic X(18) using ARG reverse-video prompt " ".
           03  line 18 column 1
                  pic X(70) using ERRMSG highlight usage is display.
           03  line 19 column 9
                     value "PF3 = END" highlight.
       01  PANEL2.
           03  blank screen.
           03  line 1 column 28
                     value "Samlaren AB" highlight.
           03  line 2 column 29
                     value "ENROLLMENTS" .
           03  line 8 column 6
                     value "FIRST NAME  : " highlight.
           03  pic X(13) using PFNAME reverse-video prompt " ".
           03  line 10 column 6
                     value "LAST NAME   : " highlight.
           03  pic X(18) using PLNAME reverse-video prompt " ".
           03  line 12 column 6
                     value "STREET      : " highlight.
           03  pic X(26) using PSTREET reverse-video prompt " ".
           03  line 14 column 6
                     value "ZIP CODE    : " highlight.
           03  pic X(5) using PZIPCODE reverse-video prompt " ".
           03  line 14 column 31
                     value "CITY : ".
           03  pic X(20) using PCITY usage is display.
           03  line 16 column 6
                     value "CUSTOMER NO.: " highlight.
```

Figure 157 (Part 6 of 11). New Panel Definitions

```
        03  pic X(10) using PCUSTNO reverse-video prompt " ".
        03  line 16 column 35
                  value "REFERENSE NO. : ".
        03  pic X(10) using PREFNO usage is display.
        03  line 18 column 6
                  value "MAIL ID     : " highlight.
        03  pic X(2) using PMAILID reverse-video prompt " ".
        03  line 18 column 35
                  value "SOURCE CODE   : " highlight.
        03  pic X(3) using PSRC reverse-video prompt " ".
        03  line 22 column 1
               pic X(70) using ERRMSG highlight usage is display.
        03  line 24 column 9
                  value "PF3 = END" highlight.
    01  PANEL3.
        03  blank screen.
        03  line 1 column 28
                  value "Samlaren AB" highlight.
        03  line 2 column 28
                  value "ADDRESS CHANGE" .
        03  line 8 column 6
                  value "FIRST NAME  : " .
        03  pic X(13) using PFNAME usage is display.
        03  line 10 column 6
                  value "LAST NAME   : " .
        03  pic X(18) using PLNAME usage is display.
        03  line 12 column 6
                  value "STREET      : " highlight.
        03  pic X(26) using PSTREET reverse-video prompt " ".
        03  line 14 column 6
                  value "ZIP CODE    : " highlight.
        03  pic X(5) using PZIPCODE reverse-video prompt " ".
        03  line 14 column 31
                  value "CITY : ".
        03  pic X(20) using PCITY usage is display.
        03  line 16 column 6
                  value "CUSTOMER NO.: " highlight.
        03  pic X(10) using PCUSTNO reverse-video prompt " ".
        03  line 16 column 35
                  value "REFERENSE NO. : ".
        03  pic X(10) using PREFNO usage is display.
        03  line 18 column 6
                  value "MAIL ID     : " .
        03  pic X(2) using PMAILID usage is display.
        03  line 18 column 35
                  value "SOURCE CODE   : ".
        03  pic X(3) using PSRC usage is display.
        03  line 22 column 1.
        03    pic X(70) using ERRMSG highlight usage is display.
        03  line 24 column 9
                  value "PF3 = END" highlight.
    01  PANEL4.
        03  blank screen.
        03  line 1 column 28
                  value "Samlaren AB" highlight.
        03  line 2 column 28
                  value "PAYMENT PANEL" highlight.
        03  line 5 column 6
                  value "FIRST NAME   : " .
```

Figure 157 (Part 7 of 11). New Panel Definitions

```
          03  pic X(13) using PFNAME usage is display.
          03  line 6 column 6
                    value "LAST NAME    : " .
          03  pic X(18) using PLNAME usage is display.
          03  line 7 column 6
                    value "STREET       : " .
          03  pic X(26) using PSTREET usage is display.
          03  line 8 column 6
                    value "ZIP CODE     : " .
          03  pic X(5) using PZIPCODE usage is display.
          03  line 8 column 32
                    value "CITY : ".
          03  pic X(20) using PCITY usage is display.
          03  line 10 column 6
                    value "CUSTOMER NO. : " highlight.
          03  pic X(10) using PCUSTNO reverse-video prompt " ".
          03  line 10 column 36
                    value "REFERENSE NO. : ".
          03  pic X(10) using PREFNO usage is display.
          03  line 11 column 6
                    value "MAIL ID      : " .
          03  pic X(2) using PMAILID usage is display.
          03  line 11 column 36
                    value "SOURCE CODE   : ".
          03  pic X(3) using PSRC usage is display.
          03  line 12 column 6
                    value "LAST ACT DATE: " .
          03  pic X(6) using PACTDATE usage is display.
          03  line 12 column 36
                    value "ADDR.CHG.DATE : ".
          03  pic X(6) using PADDRCHG usage is display.
          03  line 14 column 6
                    value "PROFIT : ".
          03  pic X(8) using PPROFIT usage is display.
          03  line 14 column 25
                    value "DUNNING CODE : ".
          03  pic X using PDCODE usage is display.
          03  line 14 column 47
                    value "COLLECTING CODE : ".
          03  pic X using PCOLLC usage is display.
          03  line 15 column 6
      value "------------------- PAYMENT TABLE -----------------------"
              highlight.
          03  line 16 column 6
              value "LINE NO          DATE        AMOUNT" highlight.
          03  line 17 column 6
                    value "01              " highlight.
          03  pic X(6) using PAYDAT1 usage is display.
          03  line 17 column 35
                    pic X(8) using AMOUNT1 usage is display.
          03  line 18 column 6
                    value "02              " highlight.
          03  pic X(6) using PAYDAT2 usage is display.
          03  line 18 column 35
                    pic X(8) using AMOUNT2 usage is display.
          03  line 19 column 6
                    value "03              " highlight.
          03  pic X(6) using PAYDAT3 usage is display.
          03  line 19 column 35
                    pic X(8) using AMOUNT3 usage is display.
```

Figure 157 (Part 8 of 11). New Panel Definitions

```
         03  line 20 column 6
                   value "04              " highlight.
         03  pic X(6) using PAYDAT4 usage is display.
         03  line 20 column 35
                   pic X(8) using AMOUNT4 usage is display.
         03  line 21 column 6
                   value "05              " highlight.
         03  pic X(6) using PAYDAT5 usage is display.
         03  line 21 column 35
                   pic X(8) using AMOUNT5 usage is display.
         03  line 22 column 6
                   value "NEW :           " highlight.
         03 pic X(6) using PPAYDATE reverse-video prompt " ".
         03  line 22 column 35.
         03        pic X(8) using PAMOUNT reverse-video prompt " ".
         03  line 23 column 1.
         03      pic X(70) using ERRMSG highlight usage is display.
         03  line 24 column 9
                   value "PF3 = END" highlight.
     01  PANEL5.
         03  blank screen.
         03  line 1 column 28
                   value "Samlaren AB" highlight.
         03  line 2 column 28
                   value "DELETE PANEL " highlight.
         03  line 5 column 6
                   value "FIRST NAME   : " .
         03  pic X(13) using PFNAME usage is display.
         03  line 6 column 6
                   value "LAST NAME    : " .
         03  pic X(18) using PLNAME usage is display.
         03  line 7 column 6
                   value "STREET       : " .
         03  pic X(26) using PSTREET usage is display.
         03  line 8 column 6
                   value "ZIP CODE     : " .
         03  pic X(5) using PZIPCODE usage is display.
         03  line 8 column 32
                   value "CITY : ".
         03  pic X(20) using PCITY usage is display.
         03  line 10 column 6
                   value "CUSTOMER NO. : " highlight.
         03  pic X(10) using PCUSTNO reverse-video prompt " ".
         03  line 10 column 36
                   value "REFERENSE NO. : ".
         03  pic X(10) using PREFNO usage is display.
         03  line 11 column 6
                   value "MAIL ID      : " .
         03  pic X(2) using PMAILID usage is display.
         03  line 11 column 36
                   value "SOURCE CODE   : ".
         03  pic X(3) using PSRC usage is display.
         03  line 12 column 6
                   value "LAST ACT DATE: " .
         03  pic X(6) using PACTDATE usage is display.
         03  line 12 column 36
                   value "ADDR.CHG.DATE : ".
         03  pic X(6) using PADDRCHG usage is display.
         03  line 14 column 6
                   value "PROFIT : ".
         03  pic X(8) using PPROFIT usage is display.
```

Figure 157 (Part 9 of 11). New Panel Definitions

```
           03  line 14 column 25
                     value "DUNNING CODE : ".
           03  pic X using PDCODE usage is display.
           03  line 14 column 47
                     value "COLLECTING CODE : ".
           03  pic X using PCOLLC usage is display.
           03  line 15 column 6
       value "------------------- PAYMENT TABLE -----------------------"
               highlight.
           03  line 16 column 6
               value "LINE NO          DATE        AMOUNT" highlight.
           03  line 17 column 6
                     value "01              " highlight.
           03  pic X(6) using PAYDAT1 usage is display.
           03  line 17 column 35
                     pic X(8) using AMOUNT1 usage is display.
           03  line 18 column 6
                     value "02              " highlight.
           03  pic X(6) using PAYDAT2 usage is display.
           03  line 18 column 35
                     pic X(8) using AMOUNT2 usage is display.
           03  line 19 column 6
                     value "03              " highlight.
           03  pic X(6) using PAYDAT3 usage is display.
           03  line 19 column 35
                     pic X(8) using AMOUNT3 usage is display.
           03  line 20 column 6
                     value "04              " highlight.
           03  pic X(6) using PAYDAT4 usage is display.
           03  line 20 column 35
                     pic X(8) using AMOUNT4 usage is display.
           03  line 21 column 6
                     value "05              " highlight.
           03  pic X(6) using PAYDAT5 usage is display.
           03  line 21 column 35
                     pic X(8) using AMOUNT5 usage is display.
           03  line 23 column 1
                 pic X(70) using ERRMSG highlight usage is display.
           03  line 24 column 9
                     value "PF3 = END                PF10 = DELETE"
                     highlight.
       01  PANEL6.
           03  blank screen.
           03  line 1 column 22
                     value "DUPLICATE SELECTION PANEL" highlight.
           03  line 3 column 1
                     value "NR CUSTNO     FIRST NAME" highlight.
           03  line 3 column 30
              value"LAST NAME          STREET                   ZIPCODE"
                     highlight.
           03  line 4 column 1
                    pic X(79) using R1 IN PANEL-RADER usage is display.
           03  line 5 column 1
                    pic X(79) using R2 IN PANEL-RADER usage is display.
           03  line 6 column 1
                    pic X(79) using R3 IN PANEL-RADER usage is display.
           03  line 7 column 1
                    pic X(79) using R4 IN PANEL-RADER usage is display.
           03  line 8 column 1
                    pic X(79) using R5 IN PANEL-RADER usage is display.
```

Figure 157 (Part 10 of 11). New Panel Definitions

```
           03  line 9 column 1
                    pic X(79) using R6 IN PANEL-RADER usage is display.
           03  line 10 column 1
                    pic X(79) using R7 IN PANEL-RADER usage is display.
           03  line 11 column 1
                    pic X(79) using R8 IN PANEL-RADER usage is display.
           03  line 12 column 1
                    pic X(79) using R9 IN PANEL-RADER usage is display.
           03  line 13 column 1
                   pic X(79) using R10 IN PANEL-RADER usage is display.
           03  line 14 column 1
                   pic X(79) using R11 IN PANEL-RADER usage is display.
           03  line 15 column 1
                   pic X(79) using R12 IN PANEL-RADER usage is display.
           03  line 16 column 1
                   pic X(79) using R13 IN PANEL-RADER usage is display.
           03  line 17 column 1
                   pic X(79) using R14 IN PANEL-RADER usage is display.
           03  line 18 column 1
                   pic X(79) using R15 IN PANEL-RADER usage is display.
           03  line 21 column 4
                     value "SELECT NR ==> " highlight.
           03  pic X(2) using VAL reverse-video prompt " ".
           03  line 23 column 1
                 pic X(70) using ERRMSG highlight usage is display.
           03  line 24 column 9
                     value "PF3 = END" highlight.
```
Figure 157 (Part 11 of 11). New Panel Definitions

Appendix D. Sample Data Extraction Program

This chapter shows the layout of our data extraction (export) program. We decided to extract the data into character format, so we could use the DB2/6000 ASCII import facility.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. EXPTAB.
       AUTHOR. LEIF TRULSSON.
       DATE-WRITTEN. 930726.

      *REMARKS.
      *         DB2 BATCH PROGRAM.
      *         EXPORTS TABLES: CUST, NAME, PAYMENT AND ZIP
      *         FROM DATABASE IBPED.

       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.

           SELECT CUSTUT           ASSIGN TO CUSTUT.
           SELECT NAMEUT           ASSIGN TO NAMEUT.
           SELECT PAYUT            ASSIGN TO PAYUT.
           SELECT ZIPUT            ASSIGN TO ZIPUT.


       DATA DIVISION.
       FILE SECTION.

       FD  CUSTUT                  BLOCK CONTAINS 0 RECORDS
                                   LABEL RECORDS ARE STANDARD
                                   DATA RECORDS ARE CUST-UT.
       01  CUST-UT                 SYNC.
           05  FILLER              PIC X(80).

       FD  NAMEUT                  BLOCK CONTAINS 0 RECORDS
                                   LABEL RECORDS ARE STANDARD
                                   DATA RECORDS ARE NAME-UT.
       01  NAME-UT                 SYNC.
           05  FILLER              PIC X(80).

       FD  PAYUT                   BLOCK CONTAINS 0 RECORDS
                                   LABEL RECORDS ARE STANDARD
                                   DATA RECORDS ARE PAY-UT.
       01  PAY-UT                  SYNC.
           05  FILLER              PIC X(80).

       FD  ZIPUT                   BLOCK CONTAINS 0 RECORDS
                                   LABEL RECORDS ARE STANDARD
                                   DATA RECORDS ARE ZIP-UT.
       01  ZIP-UT                  SYNC.
           05  FILLER              PIC X(80).



       WORKING-STORAGE SECTION.
      *
       77  CLOCK    PIC 9(8) VALUE ZERO.
      *
           EXEC SQL
           INCLUDE SQLCA
           END-EXEC.
      *
```

Figure 158 (Part 1 of 7). Program to Extract Data from MVS/DB2

```
           EXEC SQL
              DECLARE IBPED.CUST TABLE
                (CUSTNO      NUMERIC(10) NOT NULL,
                 REFNO       NUMERIC(10) NOT NULL,
                 ACTDATE     INTEGER,
                 ADDRCHG     INTEGER,
                 PROFIT      NUMERIC(7,2),
                 MAILID      NUMERIC(2) NOT NULL,
                 SOURCECODE  NUMERIC(3) NOT NULL,
                 COLLECTCODE SMALLINT,
                 DUNNCODE    SMALLINT)
           END-EXEC.
           EXEC SQL
              DECLARE IBPED.NAME TABLE
                (CUSTNO      NUMERIC(10) NOT NULL,
                 FIRSTNAME   CHARACTER(13) NOT NULL,
                 LASTNAME    CHARACTER(18) NOT NULL,
                 STREET      CHARACTER(26) NOT NULL,
                 ZIPCODE     INTEGER NOT NULL)
           END-EXEC.
           EXEC SQL
              DECLARE IBPED.ZIP TABLE
                (ZIPCODE     INTEGER NOT NULL,
                 CITY        CHARACTER(20) NOT NULL)
           END-EXEC.
           EXEC SQL
              DECLARE IBPED.PAYMENT TABLE
                (REFNO       NUMERIC(10) NOT NULL,
                 PAYDATE     INTEGER NOT NULL,
                 AMOUNT      NUMERIC(7,2) NOT NULL)
           END-EXEC.
      *
       01  PGM-CUST.
          10  CUSTNO         COMP-3 PIC S9(10)V.
          10  REFNO          COMP-3 PIC S9(10)V.
          10  ACTDATE        COMP   PIC S9(6).
          10  ADDRCHG        COMP   PIC S9(6).
          10  PROFIT         COMP-3 PIC S9(5)V9(2).
          10  MAILID         COMP-3 PIC S9(2)V.
          10  SOURCECODE     COMP-3 PIC S9(3)V.
          10  COLLECTCODE    COMP   PIC S9(1).
          10  DUNNCODE       COMP   PIC S9(1).
       01  PGM-NAME.
          10  CUSTNO         COMP-3 PIC S9(10)V.
          10  FIRSTNAME             PIC X(13).
          10  LASTNAME              PIC X(18).
          10  STREET                PIC X(26).
          10  ZIPCODE        COMP   PIC S9(5).
       01  PGM-ZIP.
          10  ZIPCODE        COMP   PIC S9(5).
          10  CITY                  PIC X(20).
       01  PGM-PAYMENT.
          10  REFNO          COMP-3 PIC S9(10)V.
          10  PAYDATE        COMP   PIC S9(6).
          10  AMOUNT         COMP-3 PIC S9(5)V9(2).
      *
```

Figure 158 (Part 2 of 7). Program to Extract Data from MVS/DB2

```
        01  BLANK-CUST.
           10  CUSTNO         COMP-3 PIC S9(10)V VALUE ZERO.
           10  REFNO          COMP-3 PIC S9(10)V VALUE ZERO.
           10  ACTDATE        COMP   PIC S9(6) VALUE ZERO.
           10  ADDRCHG        COMP   PIC S9(6) VALUE ZERO.
           10  PROFIT         COMP-3 PIC S9(5)V9(2) VALUE ZERO.
           10  MAILID         COMP-3 PIC S9(2)V VALUE ZERO.
           10  SOURCECODE     COMP-3 PIC S9(3)V VALUE ZERO.
           10  COLLECTCODE    COMP   PIC S9(1) VALUE ZERO.
           10  DUNNCODE       COMP   PIC S9(1) VALUE ZERO.
        01  BLANK-NAME.
           10  CUSTNO         COMP-3 PIC S9(10)V VALUE ZERO.
           10  FIRSTNAME             PIC X(13) VALUE SPACE.
           10  LASTNAME              PIC X(18) VALUE SPACE.
           10  STREET                PIC X(26) VALUE SPACE.
           10  ZIPCODE        COMP   PIC S9(5) VALUE ZERO.
        01  BLANK-ZIP.
           10  ZIPCODE        COMP   PIC S9(5) VALUE ZERO.
           10  CITY                  PIC X(20) VALUE SPACE.
        01  BLANK-PAYMENT.
           10  REFNO          COMP-3 PIC S9(10)V VALUE ZERO.
           10  PAYDATE        COMP   PIC S9(6) VALUE ZERO.
           10  AMOUNT         COMP-3 PIC S9(5)V9(2) VALUE ZERO.
       *
        01  TRANS-DATA               PIC X(80) VALUE SPACE.
       *
        01  CUST-DATA REDEFINES TRANS-DATA.
           05  CUSTNO                PIC 9(10).
           05  FILLER                PIC X(1) .
           05  REFNO                 PIC 9(10).
           05  FILLER                PIC X(1) .
           05  ACTDATE               PIC 9(6).
           05  FILLER                PIC X(1) .
           05  ADDRCHG               PIC 9(6).
           05  FILLER                PIC X(1) .
           05  PROFIT                PIC 9(9).
           05  FILLER                PIC X(1) .
           05  MAILID                PIC 9(2).
           05  FILLER                PIC X(1) .
           05  SOURCECODE            PIC 9(3).
           05  FILLER                PIC X(1) .
           05  COLLECTCODE           PIC 9(1).
           05  FILLER                PIC X(1) .
           05  DUNNCODE              PIC 9(1).
           05  FILLER                PIC X(24).
       *
        01  NAME-DATA REDEFINES TRANS-DATA.
           05  CUSTNO                PIC 9(10).
           05  FILLER                PIC X(1) .
           05  FIRSTNAME             PIC X(13).
           05  FILLER                PIC X(1) .
           05  LASTNAME              PIC X(18).
           05  FILLER                PIC X(1) .
           05  STREET                PIC X(26).
           05  FILLER                PIC X(1) .
           05  ZIPCODE               PIC 9(5).
           05  FILLER                PIC X(14) .
       *
```

Figure 158 (Part 3 of 7). Program to Extract Data from MVS/DB2

```
       01  PAY-DATA REDEFINES TRANS-DATA.
          05  REFNO                PIC 9(10).
          05  FILLER               PIC X(1) .
          05  PAYDATE              PIC 9(6).
          05  FILLER               PIC X(1) .
          05  AMOUNT               PIC 99999.99.
          05  FILLER               PIC X(54) .
      *
       01  ZIP-DATA REDEFINES TRANS-DATA.
          05  ZIPCODE              PIC 9(5).
          05  FILLER               PIC X(1)  .
          05  CITY                 PIC X(20).
          05  FILLER               PIC X(54) .
      *

      *
       PROCEDURE DIVISION.

       STARTIT.
      *
       OPEN-FILES.
           DISPLAY "OPEN FILES    " CLOCK.
           OPEN OUTPUT CUSTUT NAMEUT PAYUT ZIPUT.
           ACCEPT CLOCK FROM TIME.

       PROCESS-TAB.
           PERFORM PCUST.
           PERFORM PNAME.
           PERFORM PPAY.
           PERFORM PZIP.
       CLOSE-FILES.
           ACCEPT CLOCK FROM TIME.
           DISPLAY "CLOSE FILES   " CLOCK.
           CLOSE CUSTUT NAMEUT PAYUT ZIPUT.
       ENDIT.
           STOP RUN.
      *****************************
      *
      * EXPORT CUST TABLE
      *
      *****************************
       PCUST SECTION.
       AA000.
           DISPLAY "READING CUST TABLE".
           EXEC SQL
              WHENEVER SQLERROR GO TO AA800
           END-EXEC.
           EXEC SQL
              DECLARE C1 CURSOR FOR
                 SELECT *
                    FROM IBPED.CUST
           END-EXEC.
           EXEC SQL
              OPEN C1
           END-EXEC.
```
Figure 158 (Part 4 of 7). Program to Extract Data from MVS/DB2

```
 AA100.
     MOVE BLANK-CUST TO PGM-CUST.
     EXEC SQL
        FETCH C1
           INTO :PGM-CUST
     END-EXEC.
     IF SQLCODE IS = 100
        GO TO AA900.
     MOVE CUSTNO IN PGM-CUST TO CUSTNO IN CUST-DATA.
     MOVE REFNO IN PGM-CUST TO REFNO IN CUST-DATA.
     MOVE ACTDATE IN PGM-CUST TO ACTDATE IN CUST-DATA.
     MOVE ADDRCHG IN PGM-CUST TO ADDRCHG IN CUST-DATA.
     MOVE PROFIT IN PGM-CUST TO PROFIT IN CUST-DATA.
     MOVE MAILID IN PGM-CUST TO MAILID IN CUST-DATA.
     MOVE SOURCECODE IN PGM-CUST TO SOURCECODE IN CUST-DATA.
     MOVE COLLECTCODE IN PGM-CUST TO COLLECTCODE IN CUST-DATA.
     MOVE DUNNCODE IN PGM-CUST TO DUNNCODE IN CUST-DATA.
     WRITE CUST-UT FROM TRANS-DATA.
     GO TO AA100.
 AA800.
* WE HAD A SQL-ERROR
     DISPLAY "SQLCODE =" SQLCODE.
     DISPLAY "ERROR : " SQLERRMC.
 AA900.
* CLOSE TABLE
     EXEC SQL
        CLOSE C1
     END-EXEC.
 AA999.
     EXIT.
*****************************
*
* EXPORT NAME TABLE
*
*****************************
 PNAME SECTION.
 BB000.
     MOVE SPACES TO TRANS-DATA.
     DISPLAY "READING NAME TABLE".
     EXEC SQL
        WHENEVER SQLERROR GO TO BB800
     END-EXEC.
     EXEC SQL
        DECLARE N1 CURSOR FOR
           SELECT *
              FROM IBPED.NAME
     END-EXEC.
     EXEC SQL
        OPEN N1
     END-EXEC.
 BB100.
     MOVE BLANK-NAME TO PGM-NAME.
     EXEC SQL
        FETCH N1
           INTO :PGM-NAME
     END-EXEC.
     IF SQLCODE IS = 100
        GO TO BB900.
```

Figure 158 (Part 5 of 7). Program to Extract Data from MVS/DB2

```
         MOVE CUSTNO IN PGM-NAME TO CUSTNO IN NAME-DATA.
         MOVE FIRSTNAME IN PGM-NAME TO FIRSTNAME IN NAME-DATA.
         MOVE LASTNAME IN PGM-NAME TO LASTNAME IN NAME-DATA.
         MOVE STREET IN PGM-NAME TO STREET IN NAME-DATA.
         MOVE ZIPCODE IN PGM-NAME TO ZIPCODE IN NAME-DATA.
         WRITE NAME-UT FROM TRANS-DATA.
         GO TO BB100.
     BB800.
    * WE HAD A SQL-ERROR
         DISPLAY "SQLCODE =" SQLCODE.
         DISPLAY "ERROR : " SQLERRMC.
     BB900.
    * CLOSE TABLE
         EXEC SQL
            CLOSE N1
         END-EXEC.
     BB999.
         EXIT.
    *****************************
    *
    * EXPORT PAYMENT TABLE
    *
    *****************************
     PPAY SECTION.
     CC000.
         MOVE SPACES TO TRANS-DATA.
         DISPLAY "READING PAYMENT TABLE".
         EXEC SQL
            WHENEVER SQLERROR GO TO CC800
         END-EXEC.
         EXEC SQL
            DECLARE P1 CURSOR FOR
               SELECT *
                  FROM IBPED.PAYMENT
         END-EXEC.
         EXEC SQL
            OPEN P1
         END-EXEC.
     CC100.
         MOVE BLANK-PAYMENT TO PGM-PAYMENT.
         EXEC SQL
            FETCH P1
               INTO :PGM-PAYMENT
         END-EXEC.
         IF SQLCODE IS = 100
            GO TO CC900.
         MOVE REFNO IN PGM-PAYMENT TO REFNO IN PAY-DATA.
         MOVE PAYDATE IN PGM-PAYMENT TO PAYDATE IN PAY-DATA.
         MOVE AMOUNT IN PGM-PAYMENT TO AMOUNT IN PAY-DATA.
         WRITE PAY-UT FROM TRANS-DATA.
         GO TO CC100.
     CC800.
    * WE HAD A SQL-ERROR
         DISPLAY "SQLCODE =" SQLCODE.
         DISPLAY "ERROR : " SQLERRMC.
     CC900.
    * CLOSE TABLE
         EXEC SQL
            CLOSE P1
         END-EXEC.
     CC999.
         EXIT.
```

Figure 158 (Part 6 of 7). Program to Extract Data from MVS/DB2

```
*****************************
*
* EXPORT ZIP TABLE
*
*****************************
 PZIP SECTION.
 DD000.
     MOVE SPACES TO TRANS-DATA.
     DISPLAY "READING ZIP TABLE".
     EXEC SQL
        WHENEVER SQLERROR GO TO DD800
     END-EXEC.
     EXEC SQL
        DECLARE Z1 CURSOR FOR
           SELECT *
              FROM IBPED.ZIP
     END-EXEC.
     EXEC SQL
        OPEN Z1
     END-EXEC.
 DD100.
     MOVE BLANK-ZIP TO PGM-ZIP.
     EXEC SQL
        FETCH Z1
           INTO :PGM-ZIP
     END-EXEC.
     IF SQLCODE IS = 100
        GO TO DD900.
     MOVE ZIPCODE IN PGM-ZIP TO ZIPCODE IN ZIP-DATA.
     MOVE CITY IN PGM-ZIP TO CITY IN ZIP-DATA.
     WRITE ZIP-UT FROM TRANS-DATA.
     GO TO DD100.
 DD800.
* WE HAD A SQL-ERROR
     DISPLAY "SQLCODE =" SQLCODE.
     DISPLAY "ERROR : " SQLERRMC.
 DD900.
* CLOSE TABLE
     EXEC SQL
        CLOSE Z1
     END-EXEC.
 DD999.
     EXIT.
```

Figure 158 (Part 7 of 7). Program to Extract Data from MVS/DB2

Glossary

AIC Interface. A top-level widget, together with all of its descendants.

AIC Interface File. An ASCII file consisting of a header and a sequence of X Windows style resource specifications that together describe an AIC Interface. AIC generates an interface file when you save an interface.

ASCII. The standard coded character set using 7-bit characters (8th bit for parity) and used widely on non-IBM mainframe computers.

Batch program. A batch program reads its input from a file or device and writes its output to a file or device without the interaction of a user.

BSD. Berkeley Software Distribution (UC at Berkeley, UNIX).

Callback. C code that is associated with a widget or gadget and is executed when a specified event occurs. For example, a push button has an activate callback, which is executed when a the button is pressed and released.

CLIST. Command list; a mechanism on MVS for starting a COBOL program.

COSE. Common Open Software Environment, a consortium formed in 1993 by IBM, HP, Sun, and other UNIX software and hardware vendors whose goal is to create a common software environment on their UNIX-based operating systems. Their first accomplishment is the Common Desktop Environment (CDE), a specification defining APIs that software vendors can use to create applications with a COSE “look and feel.”

Downsizing. Migrating a mainframe application to a midrange or desktop computer. Sometimes also referred to as *rightsizing* by marketing literature.

EBCDIC. A coded character set of 256 8-bit characters used on IBM and other mainframes.

EDL. Encapsulator Description Language is a mechanism for integrating an application development tool with SDE WorkBench/6000. It is a component of SDE Integrator/6000.

Encapsulation. Encapsulation is the process of integrating a tool with SDE WorkBench/6000 so it communicates with other tools by means of messages. The tool has a graphical user interface similar to other SDE WorkBench/6000 tools, and it can be controlled through Tool Manager. Encapsulation is possible using SDE Integrator/6000.

FIPS. Federal Information Processing Standard (USA).

Function key. A key appearing above or beside the normal character keys on a keyboard. The key can be programmed to perform particular functions in particular program contexts.

IEC. International Electrotechnical Commission .

ISO. International Organization for Standardization .

JCL. Job control language. On MVS, a command interpreter/programming language that is used to submit jobs (executable programs) to the operating system.

Korn shell. The default UNIX shell executed on AIX. It is virtually identical to the proposed POSIX standard shell.

NIST. National Institute of Standards and Technology (USA, formerly the National Bureau of Standards)

Object. An object is a collection of data and application program code that operates on that data.

Online program. A user provides input interactively to an online program and views its output on a display, panel or window.

Open Systems. Operating systems that are available on many different vendors' computers, across which programs may be easily ported. UNIX and MS-DOS, two operating systems that are available on a great many vendor platforms, are both considered open systems by many people. Typically, an open system supports *de facto* industry and *de jure* formal standard interfaces, subsystems, languages, and utilities.

Palette. A palette is a set of interface building blocks, such as the OSF/Motif widgets.

Pane. In the AIX operating system, a display screen, a portion of a window used to present information to the user. A window can consist of one or more panes.

Panel. In ISPF, a logical subset of data displayed in a rectangular space on a character-based display terminal. Analogous to a window on a graphical display terminal. Sometimes also called an input/output screen or display map.

POSIX. Portable operating system interface for computer environments; an IEEE operating system standard, closely related to the UNIX system (software writing).

Project. A project is a set of interfaces designed for a single application.

Property. In AIC, a property is a widget variable that defines the appearance and behavior of a widget.

Shell. Generic name for UNIX command-line interpreter. UNIX shells are also noncompiled procedural programming languages with which end users and system administrators can build utility programs.

SVID. System V Interface Definition (AT&T, UNIX).

Tool. In SDE WorkBench/6000 terminology, a tool is an encapsulated application.

Top-level widget. A top-level widget is a widget at the root of a widget hierarchy. A top-level widget interacts with the window manager.

Widget. A widget is a basic component of an X Windows user interface and has a set of built-in properties and behaviors.

X client. An application that makes calls to X Windows library subroutines to request an Xserver program perform input/output at a graphical display.

X server. The X Windows software that manages the input/output resources of a graphical display, such as the monitor, the keyboard, and the pointing devices.

X station. A network-attached device that executes the X server software that controls a display unit such as the monitor, the keyboard, and the pointing device. Some X stations also support attachment of a printer, hard disk, and other I/O devices, but an X station is not a general purpose computer. Using and X station, a user must log in at another computer on the network.

X Windows. A client-server praphical windowing product from MIT (Massachuset Institute of Technology).

List of Abbreviations

4GL	Fourth-Generation Language
AIC	AIXwindows Interface Composer
AD	Application Development
ADT	Application Development Toolkit
AES	Application Environment Specification
AIC	AIXwindows Interface Composer
AIX	Advanced Interactive Executive
ANSI	American National Standards Institute
API	Application Programming Interface
APPC	Advanced Program-to-Program Communications
ASCII	American National Standard Code for Information Interchange
BMS	Broadcast Message Server
BSD	Berkeley Software Distribution
CASE	Computer Aided Software Engineering
CLI	Call Level Interface
CLIST	Command List
CMVC	Configuration Management Version Control
COSE	Common Open Software Environment
CUA	Common User Access
DB2	DATABASE 2
DCF	Data Composition Facility
DDCS	Distributed Database Connection Services
DNS	Domain Name Service
DRDA	Distributed Database Relational Architecture
DSOM	Distributed System Object Model
EBCDIC	Extended Binary Coded Decimal Interchange Code
ECMA	European Computer Manufacturers Association
EDL	Encapsulator Description Language
FTP	File Transfer Protocol
GB	Gigabyte
GUI	Graphical User Interface
HFT	High Function Terminal
IBM	International Business Machines Corporation
IO	Input/output
IP	Internet Protocol
ISPF	Interactive System Productivity Facility
ITSC	International Technical Support Center
ITSO	International Technical Support Organization
LAN	Local Area Network
LPP	Licensed Program Product
LPEX	Live Parsing Extensible Editor (PC and AIX version of LEXX)
MB	Megabyte
MF	Micro Focus
MVS	Multiple Virtual Storage
NCS	Network Computing Service
NFS	Network File System
NIS	Network Information System
NLS	National Language Support
OEM	Original equipment manufacturer
OODB	Object Oriented Data Base
OPP	Optional Program Product
OS	Operating system
OSF	Open Software Foundation
PCTE	Portable Common Tools Environment
POWER	Performance Optimized With Enhanced RISC
PROFS	Professional Office System
PTF	Program Temporary Fix
PVCS	Program Version Control System

RCS	Revision Control System
RISC	Reduced Instruction Set Computer
SAA	Systems Application Architecture
SCCS	Source Code Control System
SCRB	Software Change Review Board
SDE	Software Development Environment
SDK	Software Developers Kit
SDRB	Software Design Review Board
SEE	Software Engineering Environment
SEI	Software Engineering Institute
SMP	Symmetrical multiprocessor
SNA	Systems Network Architecture
SOM	System Object Module
SQA	Software Quality Assurance
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol
UIL	User Interface Language
US	United States
X11R4	X Windows Version 11 Release 4
X11R5	X Windows Version 11 Release 5
XL C	XL C Compiler/6000
XL C^{++}	XL C^{++} Compiler/6000

Index

ITSO Technical Bulletin Evaluation RED000

International Technical Support Organization
AIX Application Development and
How To Migrate and Enhance
Your Legacy Applications
May 1995

Publication No. GG24-4177-00

Your feedback is very important to help us maintain the quality of ITSO Bulletins. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246
- Send a note to REDBOOK@VNET.IBM.COM

Please rate on a scale of 1 to 5 the subjects below.
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction ____

Organization of the book	____	Grammar/punctuation/spelling	____
Accuracy of the information	____	Ease of reading and understanding	____
Relevance of the information	____	Ease of finding information	____
Completeness of the information	____	Level of technical detail	____
Value of illustrations	____	Print quality	____

Please answer the following questions:

a) If you are an employee of IBM or its subsidiaries:

Do you provide billable services for 20% or more of your time? Yes____ No____

Are you in a Services Organization? Yes____ No____

b) Are you working in the USA? Yes____ No____

c) Was the Bulletin published in time for your needs? Yes____ No____

d) Did this Bulletin meet your needs? Yes____ No____

If no, please explain:

____

____

What other topics would you like to see in this Bulletin?

____

____

What other Technical Bulletins would you like to see published?

____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

Name	Address
Company or Organization	
Phone No.	

ITSO Technical Bulletin Evaluation **RED000**
GG24-4177-00

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Organization
Department 471, Building 070B
5600 COTTLE ROAD
SAN JOSE CA
USA 95193-0001

Fold and Tape **Please do not staple** Fold and Tape

GG24-4177-00

Cut or Fold
Along Line

Printed in U.S.A.

GG24-4177-00