Did You Say CMVC?

Document Number GG24-4178-00

September 1994

International Technical Support Organization
San Jose Center

Take Note!

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

First Edition (September 1994)

This edition applies to Version 2, Release 1, Modification Level 0, of IBM Configuration Management and Version Control/6000, Program Number 5765-207, for use with the AIX Operating System 3.2

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 reader's feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. 471 Building 70B
5600 Cottle Road
San Jose, California 95193-0001

Abstract

This document describes the use of the IBM Configuration Management and Version Control (CMVC) product in the scope of the downsizing of a DATABASE 2 business application from an IBM mainframe running MVS to the RISC System/6000 and AIX/6000.

The book gives you a general view of the CMVC features and an understanding of how CMVC can be used to improve quality and increase productivity.

This document was written for customers and system engineers who need to know how to set up, configure, customize, and use IBM CMVC for UNIX in the scope of a given application development.

AD AX (245 pages)

Contents

Figures

Tables

Special Notices

This publication is intended to help developers, project managers, system administrators, and software configuration administrators set up, configure, customize, and use IBM CMVC for their application developement. The information in this publication is not intended as the specification of any programming interfaces that are provided by Configuration Management and Version Control/6000. See the PUBLICATIONS section of the IBM Programming Announcement for IBM Configuration Management and Version Control for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 208 Harbor Drive, Stamford, CT 06904 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (VENDOR) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

The following terms, which are denoted by an asterisk (*) in this publication, are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX	AIX/6000
AIXwindows	VS COBOL II
Common User Access	CUA
DATABASE 2	DB2

DB2/6000	IBM
MVS	OS/2
Presentation Manager	RISC System/6000
Xstation Manager	

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

Yellow Pages	British Telecommunications PLC
Hewlett-Packard	Hewlett-Packard Company
HP	Hewlett-Packard Company
HP-UX	Hewlett-Packard Company
SoftBench	Hewlett-Packard Company
Motif	Open Software Foundation, Inc.
OSF	Open Software Foundation, Inc.
OSF/Motif	Open Software Foundation, Inc.
Sun	Sun Microsystems Incorporated
SunOS	Sun Microsystems Incorporated
Solaris	Sun Microsystems Incorporated
NFS	Sun Microsystems Incorporated
Network File System	Sun Microsystems Incorporated
Micro Focus	Micro Focus Limited
Micro Focus COBOL	Micro Focus Limited
Toolbox	Micro Focus Limited
UNIX	X/Open Company Limited
X Window System	Massachusetts Institute of Technology
MIT	Massachusetts Institute of Technology
ORACLE	Oracle Corporation, Inc
SYBASE	Sybase, Inc.
INFORMIX	Informix Software, Inc.
PVCS Version Manager	INTERSOLV, Inc.
MS-DOS	Microsoft Corporation
Microsoft	Microsoft Corporation
Microsoft Windows	Microsoft Corporation
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.
Internet	Internet, Inc.
ISO	International Organization for Standardization

Preface

This document is intended for developers, project managers, system administrators, and the software configuration administrators who want to know how to set up, configure, customize, and use IBM CMVC for their application development. This book illustrates how a project applied CMVC to organize and manage application data, report the problems, and track the changes over the project time frame. It also shows how to enhance the productivity of programmers familiar with UNIX, and the quality of the produced application.

This book is organized around a hypothetical story of a project, which explored the downsizing of a legacy application from an IBM mainframe to an IBM RISC System/6000. This volume uses this project as vehicle for explaining CMVC concepts and showing the use of CMVC for UNIX. Additional topics not specifically related to this project are included in the appendixes.

This book does not address the CMVC client products for OS/2, MS-DOS, and Microsoft Windows.

How to Use This Document

This document provides information of value to readers with varying perspectives and background in software development and configuration management. We anticipate that some readers will benefit by focusing on particular chapters, while skipping others. To help the reader make this choice, we provide the following description of each chapter and its audience:

- Chapter 1, “Software Configuration Management and Change Management Overview”

  This chapter briefly describes the objectives and benefits associated with the processes known as software configuration management and change management. This chapter also introduces the main functions of IBM CMVC.

- Chapter 2, “Discovering CMVC: An New Application Project Is Introduced to CMVC”

  This chapter describes an application development project for which CMVC provided configuration management support. The project involved moving the application development environment for an MVS DATABASE 2 COBOL application to AIX, and then modernizing and migrating the application itself to AIX. This chapter illustrates how project personnel are introduced to CMVC through a series of hypothetical dialogs which the project manager, developer, builder, and tester have with their system administrator and software configuration management administrator. This second chapter should be read by all readers, because it provides the background necessary to understand the remaining chapters.

- Chapter 3, “Overview of the Application Development Project”

  This chapter describes the application under development and the development environment established on AIX. It should be read by readers who want detailed information about the original MVS application, its design, and user interfaces. Readers who have a particular interest in learning how to organize an AIX software development environment should read this chapter. It details machine configurations, LPP versions, and file system

organization. It is not, however, necessary to read this chapter as a prerequisite to reading the following chapters.

- Chapter 4, “Planning for CMVC”

  This chapter examines how this project approached CMVC. It offers general advice on setting up CMVC as well, based on other experiences with CMVC. This chapter should be of interest to software configuration management administrators and project managers. Reading this chapter is not a prerequisite for understanding the following chapters, but it is closely related to the material presented in Chapter 5, “Using CMVC.”

- Chapter 5, “Using CMVC”

  This chapter illustrates typical user interactions with CMVC Clients. It shows you how to use both the graphical and command-line user interfaces to perform the actions described in Chapter 2, “Discovering CMVC: An New Application Project Is Introduced to CMVC.” This chapter should be read by anyone wanting to learn quickly how to use the CMVC Clients. This chapter can be thought of as a short user′s guide describing the most common CMVC commands.

- Chapter 6, “Installing CMVC and Supporting Databases”

  This chapter describes how we installed and configured CMVC and CMVC prerequisite program products. This chapter should be read by system administrators.

The following appendices are narrowly focused, offering advice on a variety of topics. Readers should identify those which are relevant to their specific responsibilities and interest areas. Except for the first two, which are conceptual in nature, these appendices contain detailed technical “how-to” information.

- Appendix A, “Implementation of ISO 9001 Using CMVC”

  This appendix provides a brief introduction to the ISO 9000 and discusses how CMVC can be utilized to implement some key ISO 9001 requirements.

- Appendix B, “Monitoring and Enhancing the Quality of Software with CMVC”

  This appendix briefly outlines software quality and the fundamentals of Software Reliability, and demonstrates how CMVC was used to provide data that can be used to predict and improve the quality of software products under development.

- Appendix C, “User Exit Samples and Suggestions”

  This appendix describes two user exit programs used by our project. One inserts a module header and SCCS keywords into C source files. The other generates unique alphanumeric defect and feature numbers. It also gives some suggestions of user exit programs, which could be associated with certain CMVC actions.

- Appendix D, “Hints and Tips for Using CMVC”

  This appendix gives some general tips and hints, such as the use of the UNIX **cron** daemon in CMVC family maintenance, the shell script samples shipped with CMVC, the CMVC client log file, and the CMVC family log file. It also describes some procedures we ran to make some minor file modifications for our project, such as file name change, file deletion, and integrated track reuse.

- Appendix E, “CMVC and SDE WorkBench/6000”

  This appendix discusses how to use CMVC with SDE WorkBench/6000.

- Appendix F, “Source File and Program Identification with CMVC Keywords”

  This appendix describes some SCCS keywords supported by CMVC, and shows how we used these keywords in our C source programs.

- Appendix G, “Appendix: Setting Up NetLS”

  This appendix gives a procedure to rapidly configure NetLS in the scope of a small network configuration.

- Appendix H, “Tailoring CMVC Windows for Different Types of Users”

  This appendix shows some CMVC Tasks windows customized according to the roles of the people on our project, such as project manager, developer, and builder. It also describes how to use CMVC to calculate some statistics or metrics and to automate the problem tracking and build processes.

Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document.

CMVC Publications

- *IBM CMVC Concepts*, SC09-1633, introduces the fundamentals of the configuration management, version control, change control, and problem tracking features of CMVC. It also defines the concepts that are the foundation of the CMVC actions and describes their interrelationships. (The CMVC Version 1 equivalent document was *Understanding IBM AIX CMVC/6000 Concepts*, SC09-1433.)
- *IBM CMVC Server Administration and Installation*, SC09-1631, contains detailed information for planning, installing, customizing, operating, and maintaining the CMVC Server.
- *IBM CMVC Client Installation and Configuration*, SC09-1596, contains the detailed information you need to install and configure the CMVC clients.
- *Using CMVC Clients for HP and Sun*, SC09-1518, describes the differences between these clients and the AIX CMVC clients, and tells how to install and configure them. (These CMVC clients are not discussed in this Redbook.)
- *IBM CMVC Client/2 Getting Started*, SC09-1599, and *IBM CMVC Client/2 Users Guide*, SC09-1783, describe how to use the OS/2 client graphical user interface for CMVC. (This client is not discussed in this Redbook.)
- *IBM CMVC User's Guide*, SC09-1634, describes how to perform specific CMVC actions with the CMVC graphical user interface or the message-integrated CMVC client GUI. (The CMVC Version 1 equivalent document was *IBM AIX CMVC/6000 User's Guide*, SC09-1430.)
- *IBM CMVC User's Reference*, SC09-1597, contains the reference lists, tables, and state diagrams for CMVC, as well as a description of how message-integrated CMVC uses the Broadcast Message Server to communicate with the other integrated development tools.
- *IBM CMVC Commands Reference*, SC09-1635, describes the syntax and usage of the CMVC commands as implemented in the command-line

interface. (The CMVC Version 1 equivalent document was *IBM AIX CMVC/6000 Commands Reference*, SC09-1446.)

NetLS Publications

- *NetLS Quick Start Guide*, SC09-1661, provides the information needed to set up the Network License System software, a prerequisite for working with CMVC.
- *Managing Software Products with the Network License System*, SC09-1660, provides the information needed to manage CMVC network license information with the NetLS software.
- *Managing NCS Software*, SC09-1834, provides more detailed information on setting up and managing software with NetLS.

ORACLE Publications

- *ORACLE for IBM RISC System/6000 Installation and User's Guide Version 6.0*, 5687-60-0792
- *ORACLE RDBMS Database Administrator's Guide, Version 6.0*,3601-v6.0 1090
- For information on databases, operating systems, and software development environment used with CMVC, refer to your specific database, operating systems, and software development documentation.

International Technical Support Organization Publications

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

Bibliography of International Technical Support Organization Technical Bulletins, GG24-3070

How to Order Redbooks

IBM employees may order Redbooks 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 IBM Services Specialist.

You may order individual books, CD-ROM collections, or customized sets, called GBOFs, which relate to specific functions of interest to you.

Acknowledgments

The advisor for this project was:

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

The authors of this document are:

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

Eric Valade
IBM France

This publication is the result of a residency conducted at the International Technical Support Organization, San Jose Center.

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

Sarah McNamara
IBM Programming Systems Toronto Laboratory

Kostas Gaitanos
IBM Programming Systems Toronto Laboratory

Chapter 1. Software Configuration Management and Change Management Overview

The elements produced during the development of an application are created progressively, as new requirements are discovered and old ones are refined. You can easily forget why and when an individual element was created. You can have the latest application development tools, a highly skilled and well-managed development organization, and be following a superior development methodology, but still find that your “as-built” application does not work as it was designed, coded, tested, or documented.

It is possible that the application, which worked so well in testing, cannot be successfully re-created for delivery simply because some fixes to the code were not integrated in the final build of the application. It is also likely that some unauthorized fixes managed to find their way into the application, creating a mismatch in interfaces, calls to nonexistent subroutines, or inappropriate access to data, which no one can seem to explain. Problems of this sort represent failures in software configuration management (SCM) and change management.

Having all the right parts does not ensure a successful outcome in software development, as shown in Figure 1. It takes SCM to ensure that all the right parts are put together in the right manner, and it takes change management to ensure that any changes to those parts or their relationships are well thought out and deliberately applied.

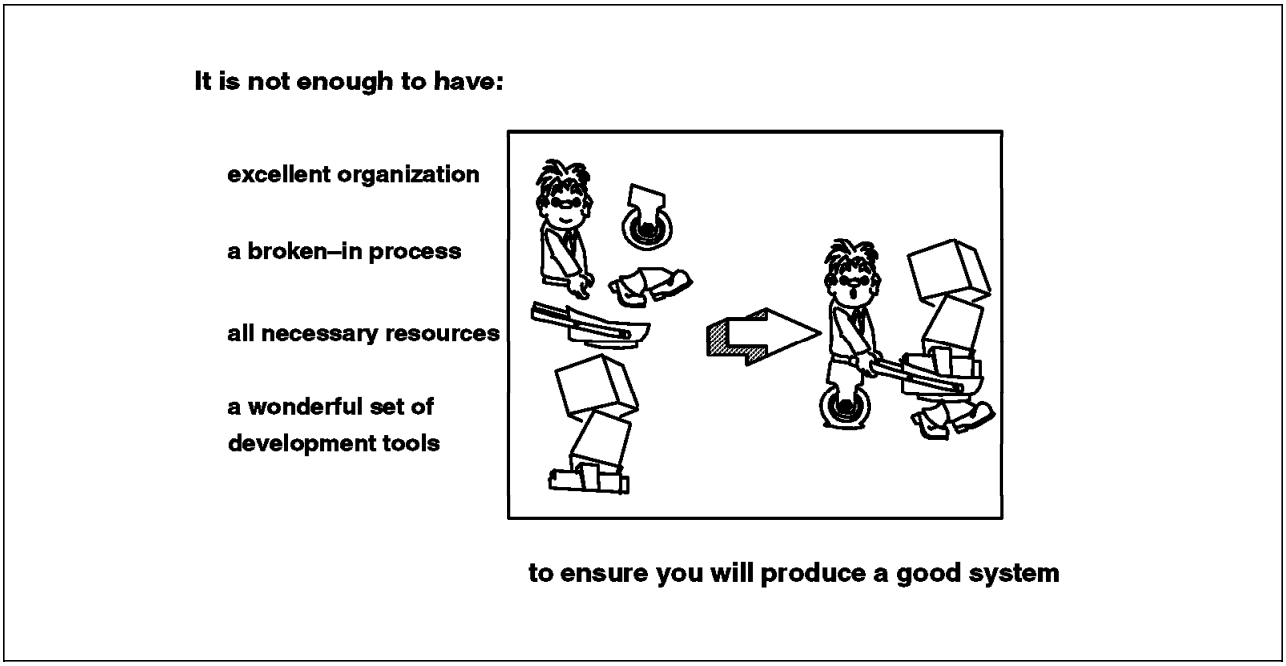

Figure 1. Why SCM and Change Management Are Necessary

In this chapter, we briefly describe the objectives and benefits associated with the processes known as SCM and change management. We also discuss the relationships among these processes, and other processes, such as project management. We also look at the relationship among these processes and software development methodologies.

1.1 Objectives

The main purpose of both SCM and change management is to ensure consistency of the elements comprising the application, as well as between the application and the documentation, which defines and supports it. The documentation that defines an application includes:

- Requirements specifications
- Interface specifications
- Design data, engineering drawings, and design specifications.

The documentation that supports an application includes various end-user manuals and separately cataloged help information.

Another very important function is to precisely identify an application's significant development "baselines." These baselines are usually associated with a major project milestone event. Typical baselines include:

- Requirements analysis and specification
- Approved design documentation
- Evaluation prototype (alpha release)
- First integration test release (beta release)
- First end-user release (first customer ship)
- Site-specific or platform-specific releases.

Identifying all the changes to a given baseline, and ensuring that they are incorporated into the next, newly forming baseline in an orderly and controlled manner is the core responsibility of change management. Change management identifies and tracks problems as well as suggested enhancements to an application. This ensures each change is carefully evaluated, and—if approved—correctly implemented and incorporated into the application.

1.2 Benefits

SCM and change management are put into place to prevent or cure problems that contribute to high development and maintenance costs, missed schedules, and customer dissatisfaction. You reap many benefits when you can successfully apply SCM and change management to software development efforts.

The primary benefit of SCM ensures that you can define and identify all elements comprising your application. It also ensures that you know exactly in which manner they are generated, preprocessed, compiled, linked or otherwise combined to form specific releases of your application and related documentation. SCM ensures that you keep a historical record of these release configurations along with the exact versions of all the components and the application itself at each release. This means that you can re-create exactly any previous release of your application, which exhibits a failing characteristic reported by end users.

SCM ensures that you can generate different parallel releases of the same application, because you know exactly which elements are unique to each version and which are common. You have a complete history of every version of every source or data file comprising your application, and you also have the ability to have parallel versions of one file incorporated in multiple releases. Release management, which is an element of SCM, ensures that you can

upgrade your application by incorporating changes in a controlled and validated sequence.

With effective change management, you can trace changes in the application source code back to a specific enhancement in the functional requirements, to an approved design change, or to a specific reported failure in the previous version of the application. You can also prevent degradation of the application between one release and another, by ensuring that untested or unauthorized components are not inadvertently incorporated.

With change management, you also prevent confusion as to who is working on which problems. Confusion of this sort causes inconsistent integration test results and unnecessary rework. For example, it results from two programmers beginning their work on different problems in the same modules, at the same time. The first one finishes, replaces the modules in the test build, and experiences successful results. The second programmer is now unknowingly working with copies of modules that are now out of date. The second programmer also experiences successful test results after replacing the modules in the test build. Now, the first programmer′s changes have been lost, and subsequent testing reveals that what used to work, now fails.

Change management provides you with the ability to report on the exact status of development or maintenance tasks. If change management is implemented well, it also records the type of data that can be used to evaluate the impact that proposed changes may have on the software quality, schedules, or resource requirements.

1.3 Development Efforts Requiring SCM and Change Management

Any application development effort worth doing, deserves some degree of SCM and change management. This is usually self-evident in the case of large and complex application development, but is not always clear to developers of smaller and simpler applications. The size and complexity of the effort determines in which phase SCM and change management are best introduced. Larger projects, having a greater investment at risk, need to be tightly controlled and monitored from the earliest requirements analysis and design phases. Smaller efforts may not require change management or SCM until the applications are complete and delivered for production use. The size of the investment and relevant company or industry standards, also determine the completeness and strictness of the procedures governing SCM and change management. However, SCM and change management are critical to the success of most development efforts, and cost-effective during the maintenance phase of all of them.

1.3.1 Large Application Development Efforts

Consider a large software development effort, such as that of developing and maintaining a major operating system. With a reasonable assortment of application products and operating system code, a typical UNIX** software system might take up 400 MB on disk when installed. It would support dozens of types of hardware peripherals, contain hundreds of end-user commands, include dozens of libraries and application programming interfaces (APIs), and provide a handful of compilers. At some point in time, it probably has a half-dozen releases in the field, and runs on at least three or four hardware architectures. The company developing such an operating system, might employ hundreds of

developers, testers, writers, and Quality Assurance (QA) representatives to handle this development effort.

Such a company cannot risk the loss of profit and sales caused by any confusion in the generation, maintenance, or delivery of its operating system and related software products. A large number of files with a complex development history are required to support this operating system. These are organized in a number of subsystems whose configurations need separate management and tracking. Control over the source and documentation for this product should begin at project conception and continue indefinitely. Communications among the development team about the development status of various application elements should be formalized, and to the greatest extent possible, automated. Project management needs sophisticated methods of measuring and auditing the development process. Quality Assurance has to be exercised over every step of the development process. Clearly, this company could not begin to manage an effort of this scope without very strict and widely-encompassing SCM and change management procedures, policies, and significant automated tool support.

1.3.2 Medium-Sized Application Development Efforts

Now, consider a medium-sized business application development effort, which involves maintaining an old COBOL application on a mainframe, downsizing it to a UNIX platform and adding a GUI, and reimplementing it in C++. This project requires four developers, a project manager, a writer, QA oversight, and end-user testing support. The application itself is not complex or large, but it is critical to the business. It handles customer, order, and payment information for products shipped on a subscription basis. The importance of the application and the facts that there will be at least three application baselines on two operating systems, development activities involving three languages, and use of a “GUI builder” tool, give the effort sufficient complexity, expense, and risk to warrant the use of SCM and change management.

This project requires SCM because the developers need to identify and control multiple versions of the source files and related documentation for multiple, parallel releases of the application. This project also requires automated mechanisms for tracking development status and notifying team members of the transition of various development elements into new baselines. This project requires change management, because technical issues related to building the application appropriately for each platform, while maximizing common code across the platforms, require careful evaluation of proposed changes to baseline releases, design, or requirements documentation. Because the application is critical to the business, it is necessary to ensure that requirements and design decisions are implemented and verified.

1.3.3 Small Application Development Efforts

The value of SCM and change management to smaller SCM development efforts is often underestimated. They are perceived to require a level of effort and formality, which is excessive compared to the total lines of code or the number of developers required to implement a small application. Developers also incorrectly perceive SCM and change management as restraints on their creativity or as impediments to their rapid progress. They fail to recognize how much time and trouble these disciplines can save. But, an application, which is so small that it does not require formal design or independent testing, still requires SCM and change management during its maintenance phase. Because end users come to depend on the availability and function of even small

applications, the software maintenance is inevitabe, because of changes to external interfaces, such as an operating system or utility subroutines, or to the set of requirements met by the application. It is unlikely that the developer responsible for implementing these changes will be the same developer who originally engineered the application. Recording the facts of the build process, identifying the source components, and design and requirements documentation, and having a history of previous changes to the application, contributes to the efficient implementation and test of these changes, no matter how small application.

Often, a company has an assortment of small applications, which collectively constitute a significant investment in a development effort. Over time, these applications develop dependencies on each other, as one application is developed to take advantage of preexisting applications. Using change management to evaluate and monitor changes made to one application can prevent a costly and unforeseen impact on others. Change management also provides a convenient mechanism for collecting the data necessary to evaluate which applications consume disproportionate maintenance resources.

1.3.4 Relationships among SCM, Change Management and Other Development Effort Processes

The “as-built” baseline of an application is the final baseline in the development phase. It consists of source code, data, and executable images, but not all preliminary baselines consist of, or even include, these elements. Which type of objects constitute an application baseline during this phase depends on the choice of development methodology chosen for the project. The milestone events and the types of baselines may vary, but the principles of SCM and change management do not. If a project applies the traditional waterfall methodology, the baselines might be:

- Functional, performance, and interface requirements
- “Build-to” system, subsystem, and component designs
- “As-built” test, integration, and delivery implementations.

In this case, the elements managed by SCM and change management might be files containing:

- Various requirements specifications
- Various forms of design notation and data definitions
- Source and executable application code.

If a newer methodology is employed, such as the object-oriented methodology described by Grady Booch in his *Object-Oriented Analysis and Design With Applications, Second Edition*, the baselines might be:

- Conceptualization prototype
- Analysis description, which models the behavior of the application
- Architectural release and descriptions of tactical policies
- Successively refined executable releases.

The objects controlled by SCM and change management in this case might be files containing:

- Object and class diagrams, finite state machine descriptions, and documentation of nonbehavioral aspects of the design, such as portability, reliability, security, and efficiency

- Class and object structure diagrams and an architectural release, which is a high level source code release of the application in which the details of base classes are not yet defined
- Source and executable application code.

SCM and change management procedures and mechanisms must be tailored to meet the requirements of the chosen development methodology. Both of these processes are complimentary to the goals of the development methodology and related test and QA disciplines in that they help to achieve and maintain a high degree of software quality and engineering process integrity.

1.3.5 Interaction with Project Management

SCM and change management provide feedback and controls to project management. Change management mechanisms provide for the creation, update, and status tracking of changes to the software by all members of a project, including project management. Change management is particularly helpful to project management in making technical decisions, which impact workload distribution, schedules, or definition of the deliverables. Change management creates the opportunity for project management to mandate these factors be identified and evaluated before the decision is made on how or whether to implement an enhancement. It also provides a means by which project management can assign responsibility for the implementation of the approved changes, as well as monitor the progress of these changes through integration and testing into the approved baseline.

Through configuration identification and version history of individual files, SCM helps project management to ensure that developers implement significant technical decisions, as directed. It also provides an audit trail, which facilitates evaluation of procedural flaws that allow incorporation of inadvertent errors into a baselines, when these problems occur.

SCM identifies and relates requirements, design, implementation, documentation, test data, and deliverables. Project management uses this organization of the application itself, to identify and organize the team and effort required to implement the application. Project management manages the effort, establishes the schedules, and defines the milestone events at which various baselines are evaluated and delivered.

SCM and change management are a formal means by which project management exerts and organizes its control over the application development effort.

1.3.6 Interaction with Quality Assurance

SCM and change management processes implement the specific policies and practices adopted by an application development team. The QA organization exists to inspect and verify that the team adheres to these policies, as well as to any externally applied rules and regulations. Formally documenting and automating SCM and change management procedures and practices significantly enhances the job of the QA representative. A QA representative can easily inspect actual practices, procedures, records, and related software libraries to verify that the development team follows the documented practices and use the approved automated tools.

SCM and change management also can be structured to help achieve the QA goals:

- Ensuring conformance to programming standards, such as inclusion of module headers, adequacy of source code comments, and formation of module names according to conventions
- Recording approval of QA representatives that design, code, or documentation has met required criteria before entering the next baseline
- Ensuring related design updates, test results, software quality metrics data, and/or end-user documentation are upgraded and submitted to the library whenever related code changes are accepted.

1.3.7 SCM History and Statistics

If SCM and change management are implemented with a database that provides generalized query capability, then risk, cost, quality metrics, and other data can be captured and analyzed for project management uses also. Various statistics related to project management, software quality SCM and change management can be derived from a well-maintained SCM database. It is possible to extract sufficient data from such a database to project cost, staffing, software sizing, and development schedule data of similar projects under planning.

SCM and change management efforts together form a critical component in the organization of an application development effort. Figure 2 shows how these two processes relate to technical development, quality management, and project management.

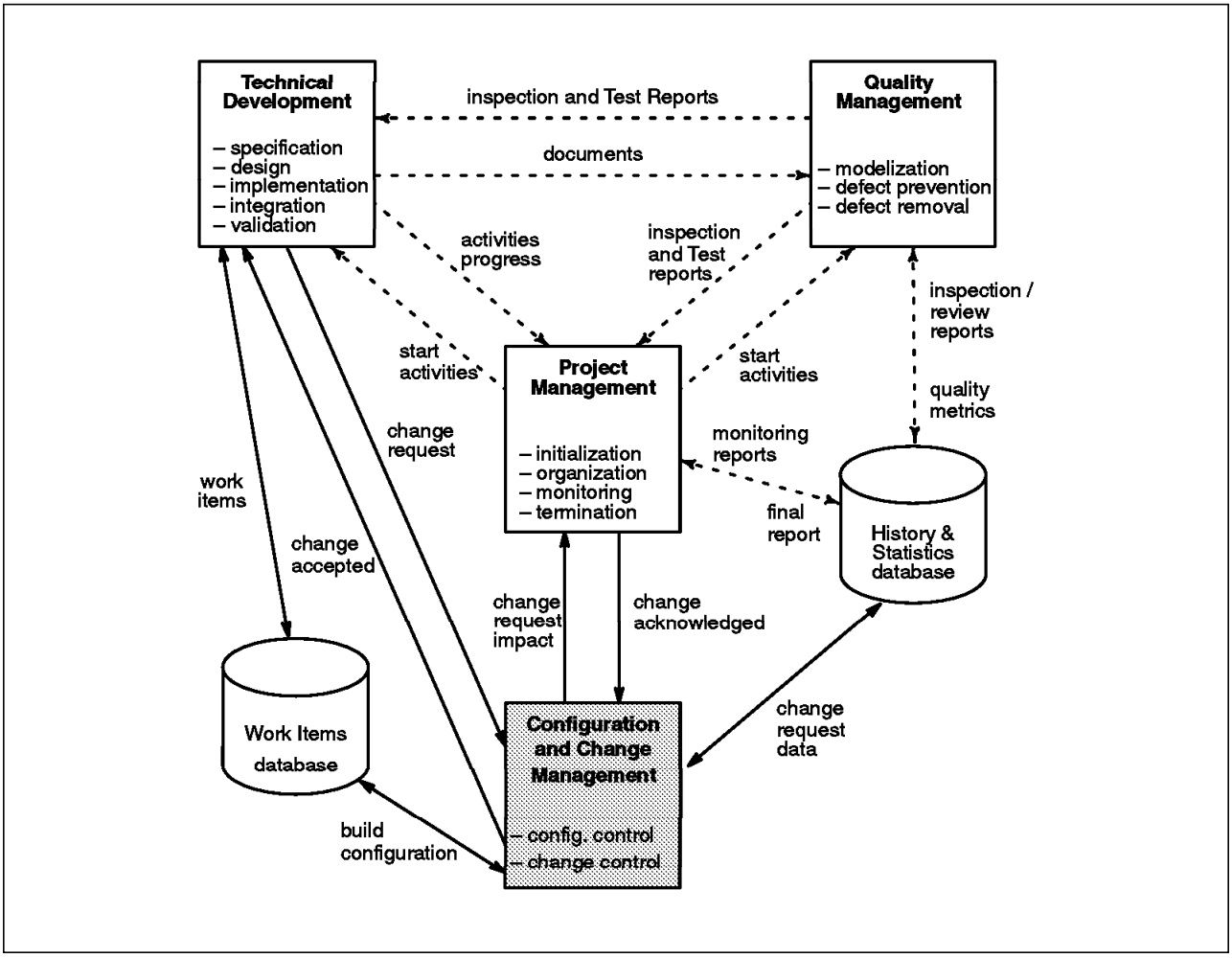

Figure 2. Development Process Relationships

1.4 Automated Support for SCM and Change Management

While procedures and practices to implement SCM and change management can be manual, and in fact were for many years, they lend themselves particularly well to automation. As application development became increasingly complex, it became not only more convenient, but absolutely necessary to automate most of the tasks and procedures supporting SCM and change management. When it became possible for SCM and change management tools to take advantage of relational database technology, data about the configuration objects and change reports could be accumulated and accessed in a variety of ways beyond that necessary merely for SCM and change management purposes if the SCM tools.

When IBM undertook to develop its own UNIX-based operating system, AIX, it discovered it needed a UNIX-based industrial strength SCM and change management system that could support thousands of users, hundreds of Gigabytes of project data, and tens of thousands of report queries daily. IBM needed a product which supported its development methodology, met its QA requirements, and was compatible with its development environment. IBM needed reliability, flexibility, and performance. No one SCM product at that time included all the features, which IBM knew it required for AIX development. Many provided version control or release management, but none integrated automated change management with automated SCM

IBM, therefore, developed its own SCM and change management tool on AIX. This tool, developed for internal use, was called Orbit. After Orbit had been successfully used to bring out several releases of AIX, IBM realized that other software engineering and business application developers could also benefit from a tool with Orbit's capabilities. So, they developed a commercial SCM and change management tool from Orbit and named it Configuration Management Version Control (CMVC).

1.4.1 Configuration Management Version Control

This section gives a brief overview of the features and functions of CMVC.

CMVC is a client-server application. CMVC products execute on the HP-UX** from Hewlett-Packard** (HP**), on SunOS** and Solaris** from Sun** and on AIX/6000. Client portions of these products interoperate with any server portion. There are a command-line client, a stand-alone graphical client, and graphical client, which can be integrated into the IBM Software Development Environment (SDE) WorkBench/6000 or the HP SoftBench** environment. The CMVC server accesses data stored on its host's file system and data stored in a relational database, managed by DATABASE 2/6000 (DB2/6000*), ORACLE**, INFORMIX**, or SYBASE** products. CMVC provides a wide range of functions.

1.4.1.1 Configuration Management

CMVC provides mechanisms for identifying, monitoring, and managing changes made to a software baseline. The baseline may contain any type of data, including: documentation, design and specification data, and build and compile control information, as well as the source code itself. Files managed may contain text or binary data. CMVC supports files containing these types of data by associating them with CMVC "components." Components may be organized

into a component hierarchy to reflect the application's design, responsibilities in the development organization, or other relevant schema. Components are owned and manipulated by CMVC user IDs which are mapped to operating system user IDs, on specific network hosts.

1.4.1.2 Version Control

Version control is provided by standard UNIX Source Code Control System (SCCS), or by PVCS Version Manager**, a product available from INTERSOLV, Inc. Version control ensures that any given version of a file from the present back to its initial version can be identified and retrieved, and that the differences between any two versions can be readily identified. Version control in CMVC applies to both ASCII and binary data files.

1.4.1.3 File Change Control and History

CMVC ensures that an audit trail is maintained for every file by identifying for any file change: when the change occurred, who was responsible, and why the file was modified. If problem tracking is in place, CMVC ensures that all file changes identify the authorizing defect or feature, and that no file changes are allowed without such authorization.

1.4.1.4 Integrated Problem Tracking

Problem tracking, both for feature and defect changes is provided by CMVC. Features and defects are associated with a CMVC component. In addition to describing the enhancement proposed or problem encountered, they identify the specific versions of all controlled files, which implement the feature or defect. Problem tracking implements a configurable process. This means that defects and feature processing can be omitted. If they are used, defect and feature processing can go through a series of states, some of which are optional. Defects and features can be opened, cancelled, returned, or implemented after an optional design, size, and review subprocess is conducted. There is also an optional verify subprocess to verify that the changes were satisfactorily incorporated in a formal release.

1.4.1.5 Release Management

CMVC supports the concept of a "track," which is a mechanism to relate an individual defect or feature with the set of file changes that implement that defect or feature in a given "release" or "level of a release" of an application. Use of tracks is also a configurable process; tracks processing is optional, and if used, has optional subprocesses for approval, fix, and test. If used, tracks go through a series of states which include: approve, fix, integrate, commit, test, and complete.

Releases and levels are CMVC mechanisms for defining interim baselines of the application. CMVC records the exact version of every file comprising the release, including build instructions, and can extract those files into build directories. Release management is a configurable process which can include or omit the track process, and if the track process is employed, an optional level subprocess. A level is a group of changes that are incorporated into a release in a sequential and carefully monitored manner. A level is first in a working state, then tested in an integrated build committed when satisfactorily tested and marked as complete when all changes identified for that level have been successfully incorporated into the release.

1.4.1.6 Access Control

Components provide a mechanism by which CMVC controls access to files under its control. Access of a variety of sorts can be defined for all files associated with a given component. CMVC user IDs implicitly acquire some access authority for components by virtue of owning them, and may inherit other access authority from parent components. They can explicitly grant or deny access authority over components which they own to other CMVC user IDs.

1.4.1.7 Automatic Notification

CMVC provides for automatic notification of CMVC actions affecting particular components and their files to “interested” users. Notification is provided by electronic mail, so a user does not have to start up CMVC to be aware of the CMVC actions. A CMVC user ID′s “interest” in being notified of CMVC actions can be specified in terms of specific CMVC actions and affected components.

1.4.1.8 Customization

CMVC allows additional fields to be added to the database records that implement CMVC features, defects, files, and users. These new fields are reflected by appropriate changes to CMVC windows, reports, and command-line parameters.

CMVC also enables configuration of the processes that manage CMVC objects, such as files, features, and releases. Configuring these processes determines the various states through which these objects can pass.

CMVC allows you to define “user exits” that automatically execute a UNIX shell command file, or user-written executable program whenever specific CMVC commands are executed. You are allowed to select parameter data, related to the CMVC action and object it is affecting, to be passed by the CMVC command to your the shell command file or program. You can also determine if the user exit is triggered before or after the CMVC command executes.

Chapter 2. Discovering CMVC: An New Application Project Is Introduced to CMVC

Suppose for a moment, CMVC is an integral tool in your application development environment. Further, suppose you are on a development team which is about to undertake a new application development project. Your team members would need to learn how to use CMVC to support your development effort and assure its quality. With the help of your system administrator, software configuration manager, and your software Quality Assurance representative, your team members would learn to map the real-life terms and concepts which they already understand to the CMVC vocabulary, objects, and actions. This chapter contains several hypothetical conversations, which would take place under these circumstances. In this chapter, the reader is asked to take the perspectives of different members of this application development team as they learn about CMVC. Perspectives presented include project manager, software engineer, build engineer, and test engineer.

2.1 Project Short Description

Put yourself in the shoes of a project manager who has just been given a project involving maintenance of some existing code and development of new code to meet clearly defined requirements. This section provides a brief description of just such a project, which was undertaken with the help of CMVC during an ITSO residency project. This project is typical of application development efforts in many ways. It is small enough to discuss in a Redbook, yet large enough to illustrate the use of an automated software configuration management (SCM) product, such as CMVC.

The application you will work with is but one in the large legacy of COBOL business applications on which your company depends on daily. This application consists of both batch and interactive programs that support customer ordering and billing for a series of collector′s items, which your company sells. This application makes use of a relational database.

Your project will “downsize” this application from the proprietary operating system and IBM mainframe on which it currently executes, to an open systems operating system, AIX, executing on a smaller, less-expensive, LAN-attached computer; the RISC System/6000. This project is feasible because the smaller platform supports a modern development environment with many application development tools, the same relational database product on which your application depends, the same language (COBOL) in which your application is implemented, and the object-oriented language (C++) in which you would like to implement your future applications.

You have decided your downsizing will be divided into three stages:

1. Migration of the application development environment from the mainframe to AIX. The application development environment during this stage is AIX, although the application will be compiled and executed on the mainframe.
2. Downsizing the application. Both the application development and target execution environments are AIX. First, you will perform a minimum effort port of the COBOL code to AIX creating a graphical user interface (GUI) which looks very much like the mainframe character-based user interface.

Nearly identical AIX and MVS versions of the COBOL application will be maintained, executing in parallel, for a period of time while you evaluate the success of the effort. Second, you will improve the GUI according to modern standards.

3. Object-Oriented (OO) reimplementation of the application. Both the development and execution environments are AIX. In this phase you will reuse the improved GUI design, but implement it as well as the COBOL code in C++.

2.1.1 CASE Environment

Your project requires a different combination of application development tools for each stage:

1. Migrating the application development environment to AIX requires:
   - IBM VS COBOL II* compiler to build the application onto the target mainframe
   - POWERbench COBOL to edit, compile, build, and debug the application on AIX
   - ISPF to generate the 3270 user interface on the MVS platform
   - CMVC/6000 to manage data and control the changes on AIX.
2. Downsizing the application requires:
   - POWERbench C/C++ to edit, compile, and debug the C code on AIX
   - POWERbench COBOL and ToolBox to edit, compile, build, and debug the COBOL code on AIX
   - IBM AIC/6000 to generate C code of the OSF/Motif** graphical user interface for AIX
   - CMVC/6000 to manage data and control the changes on AIX.
3. OO reimplementing the application in an object-oriented language requires:
   - POWERbench C/C++ to edit, compile, and debug the C++ code on AIX
   - IBM AIC/6000 to generate C++ code of the OSF/Motif GUI for AIX
   - CMVC/6000 to manage data and control the changes on AIX.

2.1.2 Hardware and Software Environments

Your project has four RISC System/6000s and one X station connected by a Token Ring Local Area Network (LAN). The AIX machines communicate with each other through the TCP/IP protocol. Also, a MVS mainframe is connected to the LAN. To build the MVS application, all data is uploaded to the mainframe using TCP/IP (**ftp**).

Table 1 shows the distribution of the software services in our network.

Table 1 (Page 1 of 2). Host Names and Associated Software Services

Host Names	Software Services	Assignments
bering	• AIC • POWERbench C/C++ • CMVC Server • CMVC Client • ORACLE RDBMS	Project management and software configuration management

Table 1 (Page 2 of 2). Host Names and Associated Software Services		
Host Names	**Software Services**	**Assignments**
bengal	• POWERbench C/C++ • POWERbench COBOL • CMVC Client • DB2/6000	COBOL DB2 development
yellow	• POWERbench C/C++ • CMVC Client • AIC • DB2/6000	C++ development
sargasso	none	GUI development
zorin1	none	Alternative workstation

Figure 3 shows the entire application development environment with the network, the machines, the software, and the various UNIX login names.

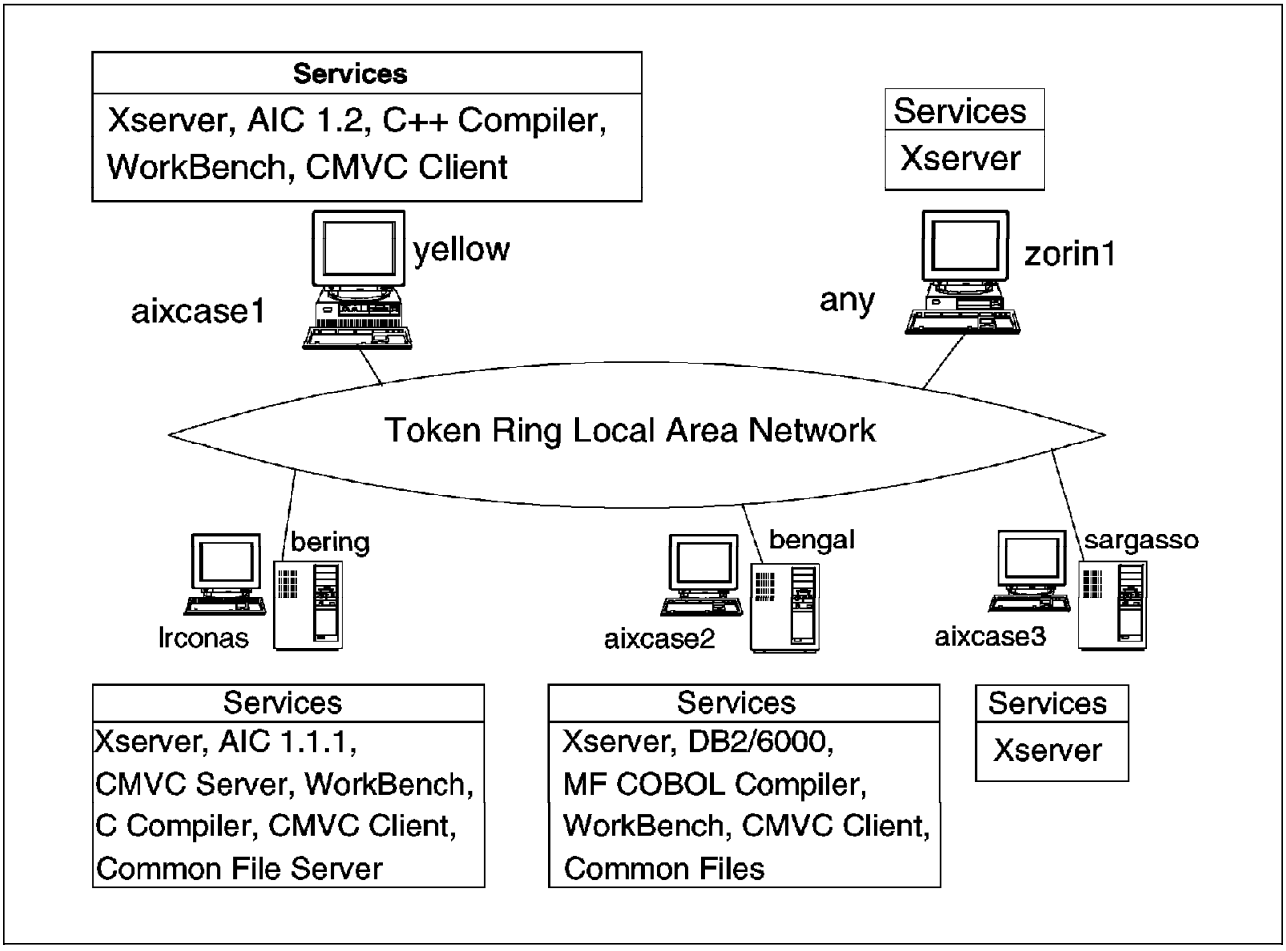

Figure 3. Project Environment

2.1.3 Roles and Responsibilities

Your development team is composed of a few people. Each team member has more than one role during the various development phases. Each time team members change roles, their objectives and work processes are different.

The following gives a short description of the responsibilities for each role in your team.

Role	Responsibility
Project manager	Controls and monitors the project according to a schedule and a budget. The manager assembles the team and the system and ensures that they have the resources necessary to accomplish the project deliverables.
Developer	Designs and implements the application, and performs the unit test. Different developers have separate skill sets. Your development team consists of developers with specific skills in COBOL, DB2, C, C++, and OSF/Motif GUI.
Writer	Writes documentation associated with the application such as end-user manuals and installation manuals.
Builder	Generates drivers and the final application by integrating various parts of the application with each other.
Tester	Verifies that the application implements the requirements as they are specified.

In addition to your team, the company employs people in certain other roles which cross project boundaries for economy, efficiency, and consistency. These roles and a brief description of each are:

System administrator Has several responsibilities:

- Management of the network configuration, such as Internet** address, route, and socket numbers
- Management of the UNIX groups and users
- Maintenance of UNIX data associated with CMVC, such as file systems and databases.

SCM administrator Is in charge of data backup, archives, restoring, and tailoring the SCM environment.

QA representative Verifies that the development methodology is correctly applied and the procedures and practices of both company and project are followed.

We provide a detailed description of the project, the application, and the development environment in Chapter 3, “Overview of the Application Development Project” on page 41, but this introduction provides sufficient background to now begin our conversations about CMVC.

2.2 Project Story

Continue to imagine you are the project manager while you read this section. At any given time, your SCM requirements vary according to your project status and goals, and perhaps your company’s policies. You need to know how CMVC can help you.

2.2.1 Prologue

Your system administrator and SCM administrator know CMVC very well. They have already used CMVC for other projects and know UNIX very well. You and your team members only know the CMVC concepts, but do not have any experience using CMVC.

Your first SCM requirement is to migrate all the source files of the application from MVS to CMVC on AIX. Your second requirement is to organize and control the new development. You plan the implementation of those requirements by:

1. Setting up an AIX environment for CMVC
2. Customizing CMVC according to your company policies, procedures, and practices
3. Starting your CMVC client
4. Using CMVC to organize project data based on various requirements
5. Migrating files into CMVC
6. Building the MVS version
7. Testing the MVS version
8. Freezing the MVS version
9. Starting the AIX version.

The following sections describe each of the above activities through a dialog.

2.2.2 Project Manager Asks about Setting up the SCM Environment

You have a budget, a delivery schedule, and a team. Now, you need some help to implement your SCM requirements for CMVC, so you (PM) contact your system administrator (SA) to create a SCM environment and have the following dialog:

PM: Hi. I am starting a project and I need to use CMVC.
Unfortunately, I do not know CMVC very well; my team doesn't know it either.
The system administration service gave me your name.
Can you help me?

SA: I'm sure I can.
Let's start with my CMVC setup check list.
How many users will use CMVC?

PM: Five people work on my project.

SA: Let me check and see if CMVC has five free network tokens.
Yes, that will be OK.

SA: Will you need to reuse any existing code that is under CMVC control now?
If so, you want to use the CMVC "family" that owns that code.

PM: I do not need to reuse existing code that is in CMVC today.

SA: Do you plan to share your code at a later time with other projects currently using CMVC?
If so, you want to use their CMVC family.

PM: No, I do not need to share code or data with any other project, however I have to develop three different versions of the same application.
Those versions will be sharing code and data among each other.

SA: Fine. We will create a new CMVC family for your project.

PM: My team and I do not know CMVC very well, I would like to be able to play around with CMVC without polluting real project data.

SA: OK! Let's create two different families: one family to manage your application and another for CMVC training purposes. Remember, that CMVC cannot share data among families.

PM: That's OK.

SA: I need a name for each family and it must be less than eight characters long.

PM: Why not use *prod* for the first family, and *dev* for the training family.

SA: Those names are fine, because they are not already used as AIX login names on the CMVC server machine.

SA: Are you using a relational database management system (RDBMS) for your project? CMVC requires one.

PM: We will use DB2/6000 in our application, but we have no database administration experience with it yet. Will you administer the CMVC database for us?

SA: We are already managing an ORACLE RDBMS for other CMVC users. We could expand our existing ORACLE license and support your project with ORACLE, but we have no DB2/6000 skills at present.

PM: Could you migrate our CMVC data from ORACLE to DB2/6000 later when we are more skilled in DB2/6000 administration?

SA: Yes, the CMVC product provides programs to migrate CMVC data from ORACLE, INFORMIX, or SYBASE to DB2/6000.

PM: In that case, let's start CMVC with ORACLE for now.

SA: Now, I need a CMVC user identifier (ID) for the SCM administrator. I also need the host name and login name that your SCM administrator usually uses. This data will be used by CMVC to authorize access to your project's families.

PM: The login name is *lrconas* on *bering*. The CMVC "user ID" could also be *lrconas*.

SA: I will send you a note with the information that will be used to set up and maintain your CMVC families, but let's go over it quickly now while I fill it in:

- Your family names will be *dev* and *prod*
- Your CMVC users will need the following data to access your CMVC families: TCP/IP port number *1221* for *dev* and *1222* for *prod*
- Your SCM administrator *lrconas* will manage your CMVC data from *bering* as *lrconas*
- You will have two ORACLE databases on *bering* with these system database identifiers (SIDs): *DEV* for *dev* family and *PROD* for *prod* family
- One AIX login name will be created on *bering* for each family with the password *sw89ty* for *dev* and *tx5io9* for *prod*.

PM: That looks good! Thank you for your help.

SA: You are welcome.

The following actions follow this conversation:

1. To enable this new CMVC environment, the system administrator configures TCP/IP to allow the CMVC server access from CMVC clients. Figure 4 shows changes made to the file /etc/hosts on the server host. (The changes in this and other files are noted in bold typeface.) Figure 5 illustrates changes made to the file /etc/services on the server host. The changes were made using an editor, such as **vi**.

```
9.113.44.145    bering prod
```

Figure 4. Initializing CMVC Server Access by Creating a Host Name Alias

```
prod  1222/tcp       # prod family for CMVC
```

Figure 5. Initializing CMVC Server Access by Setting up a TCP/IP Port

2. The system administrator creates an AIX login name for the *prod* family, and .profile file for *prod* from a profile template, which according to your company practices is located in the /usr/lib/CMVC directory. Figure 6 shows the AIX commands issued to create the AIX login name.

```
root@bering/>mkuser pgrp=system home=/production prod
root@bering/>passwd prod
prod's passwd:
root@bering/>su - prod
prod@bering/production>cp /usr/lib/CMVC/profile.oracle .profile
```

Figure 6. Creating an AIX Login Name for the Family

Figure 7 shows the .profile file template as it was tailored for *prod*. The words in bold typeface highlight the changes made in the template file for this particular user.

```
export CMVC_HOME=/usr/lpp/cmvc
export CMVC_VCBIN=/usr/bin
export CMVC_VCTYPE=sccs
export ORACLE_HOME=/oracle
export ORACLE_PASS=oracle
export ORACLE_DBA=system/manager
PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin
CMVC_PATH=$CMVC_HOME/bin:$CMVC_HOME/samples:$CMVC_HOME/install

export PATH=$PATH:$CMVC_PATH:$ORACLE_HOME/bin:$HOME/bin
########### Change these variables for your family
export CMVC_FAMILY=prod
export CMVC_SUPERUSER=lrconas
export CLIENT_HOSTNAME=bering
export CLIENT_LOGIN=lrconas
export ORACLE_SID=PROD
################################################
```

Figure 7. Initializing CMVC Environment Variables in the Family AIX Login

3. The system administrator creates the family file system and database. Your company's development methodology dictates that all CMVC customization files are located in the /usr/lib/CMVC directory. Figure 8 on page 18 shows the CMVC server and AIX commands issued to create a family. These commands are issued by the system administrator while logged in as the family AIX login name in that login home directory, after starting up the database *PROD*.

```
prod@bering/production>mkfamily -s
prod@bering/production>cp /usr/lpp/cmvc/install/*.ld .
prod@bering/production>cp /usr/lib/CMVC/config.ld .
prod@bering/production>cp -r /usr/lib/CMVC/configField .
prod@bering/production>mkdb -s
```

Figure 8. Creating Family File System and Database

4. To verify that the creation of the *prod* family was successful, the system administrator starts up one CMVC server daemon and one notification daemon. Figure 9 shows the CMVC server commands issued to start up a family.

```
prod@bering/production>cmvcd prod 2&
prod@bering/production>notifyd&
```

Figure 9. Starting up the CMVC Server

The system administrator checks to see if the CMVC server daemons are running. Figure 10 shows the AIX and CMVC commands issued to see if the CMVC server daemons are running.

```
prod@bering/production>ps -u prod|egrep "cmvcd|notifyd"

 202 12378      -  0:00 cmvcd
 202 25441      -  0:00 cmvcd
 202 27494      -  0:00 cmvcd
 202 28127      -  0:00 cmvcd
 202 44392      -  0:00 notifyd
```

Figure 10. Checking for CMVC Server Daemons

5. The project manager sees he has new electronic mail. After opening the mail, the project manager reads a message from the system administrator:

Note from Your System Administrator

The CMVC families prod and dev are operational.
SCM administrator identified by lrconas may take
control of those families.
The SCM administrator has been defined as CMVC superuser for
both prod,and dev families.
The SCM administrator can access families with the login name
lrconas from the bering host.
The families are located on bering as prod,
and dev login names with the passwords you requested.
The ORACLE databases are identified by SIDs PROD,
and DEV.
After logging in with the family login names, you can access the
ORACLE databases by using the following commands:
sqlplus prod/oracle
for prod family
sqlplus
dev/oracle for dev family
!!!
However, it is advised that you let CMVC manage this data.
Any corruption of the database by use of non-CMVC commands
may result in serious CMVC error conditions!

2.2.3 Project Manager Asks about Implementing Quality Metrics

Continue to put yourself in the place of this project manager. You see you have new electronic mail. You open your mail again, and read a message from your QA representative:

Note from Your Quality Assurance Representative

Your project has been selected to participate in an
experiment, which computes various quality metrics.
Among other data, you will need to provide
the number of lines of code (LOC) developed on your project over time.
The LOC count should be recorded every time you
check in a new version of any file.
At any time, you should be able to retrieve the LOC number
for the most recent version of any file.

A LOC counter program called **"**locCounter**"** has been sent you with the
associated documentation.

For more information don't hesitate to contact me.

You think: "I am not a lucky manager, there 20 different projects in the company and mine is selected." But then, you think that maybe CMVC can help you with this new requirement. To make sure CMVC capabilities can implement the QA representative's requirement, you (PM) decide to contact your SCM administrator (SCMA):

PM: I forwarded you a note which I received from our QA representative.
Do you think that we can do something with CMVC for the
prod family to implement the LOC counting?
Is it possible to record the LOC count every time we check in
a file?

SCMA: That's possible and not too difficult to do!
We could add new CMVC object attributes, for example LOC, by adding a field into the CMVC database record template for the CMVC "File" object.
The CMVC server command **chfield** allows us to customize certain CMVC objects.

PM: Great! After having recorded the LOC count in CMVC, could I make some queries to retrieve this data?

SCMA: Sure! For example, you could query CMVC to get all file names with LOC counts greater than 1789.

PM: That's great! Maybe CMVC can do more?
The best solution would be that the counting and recording of the LOC be done automatically.

SCMA: The LOC field could be updated when creating or checking in files in CMVC.
We could create a user exit program that would be executed automatically.
It would call the locCounter program and then modify the LOC by using the CMVC client command **File -modify**.

PM: That's wonderful! How long will it take to implement such a solution?

SCMA: I think, one full day to develop and test it.

PM: OK! Go ahead. Thanks.

Figure 11 on page 21 shows an example of a user exit program implemented as a Korn shell script. This program is executed by the CMVC **File** command after it has completed the process of either creating or checking in a CMVC file. This program extracts the version of the file just created or checked in, uses the locCounter[1] program to determine a LOC on the extracted file, and updates the *locn* field in this same file's File record. The CMVC **File** command completes execution after the user exit program has completed.

[1] For IBM internal use only, there exists a LOC counting program named SLOCC. It is available on the AIXTOOLS tools disk.

```
#!/bin/ksh
FILENAME=$1
RELEASENAME=$3
SCMA=lrconas
FAMILY=prod
HOMEDIR=/production
## Test if a least two cmvcd daemons are running
## I need two because I use a CMVC command in this UE program
## Otherwise ===> dead lock
if [[ $(ps -u $FAMILY|grep cmvcd|wc -l) -ge 4 ]]
then
    ## Get the file type with the file suffix
    FILETYPE=${FILENAME:##*.}
    ## Extract a copy of the file to count LOC
    File -extract $FILENAME \
         -release $RELEASENAME  \
         -relative $HOMEDIR \
         -become $SCMA \
         -family $FAMILY
    ## Counting the number of lines of code
    LOC=$(locCounter $HOMEDIR $FILENAME $FILETYPE)
    ## Remove temporary file
    rm -f $HOMEDIR/$FILENAME
    ## Test if error while counting LOC
    if [[ $LOC = "error" ]]
    then
        print "The number of lines of code cannot be calculated:"
        print "      File type unknown "
        print "The file has been stored into CMVC with a blank LOC value"
        exit 1
    else
        ## updating the number of lines of code
        File -modify $FILENAME \
             -release $RELEASENAME \
             -locn $LOC \
             -become $SCMA \
             -family $FAMILY
        exit 0
    fi
else
    print "The number of lines of code cannot be calculated:"
    print "      Not enough family daemons are running"
    print "The file has been stored into CMVC with a blank LOC value"
    exit 1
fi
```

Figure 11. An User Exit Program to Count the Lines of Code in a Source File

Appendix B, “Monitoring and Enhancing the Quality of Software with CMVC” on page 175 discusses how CMVC can be used to improve software quality and monitor defect resolution by providing some metrics.

2.2.4 Project Manager Asks about Implementing Project Practices

As project manager, you would also like to implement two company standard practices. The first requires that each source file have a module header with the following project information:

- Copyright
- Product version
- Author
- File version
- Date of latest change.

The second requires that each problem report have a unique identifier. To be sure that the identifier is unique among all families, you want to use an identifier generator program you developed for another project.

The following dialog might occur between you (PM) and your SCM administrator (SCMA) as you try to customize CMVC family to implement the two practices:

PM: For quality metrics, I have seen that it was possible to customize CMVC actions.
Can a user exit program also be used to add a module header to each source file to enforce our company standards?

SCMA: Yes, that is possible.
I have already implemented a similar function for other projects.
The user exit program is executed automatically whenever you place a new file under CMVC control.

PM: I need the module header to contain these items:

- Copyright message with this year indicated
- Product version
- Author's name
- File name
- File version
- Date of latest check-in.

SCMA: No problem! I'll do it in such a way that the file version number and check-in date are automatically updated when a file is extracted to use in an application build, but not when the file is checked out to so you can make changes to it.
Is that OK with you?

PM: Fine! What about problem identifier generation?

SCMA: CMVC automatically generates a numeric problem identifier, such as 1, 2, and 3, for each opened problem report.

PM: I would prefer to have a alphanumeric identifier.

SCMA: OK. Send me your problem identifier generator program, I'll call it from a user exit program that is executed when you open a new problem report.

PM: Thanks a lot.

Appendix C, "User Exit Samples and Suggestions" on page 183 gives examples of two user exit programs. One program inserts a module header into C code source files, and the other generates a problem number automatically.

2.2.5 Project Manager Asks about Starting Up CMVC Client

As project manager, you receive the following electronic note from the SCM administrator.

Note from Your SCM Administrator

```
The customization of both families prod and dev is completed, and
the CMVC family daemons are running.

If you have too long of a response time, please contact me.
I can start up additional "cmvcd" daemons.

Also, please contact me to start up your project.
```

Now that your SCM environment is operational with the right customization (company, project, and QA), you want to start using CMVC yourself. The following conversation might occur between you (PM) and your SCM administrator (SCMA):

PM: You asked me to contact you before I start up my project.

SCMA: Yes, I did.
I have to define your login name as CMVC family superuser ID.
This authorizes you to define your team members yourself.
I need the following pieces of information:

- Your CMVC user ID
- Your mail address
- Your login name and host name.

You should get the same information from each team member you want to authorize to access your families.
Remember that each CMVC user ID must be unique within the family.

PM: Question! Is it possible to have more than one login name on a host or to have more than one host names associated with a login name?

SCMA: Yes, both cases are supported.

PM: Here is the information you asked me for:

- My CMVC user ID should be *projA_lead*
- My electronic mail address is *aixcase4@bering*
- My login name is *aixcase4*
- My host names are *yellow*, *bering*, and *bengal*.

SCMA: I'll send you a note when it is done.

A few minutes later, you receive a notification through electronic mail from both *prod* and *dev* families, stating:

Notification from CMVC

```
A CMVC user ID has been created for you
with the user ID projA_lead.
```

You also receive another note from the SCM administrator:

Note from Your SCM Administrator

By the way, copy the cmvcrc.manager file from the
/usr/lib/CMVC directory into your home directory as .cmvcrc.
That file gives you the CMVC Tasks window customized
according to your responsibilities.
You can change this customization.
Similar files are provided for members of your team performing the
roles of developer, builder, and tester.

When you start up the CMVC client GUI for the first time,
the Set Family dialog box will be displayed.
Fill in the Family field with either dev@bering@1221 or
prod@bering@1222, and fill in the User ID field with projA_lead.
These values will be stored for subsequent CMVC client GUI sessions.

If you use the command-line interface, it is more convenient
set default values for the family and user ID in Korn shell
variables than to type them as parameters on every command line.
Add the following to your .profile file:

```
export CMVC_FAMILY=prod@bering@1222
export CMVC_BECOME=projA_lead.
```

Please forward this note to your team members.
If you have any questions or problems don't hesitate to contact me.

You take the following actions:

1. Forward the latest SCM administrator note to each team member.
2. Get the data necessary to define each of them in CMVC.
3. Login as *aixcase4* on *bering*, copy the cmvcrc.manager file from the /usr/lib/CMVC directory into the home directory as .cmvcrc, then start the CMVC client GUI.
4. Fill in the fields *Family* and *User ID* with prod@bering@1222 and projA_lead, respectively, in the Set Family dialog box.
5. Define the following team members in CMVC: *MVStester*, *MVSbuilder*, *manager*, *krt*, and *truls*. Team member receive notification of the new CMVC user IDs.

2.2.6 Project Manager Asks about Project Organization in CMVC

As project manager, you would like to organize project data according to your project requirements. You know that the CMVC "component" and CMVC "release" can help you to organize your data. But you do not know which components or releases to create, nor do you know how many components and releases your project will require. Fortunately, the SCM administrator is a CMVC expert who can advise you about CMVC component and release usage.

The following dialog might occur between you (PM) and your SCM administrator (SCMA) about how to organize data project with components and releases:

PM: I would like to organize my project data according to
the three phases of my project which are:

1. Migrating the AD platform from MVS to AIX for a legacy application called *productA*.

2. Downsizing the legacy application to AIX, developing a GUI for it that mimicks its ISPF panels, and later modernizing the GUI.
3. Reimplementing the future application by using object-oriented technique.

SCMA: What do you know about CMVC components and releases?

PM: I've read the concept manual, but I don't know how to use them for my project. Can you help me?

SCMA: From this brief description, I can highlight some key words: "productA," "legacy," "MVS," "AIX," and "future." I can infer that the *productA* consists of a legacy and future application. Moreover, the legacy application is composed of MVS and AIX implementations. These relationships suggest a top-level component hierarchy.

PM: So, you suggest I create a component for *productA* with two children components, *legacy* and *future*. Then, I create two children components below *legacy* named *AIX* and *MVS*. Now, I "manage" the source files for these different implementations in the separate components. How else can this component structure help me to manage my project?

SCMA: I presume that you have different people responsible for the MVS, AIX, and future object-oriented implementations of your application. You can have these people own and control the specific components associated with their separate responsibilities.

I also suppose you would like to communicate information to different people according to their role in your project. Each component has a separate "notification list" that ensures CMVC will inform team members other than the component owner about the actions performed on data managed through the component.

Each component also has an "access list" that lets the component owner grant authority to other project members to access or manipulate the data managed by the component.

Both of the access list and notification lists are inherited by components at lower ranks in the component hierarchy from the components above. You can use combinations of these lists along with ownership assignments of subsidiary components to support the work groups on your project.

PM: So, it sounds like a developer who works on both AIX and MVS source code, but is responsible for neither can be given authority to check in and out files managed by either component, or the builder who owns none of these components, can be informed automatically whenever a new file is created in any of them. This seems very useful.

SCMA: About the CMVC releases... You could create one for each pproduct version:

- *MVS_Release_0* managed by the *MVS* component
- *AIX_Release_1* managed by the *AIX* component
- *OO_Version_1* managed by the *future* component.

PM: What is the advantage of associating each release with a different component, instead of having them all managed by the *productA* component?

SCMA: Each version can be governed by a different build process and managed by different people.
Do you know which "processes" you want to prevail over your components and releases?
They determine how problem tracking and release integration are performed.

PM: I don't know.
What I'd like to see is the ability to file problem reports or change requests, and track their progress until they are resolved.
I'd like to be sure that all proposed changes are either implemented and integrated into the releases of the application, or rejected.
I'd also like to be confident that the only changes getting into the releases are required by an approved problem report or change request.
I also want to be able to incorporate changes into the release in an orderly and manageable manner and record their success or failure during testing.
What are your suggestions?

SCMA: I think your requirements would be fulfilled by selecting the "maintenance process" for both components and releases.
This combination does not define the most elaborate development process, but should offer sufficient controls for your needs.
Selecting the maintenance process on a component ensures that:

- Any one can "open" a defect or feature.
- The component owner can "accept" or "reject" it.
- The defect or feature creator can "verify" that it was resolved satisfactorily.
- The defect or feature will be "closed" when all is done.

Selecting the maintenance process on a release will ensure that you can:

- Link changes to controlled files with the relevant defects or features, and the releases in which these changes will be incorporated
- Enable component owners to approve file changes implementing fixes which resolve defects or features before the files are incorporated into a release
- Enable the orderly incorporation of "fixes" into a release on a defect or feature basis.

This process also requires you to specify a tester name and a test environment, such as a target platform to create a CMVC "environment list" for each release.
Then, "test records" will be created to ensure that those releases are tested for each appropriate environment.

PM: But can I change my mind about these process selections later?

SCMA: You can change them under appropriate circumstances, and you can also use different choices when creating new releases and components later.

PM: Thanks.
I am now ready to populate my family with components and releases.

After creating your component hierarchy and your releases, you open a defect on the *MVS* component to authorize bringing the MVS source files under CMVC control. CMVC automatically sends a note to the MVS developer, who is the owner of the *MVS* component, about the opening of that defect.

2.2.7 Project Manager Asks about Reporting

As project manager, you would like to generate a report identifying the decisions you are implementing with CMVC. The following dialog might occur between you (PM) and the SCM administrator (SCMA) on how to query CMVC to extract data:

PM: I would like to generate a report containing these items:

- User list with CMVC user ID, name, and area
- Component list with name, owner's CMVC user ID, and associated change process
- Release list with name, component name, associated problem tracking process, and owner's CMVC user ID
- Host list with CMVC user ID, login name, and host name
- Access list with component name, member's CMVC user ID, and granted authority
- Notification list with component name, member's CMVC user ID, and interest
- Environment list with test environment, release name, and tester's CMVC user ID.

SCMA: I am going to send you by mail, a set of shell scripts
that allow you to get this information.
These make use of the CMVC command-line interface command, **Report.**
This command makes SQL-like queries the ORACLE database in which
CMVC **"meta-data"** or data about CMVC objects such as users, files,
releases, components, defects, and features is stored.
If you would like to make other reports, you should look at
the views.db and tables.db files located in the
/usr/lpp/cmvc/install directory to learn about the CMVC
tables and views created in the ORACLE database.
Don't forget to initialize the
CMVC environment variables *CMVC_FAMILY* and *CMVC_BECOME*
used by the CMVC command, **Report.**

PM: Great! Thanks!

The following illustrations show the how this reporting requirement is implemented:

1. Figure 12 on page 28 shows how you set the required environment variables before issuing the command shown in Figure 13 on page 28.

```
export CMVC_FAMILY=prod@bering@1222
export CMVC_BECOME=projA_lead
```

Figure 12. Setting up CMVC Environment Variables

2. Figure 13 shows the CMVC **Report** command, which identifies your project users and their CMVC IDs.

```
printf "%-20.20s %-25s %s\n" "User ID" "Real Name" "Area";\
Report -view users \
-where "dropDate is null order by login" -raw |
awk -F'|' '{printf "%-20.20s %-25s %s\n", $1,$2,$3}'|
sed "/InheritedAccess/d"
```

Figure 13. How to Get a User List with the Report Command

3. Figure 14 shows the list of users obtained by running the command shown in Figure 13.

```
User            Real Name               Area

MVSbuilder      Richard Kortmann        MVS_Building
MVStester       Leif Trulsson           MVS_Testing
branko          Branko Peteh            Future-ProductA
krt             Richard Kortmann        AIX-ProductA
lrconas         Lorna Conas             SW_Config_Mgmt
manager         Department Manager      Production
projA_lead      Project A Lead Engineer ProjectA
truls           Leif Trulsson           MVS-ProductA
```

Figure 14. User List from Report -view users Command

4. Figure 15 shows the CMVC **Report** command, which identifies your components, their owner's CMVC user ID, and the CMVC change process governing them.

```
printf "%-20.20s %-15s %s\n" "Component" "Owner" "Process";\
Report -view compView \
-where "userId like '%' order by name" -raw |
awk -F'|' '{printf "%-20.20s %-15s %s\n", $1,$2,$9}'
```

Figure 15. How to Get a Component List with Report Command

5. Figure 16 on page 29 shows the list of components obtained after having executed the command shown in Figure 15.

```
Component               Owner           Process

AIX                     krt             maintenance
MVS                     truls           maintenance
future                  projA_lead      maintenance
legacy                  projA_lead      maintenance
productA                projA_lead      maintenance
```

Figure 16. Component List from Report -view compView Command

6. Figure 17 shows the CMVC **Report** command, which identifies the releases managed by the components, the CMVC problem tracking process, and CMVC user ID governing them.

```
print f "%-20.20s %-15s %-15s %s\n" \
"Release" "Component" " Process" "Owner";\
Report -view releaseView \
-where "userId like '%' order by name" -raw |
awk -F'|' '{printf "%-20.20s %-15s %-15s %s\n", $1,$2,$3,$4}'
```

Figure 17. How to Get a Release List with Report Command

7. Figure 18 shows the list of releases with their main attributes. You get this list by running the command shown in Figure 17.

```
Release         Component       Process         Owner

AIX_Release_1   AIX             maintenance     krt
MVS_Release_0   MVS             maintenance     MVStester
OO_Version_1    future          maintenance     lrconas
```

Figure 18. Release List from Report -releaseView Command

8. Figure 19 on page 30 shows a shell script with several CMVC **Report** commands. This script generates a comprehensive report describing the:
   - Host list
   - Access list of each component
   - Notification list of each component
   - Environment list of each release.

```
#!/bin/ksh
family=$1
userID=$2
fileOut=$3
###Getting logins and hosts for each users
printf "%-20.20s %-15s %s\n" "User ID" "Login" "Host">$fileOut
Report -view HostView -raw -family $family -become $userID \
-where "userId like '%' order by login" |
awk -F'|' '{printf "%-20.20s %-15s %s\n",$3,$1,$2}'>>$fileOut

###Getting work groups for each component
printf "%-20.20s %-15s %s\n" "Component" "User ID" "Authority">>$fileOut
Report -view AccessView -raw -family $family -become $userID \
-where "compId like '%' order by compName" |
awk -F'|' '{printf "%-20.20s %-15s %s\n",$1,$2,$5}'>>$fileOut

###Getting distribution lists for each component
printf "%-20.20s %-15s %s\n" "Component" "User ID" "Insterest">>$fileOut
Report -view NotifyView -raw -family $family -become $userID \
-where "compId like '%' order by compName" |
awk -F'|' '{printf "%-20.20s %-15s %s\n",$1,$2,$6}'>>$fileOut

###Getting test environments and associated testers
### for each release
printf "%-20.20s %-15s %s\n" "Environment" "Release" "Tester">>$fileOut
Report -view EnvView -raw -family $family -become $userID \
-where "userId like '%' order by releaseName"  |
awk -F'|' '{printf "%-20.20s %-15s %s\n",$1,$2,$3}'>>$fileOut
```

Figure 19. Shell Script to Get CMVC Lists

9. Appendix H, "Tailoring CMVC Windows for Different Types of Users" on page 217, describes an example of a customization of the CMVC - Tasks window for a project manager. It allows you to get this same information by simply clicking twice on certain items presented in the Tasks List.

2.2.8 Developer Asks about Components and a Release for the Original MVS Baseline

Take the MVS developer perspective on this project for a moment. As a result of actions taken by others, you have received several notes from the family AIX login name, *prod*, telling you:

1. The CMVC user ID, *truls*, has been created for you.
2. The *MVS* component has been created as a child of the *legacy* component, by *projA_lead*. You are now the owner of this component.
3. An access list member with *general* authority has been created and associated with the *productA* component for the *truls* user by *projA_lead*. This insures you will inherit *general* authority over all components created for your project.
4. The *MVS_Release_0* release has been created and associated with the *MVS* component by *projA_lead*. You are also the owner of this release.
5. The CMVC *prod_00001* "defect" has been opened by *projA_lead* on the *MVS* component, for the *MVS_Release_0* release with the abstract "MVS Pre-Migration Baseline" and the following comment:

Defect prod_00001 Comments

```
This first defect against MVS component will bring under CMVC
control the original MVS source code of the legacy application.
```

As owner of the *MVS* component, you are the owner of this defect.

The project manager informs you to bring the source files from MVS to AIX and place them under control of CMVC and if you have problems, you may contact your SCM administrator.

You know that to create a file in CMVC, you must reference a component and a release. However, you have several types of files and you are hesitant about putting them all together in just one component. The following dialog might occur between you (DEV) and the SCM administrator (SCMA):

DEV: I work on the *productA* project and I have to bring files under CMVC control. I am the owner of the *MVS* component and the *MVS_Release_0* release.

SCMA: Are your files presently under SCCS control? If so, use the import procedure shipped with CMVC.

DEV: No, they are not! These are MVS files. Anyway, I would like to classify the files according to the type of data they contain, such as COBOL source code and ISPF panel definitions.

SCMA: You could create a component for each file type. You may create children components of the component *MVS* because you are its owner.

DEV: OK. I'll create the following components to manage my files:

- *COBOL*
- *JCL*
- *CLIST*
- *ISPF*
- *Assembler*.

SCMA: By the way, for your project, you have to create your components specifying the *maintenance* process.

DEV: Thanks!

As a result of this conversation, you:

1. Login as *aixcase2* on *bering*, and copy the cmvcrc.developer file from the /usr/lib/CMVC directory into the home directory as .cmvcrc.
2. Create a directory source/cobol in the home directory to store the files coming from MVS, then, you download the files from MVS into the directory source/cobol.
3. Start the CMVC client GUI. The CMVC Tasks window opens, and a Set Family dialog box is displayed. Fill in the two fields, *Family* and *User ID*, with prod@bering@1222 and truls, respectively.
4. Create the components *COBOL*, *ISPF,CLIST*, *assembler*, and *JCL* as children of the *MVS* component with the *maintenance* process.

You are now ready to create the CMVC files in reference to these components and the *MVS_Release_0* release. Figure 20 on page 32 shows the component hierarchy you would have created, thus far.

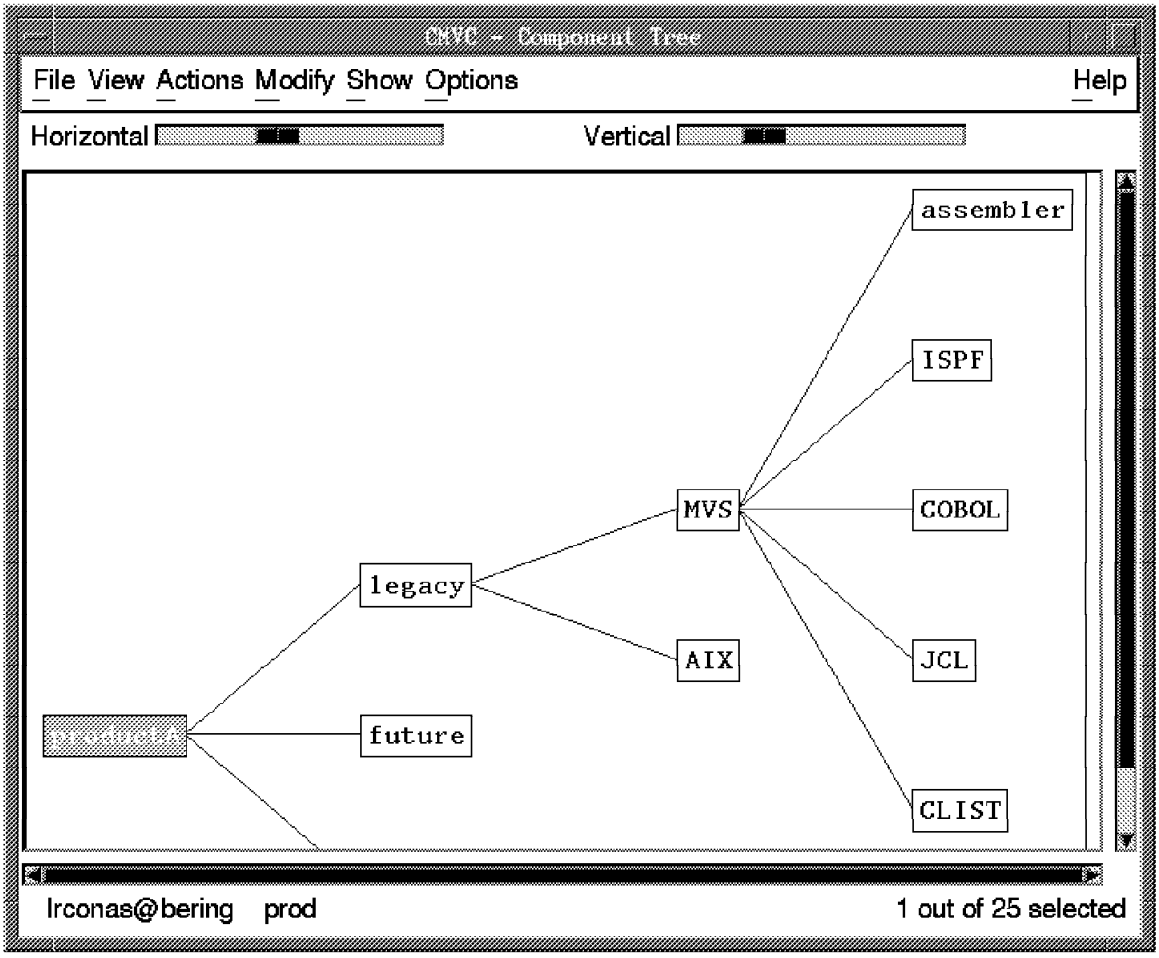

Figure 20. Initial Component Hierarchy

2.2.9 Developer Asks about Bringing MVS Source Files under CMVC Control

Continuing to look at this from the MVS developer's perspective, you now want to create the file, cobol/ibmbsel.cob. You try to create this file, managed by the component *COBOL* and belonging to the release, *MVS_Release_0*, using the CMVC client GUI. Unfortunately, the following error message comes up:

CMVC Error Message

```
0010-258 The requested action requires that you specify one or
more defects or features.
0010-263 File cobol/ibmbsel.cob associated with release
         MVS_Release_0 cannot be created.
```

You do not understand this message, so you call the SCM administrator for help. The following dialog might occur between you (MVS DEV) and the SCM administrator (SCMA):

MVS DEV: I got an error message when I tried to create a file: "The requested action requires that you specify one or more defects or features.." What should I do?

SCMA: Did you receive a notification about the opening of any defects or features?

MVS DEV: Yes, I did!
The message said defect, *prod_00001*, was opened on my component, *MVS*.
The defect comments said I should bring the MVS files under CMVC control.
That was what I was trying to do when I got this error message.

SCMA: You need to create a "track" to identify this defect with the act of creating a new file in the release.
To do so:

1. Accept the defect.
2. Create a track naming both this defect number and *MVS_Release_0*.
3. Now create the file in the release.
4. CMVC automatically creates a "fix" record for each component affected by this track.
5. Complete the fix records, and CMVC will notify the right people based on the CMVC object ownership and the notification lists.

MVS DEV: Why do I have to do all those things?

SCMA: Because your project manager has chosen the *maintenance* process for the release *MVS_Release_0*.
If your manager had chosen the *no_track* process,
you could create your file without using a track and fix records.

MVS DEV: By the way, do you have a procedure to create the files in batch mode?

SCMA: Yes, I have. I'll send it to you. Before you execute this command, you should initialize these CMVC environment variables with values, such as these:
CMVC_FAMILY=family name
CMVC_BECOME=your CMVC user ID
CMVC_RELEASE=your release
CMVC_TOP=the file source directory.

MVS DEV: Thanks.

As a result of this conversation you:

1. Accept the *prod_00001* defect.
2. Create a track for that defect in the *MVS_Release_0* release.
3. Initialize the CMVC environment variables, as shown in Figure 21.

```
export CMVC_FAMILY=prod@bering@1222
export CMVC_BECOME=truls
export CMVC_RELEASE=MVS_Release_0
export CMVC_TOP=/u/aixcase2
```

Figure 21. CMVC Client Environment Variables Set up

4. Execute the SCM administrator's command for each type of file. Figure 22 shows the command to bring a group of files under CMVC control.

```
find ./source/cobol -type f -name '*.cob' -exec \
File -feature prod_00001 -component COBOL -create {} \;
```

Figure 22. Commands to Bring a Group of Files under CMVC Control

5. Verify the success of the file creation using the CMVC client GUI. Display the file list of each component by typing the component name in Component field

in the Open File List CMVC window. See if the LOC count has been filled in for each file, and if the module header has been inserted at the beginning of each file.

6. Indicate to CMVC that you are done with the work associated with the *prod_00001* defect:
   a. Click twice on the line **List defects for which I have changed files** in your customized CMVC - Tasks window.
   b. After the CMVC - Fix Records window is displayed, component by component, highlight the fix records, and then select **Complete...** from the Action menu. CMVC automatically sends a note to the originator of this defect, indicating that it has been fixed.

Appendix H, “Tailoring CMVC Windows for Different Types of Users” on page 217 describes an example of a customization of the CMVC - Tasks window for a developer that facilitates fixing defects.

2.2.10 Builder Asks about Building Application on MVS

Now, imagine you are the MVS builder on your project. You have received notes, from the family AIX login name, *prod*, telling you:

1. An access list member with *builder* authority has been created and associated with the *MVS* component for the *MVSbuilder* CMVC user ID.
2. The *prod_00001* defect has been fixed for the *MVS_Release_0* release.

The following dialog might occur between you (BUILDER) and the SCM administrator (SCMA) discussing how to build the *MVS_Release_0* release:

BUILDER: I work on the *productA* project and I have to build the MVS implementation of it. I have the *builder* authority for the *MVS* component that manages the *MVS_Release_0* release. I would like to make a baseline consisting of the MVS files I just put under CMVC control with reference to the *prod_00001* defect. I want to be able to extract and rebuild this baseline at any time.

SCMA: Because the *MVS_Release_0* process is *maintenance*, you should create a CMVC “level” to integrate the defect *prod_00001*. Then you will be able to extract that level of the *MVS_Release_0* release. All the files created with reference to the *prod_00001* defect, will be automatically extracted from CMVC.

BUILDER: Does that mean the level allows me to freeze the release according to a collection of changes?

SCMA: Correct.

BUILDER: Can I extract the level up to the MVS machine?

SCMA: I know that TCP/IP and Network File System** (NFS**) both are supported on MVS. You can extract the level onto any NFS file system, or you can use **ftp** or **hcon** to upload the files onto the mainframe.

BUILDER: OK, thanks.

Based on this conversation, you:

1. Login as *aixcase2* to *bengal*, copy the cmvcrc.builder file from the /usr/lib/CMVC directory into your home directory as .cmvcrc, and then start

the CMVC client GUI. The CMVC - Tasks window opens and a Set Family dialog box is displayed. Fill up the two fields, *Family*, and *User ID* respectively, with `prod@bering@1111` and `MVSbuilder`.

2. Create the *0* level for the *MVS_Release_0* release. Fill in the Type field with *production* because this level will be shipped to your customers. CMVC does not use the contents of the Type field to control any of its processing. The meaning of this field, and the range of acceptable values, is up to your SCM administrator to define. For example, the *type* field could help you to identify those levels which you would like to archive. You would have a default value of `production` given to all levels as they are created. When a level is no longer current, if you change the value to `obsolete`, then you can have automatic archival occur nightly on all levels with that *type* value. Appendix D, “Hints and Tips for Using CMVC” on page 193 describes how to manage your obsolete levels to free storage space on the CMVC server host.
3. Integrate the track associated with the *prod_00001* defect and the *MVS_Release_0* release into the *0* level. The track now becomes a CMVC “level member” of the *0* level.
4. Create the extraction target directory `ProductA/MVS_Release_0` in the `/ad` file system, which was created and exported by your system administrator. You can now extract the *0* level to the `/ad/ProductA/MVS_Release_0` directory located on *bengal*.
5. After checking that all the files of the original legacy application have been extracted to the right target directory, upload the files of the *MVS_Release_0* release from `/ad/ProductA/MVS_Release_0` to the target MVS machine. Then build the application on MVS.
6. Check the execution of the build. If there were no compile nor link-edit errors, you can freeze the *0* level. Click twice on the line **Show all my levels with the state integrate** on your customized CMVC - Tasks window. The CMVC - Levels window is displayed. You can now select the *0* level of the *MVS_Release_0* release. Select **Commit...** from the Action menu, then select **OK** in the Commit Levels dialog box. Select **Complete...** from the Action menu, and then select **OK** in the Complete Levels dialog box.

Now, the *0* level of the *MVS_Release_0* release is a baseline; it can always be re-created exactly as it is now.

Appendix H, “Tailoring CMVC Windows for Different Types of Users” on page 217 describes an example of a customization of the CMVC - Tasks window for a builder to automate the build process.

2.2.11 Testing the Application

Imagine you are now the MVS tester. You have received several notes, from the family AIX login name *prod*, telling you:

- A notification list member with *tester* authority was created and associated with the *MVS* component for the *MVStester* user.
- An environment list member, *MVS_legacy*, with the *MVStester* tester was created and associated with the *MVS_Release_0*. This record identifies a test environment in which you will test the application.
- The *0* level associated with the *MVS_Release_0* release was committed and completed by the *MVSbuilder* CMVC user ID. The level is now ready for test.

- A test record for the track identified by the *MVS_Release_0* release and *prod_00003* defect has been created for the *MVS_legacy* environment which is your responsibility.

On receipt of the last message, you:

1. Login to the MVS mainframe and successfully execute the test cases for the *productA* application.
2. Login as *aixcase2* on *bengal*, and copy the `cmvcrc.tester` file from the `/usr/lib/CMVC` directory into the home directory as `.cmvcrc`. Start the CMVC client GUI. The CMVC Tasks window and Set Family dialog box are displayed. Fill in the two dialog box fields, *Family*, and *User ID*, respectively, with `prod@bering@1111` and `MVStester`.
3. Click twice on the line **Show all fixed defects I have to test** in the customized CMVC - Tasks window. When the CMVC - Test Records window is displayed, highlight the test record and select **Accept...** from the Action menu.

2.2.12 Concluding the Migration

Imagine you are the project manager again. You receive a note from *prod*, the family AIX login name, indicating that the *MVS_Release_0* has been tested successfully on the MVS mainframe. The project manager takes these final actions:

1. Login to the MVS mainframe to verify the behavior of the *productA* application. The application looks good.
2. Login as *aixcase4* on *bering*, then start the CMVC client GUI. The CMVC Tasks window is displayed.
3. Click twice on the line **Show all defects that I have to verify** in the customized CMVC - Tasks window. When the CMVC - Verification Records window is displayed, highlight the verification record and select **Accept...** from the Action menu. The *prod_00001* defect is now closed.

2.2.13 Project Manager Asks about Sharing Files with AIX Release

The original MVS version of the application is now under CMVC control. You would like to reuse all the existing files for the AIX implementation of the application, which will be identified to CMVC as the *AIX_Release_1* release.

The following dialog might occur between the you (PM) and the SCM administrator (SCMA) concerning reuse of the files in the *MVS_Release_0* release:

PM: I would like to reuse files from the *MVS_Release_0* release in the *AIX_Release_1* release.
Over time, I expect that I may need two different versions of some of the files, and that I will probably add new files to the AIX release.
How can I do this?

SCMA: First, you should link the *MVS_Release_0* files to the release *MVS_Release_0* by performing CMVC release link action.
After the linking, the files are "common" to both releases.
This means both releases are composed of the exact same version of every file.
Later, you can create a new version of a file for only the *AIX_Release_1* release.

You would check the file out, make changes to it, and when checking that file in, you select **Break common link** in the Check In Files dialog box. After the check-in, the *MVS_Release_0* file version number will remain 1.1, but the *AIX_Release_1* release's file version will be 1.2. In other words, CMVC created a new branch in the file version tree for this one file. In CMVC terminology, this file is now "shared" between two releases.

PM: Should I create a defect to perform the release link action?

SCMA: Yes, you should, because the *AIX_Release_1* also has the *maintenance* component process.

PM: Thanks.

Different team members play a role in this process, so CMVC automatically notifies them when their actions are required:

1. You open a new defect on the *AIX* component, for the *AIX_Release_1* release. The user exit program generates the defect identifier *prod_00002*.
2. CMVC notifies the AIX developer, who is the owner of the component *AIX*, about the opening of the *prod_00002* defect. The AIX developer accepts the *prod_00002* defect.
3. The AIX developer creates a track associated with that defect and the *AIX_Release_1* release.
4. The AIX builder links the *MVS_Release_0* release to the *AIX_Release_1* release, specifying this track.
5. You tell team members to start development activities on the *AIX_Release_1* release. This results in new versions of some of the files because they are updated only for this release.

2.2.14 Project Manager Asks about Audit Log of CMVC Commands

So far, you and other team members have performed a lot of CMVC actions. As project manager, you realize it may be necessary to examine an audit log, which documents all the CMVC actions executed for your family over a period of time. Again, you (PM) call your SCM administrator (SCMA) to discuss an audit log:

PM: I'd like to know who issued which CMVC commands, and when they did so.

SCMA: Click twice on the line "Show CMVC activity log file" in your customized CMVC - Tasks window. This action displays a CMVC - Information window, which contains the contents of an audit log file called log, located in the /production/audit directory. It contains a record of all the CMVC actions taken against the *production* family data.

Figure 23 on page 38 contains an example of the contents of this audit log file. Each entry in this file identifies the following data items for a given CMVC action:

1. Transaction type, such as *FileCreate*, *DefectOpen*, or *CompCreate*
2. Transaction status, either *SUCCESS*, *FAILURE*, or *UNAUTHORIZED*
3. Date and time
4. CMVC user ID
5. Login name at host name

6. Additional information, such as whether a transaction was successful or unsuccessful, and an error message, if one was issued.

```
UserCreate,SUCCESS,10/05/93,9:45:12,lrconas,lrconas,bering,projA_lead
HostCreate,SUCCESS,10/07/93,9:21:25,lrconas,lrconas,bering,projA_lead,bering,aixcase4
CompCreate,SUCCESS,10/07/93,10:23:27,projA_lead,aixcase4,bering,productA
AccessCreate,SUCCESS,10/07/93,11:02:45,projA_lead,aixcase4,bering,krt,legacy,releaselead
NotifyCreate,SUCCESS,10/07/93,11:31:36,projA_lead,aixcase4,bering,krt,MVS,low
ReleaseCreate,SUCCESS,10/07/93,12:53:52,projA_lead,aixcase4,bering,MVS_Release_0
EnvCreate,SUCCESS,10/07/93,12:16:17,projA_lead,aixcase4,bering,MVStester,MVS_Release_0,MVS_legacy
DefectOpen,SUCCESS,10/07/93,14:23:00,projA_lead,aixcase4,bering,1
DefectModify,SUCCESS,10/07/93,14:23:10,projA_lead,aixcase4,bering,1
CompCreate,SUCCESS,10/08/93,8:00:13,truls,aixcase2,bering,COBOL
DefectAccept,SUCCESS,10/08/93,9:13:56,truls,aixcase2,bering,prod_00001
TrackCreate,SUCCESS,10/08/93,11:00:12,truls,aixcase2,bering,prod_00001,MVS_Release_0
FixComplete,SUCCESS,10/09/93,9:47:45,truls,aixcase2,bering,prod_00001,MVS_Release_0,COBOL
LevelCreate,SUCCESS,10/09/93,13:34:15,MVSbuilder,aixcase2,bengal,0,MVS_Release_0
MemberCreate,SUCCESS,10/09/93,13:35:45,MVSbuilder,aixcase2,bengal,0,prod_00001,MVS_Release_0
LevelExtract,SUCCESS,10/09/93,13:55:23,MVSbuilder,aixcase2,bengal,0,MVS_Release_0
LevelCommit,SUCCESS,10/09/93,17:56:42,MVSbuilder,aixcase2,bengal,0,MVS_Release_0
LevelComplete,SUCCESS,10/09/93,17:58:13,MVSbuilder,aixcase2,bengal,0,MVS_Release_0
TestAccept,SUCCESS,10/16/93,10:34:56,MVStester,aixcase2,bengal,prod_00001,MVS_Release_0,MVS_legacy,MVStester
VerifyAccept,SUCCESS,10/17/93,18:45:48,projA_lead,aixcase4,bering,prod_00001,projA_lead
DefectOpen,SUCCESS,10/18/93,9:37:45,projA_lead,aixcase4,bering,2
DefectModify,SUCCESS,10/18/93,9:37:50,projA_lead,aixcase4,bering,2
DefectAccept,SUCCESS,10/19/93,13:56:32,krt,aixcase3,yellow,prod_00002
TrackCreate,SUCCESS,10/19/93,9:45:25,krt,aixcase3,yellow,prod_00002,AIX_Release_1
ReleaseLink,SUCCESS,10/20/93,16:17:34,AIXbuilder,aixcase3,bering,MVS_Release_0,AIX_Release_1
```

Figure 23. CMVC Family Log File Example

2.2.15 Epilogue

The project manager and team have taken a series of CMVC actions to set up a CMVC family for this project and organize CMVC control over the application development effort. CMVC notifications communicated these CMVC actions to appropriate team members, in some cases reminding them that they should act on the notification. These actions are summarized below:

1. The SCM administrator created a CMVC user ID with superuser authority for your project.
2. The project manager organized the project by creating:
   a. One CMVC user ID for each team member
   b. A host list for each CMVC user ID
   c. Top-level components with access and notification lists
   d. Releases with an environment list.

   The project manager also opened the *prod_00001* defect, attached to the *MVS* component, to authorize controlling the MVS source code under CMVC.
3. The MVS developer took the following steps to control the original MVS source files:
   a. Created children components for the *MVS* component with which to control the various types of MVS source files
   b. Accepted the *prod_00001* defect
   c. Created a track, linking the *MVS_Release_0* release with the *prod_00001* defect
   d. Created the new CMVC files under authority of this track

e. Informed the rest of the team that the *prod_00001* defect in the *MVS_Release_0* release was fixed.

4. The MVS builder took the following steps to create the original MVS baseline:
   a. Created the *0* level for the *MVS_Release_0* release with which to integrate files associated with the *prod_00001* defect
   b. Extracted the *0* level of the *MVS_Release_0* release to a build directory for testing
   c. Committed the level after evaluating it
   d. Completed the release.
5. The MVS tester accepted the test of the *MVS_Release_0* release for the *MVS_legacy* environment.
6. The project manager verified that the *prod_00001* defect was resolved satisfactorily.
7. The project manager then opened the *prod_00002* defect on the *AIX* component to authorize the reuse of all files in the *MVS_Release_0* release in the *AIX_Release_1* release.
8. To accomplish this, the AIX developer did the following:
   a. Accepted the *prod_00002* defect
   b. Created a track linking the *AIX_Release_1* release and the *prod_00002* defect.
   c. The AIX builder then linked all files in the *MVS_Release_0* release to the *AIX_Release_1* release, referencing the *prod_00002* defect.

   As result of these actions, project data has been stored and managed by CMVC in the *prod* family. The family data is stored in two places: the database tables and the UNIX file system associated with the *prod* AIX login name. CMVC recorded the following information in the database (the names in parentheses represent the CMVC table names in the database):

   - Members of the development team (*Users*)
   - Network topology information (*Hosts*)
   - Project and data organization information (*Components*, *AccessTable*, and *Notification*)
   - File path names, versions ID, and releases associated with (*Files*, *Versions*, and *Releases*)
   - Problem management and change control information (*Defects*, *Tracks*, *Levels*, *LevelMembers*, *Fix*, and *Verify*).
   - Change process history (*History*)
   - Electronic notes exchanged during problem resolution (*Notes*).

   Figure 24 on page 40 shows the CMVC - Level Change History window for the level *0* of the release *MVS_Release_0*.

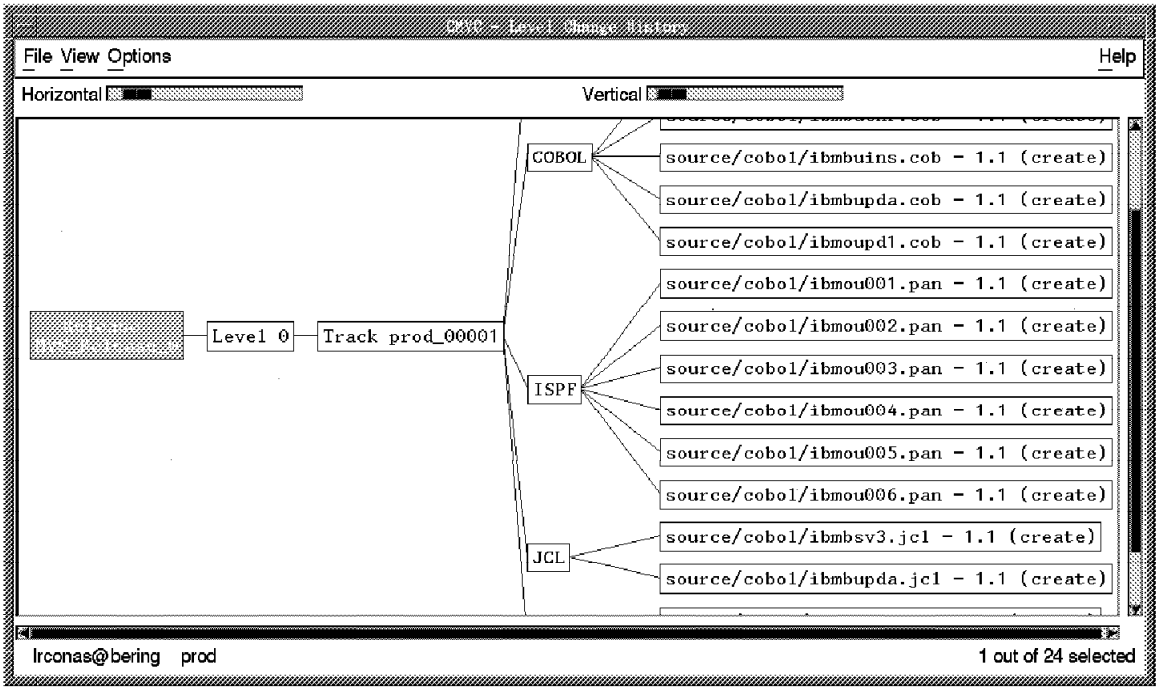

Figure 24. Level Change History

The application source files, managed by SCCS, are stored on the file system associated with the AIX family login name, *prod*. The *prod* family audit log file (/production/audit/log) traces all CMVC transactions that were performed.

Chapter 3. Overview of the Application Development Project

In this chapter we describe the legacy application and the AIX software development environment in more detail than that described in 2.1, “Project Short Description” on page 11. The application design and user interfaces are documented, as well as the network topology, configuration machines, versions of operating systems, versions of LPPs, and the file system organization implemented in the development environment.

3.1 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 describe the following characteristics of this application:

- Programs
- Database design
- User interface.

3.2 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 produced.

3.3 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 Panel 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 Command List (CLIST) 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 Job Control Language (JCL) file. This form reflects either:

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 has been created by the same scanning routine. Programs that execute a batch enrollment of new customers. They are started from the same JCL as IBMBUPD.

IBMBSEL A program that produces a report according to specific criteria. It 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.

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 user interface for the ONLINE UPDATE program utilizes ISPF panels, which were designed for IBM 3720 color terminals. They use various color schemes to display input and output fields in different colors. Users initiate actions by pressing function keys.

3.4 Description of Our Application Development Environment

In this section we describe the specific hardware, software, and file system topologies used for this project. We also describe why we distributed our users and the various compute services across our computers.

3.4.1 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 X servers or development environment framework products.
- The workstation consoles and network attached graphical displays, are dedicated to specific users, but users may login at 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.”

3.4.2 Hardware and Network Topology

Our network topology, as illustrated in Figure 25, consisted of four IBM RISC System/6000 computers, and a RISC System/6000 X station connected on a Token Ring LAN. Table 2 on page 44 identifies the host names, Internet 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*.

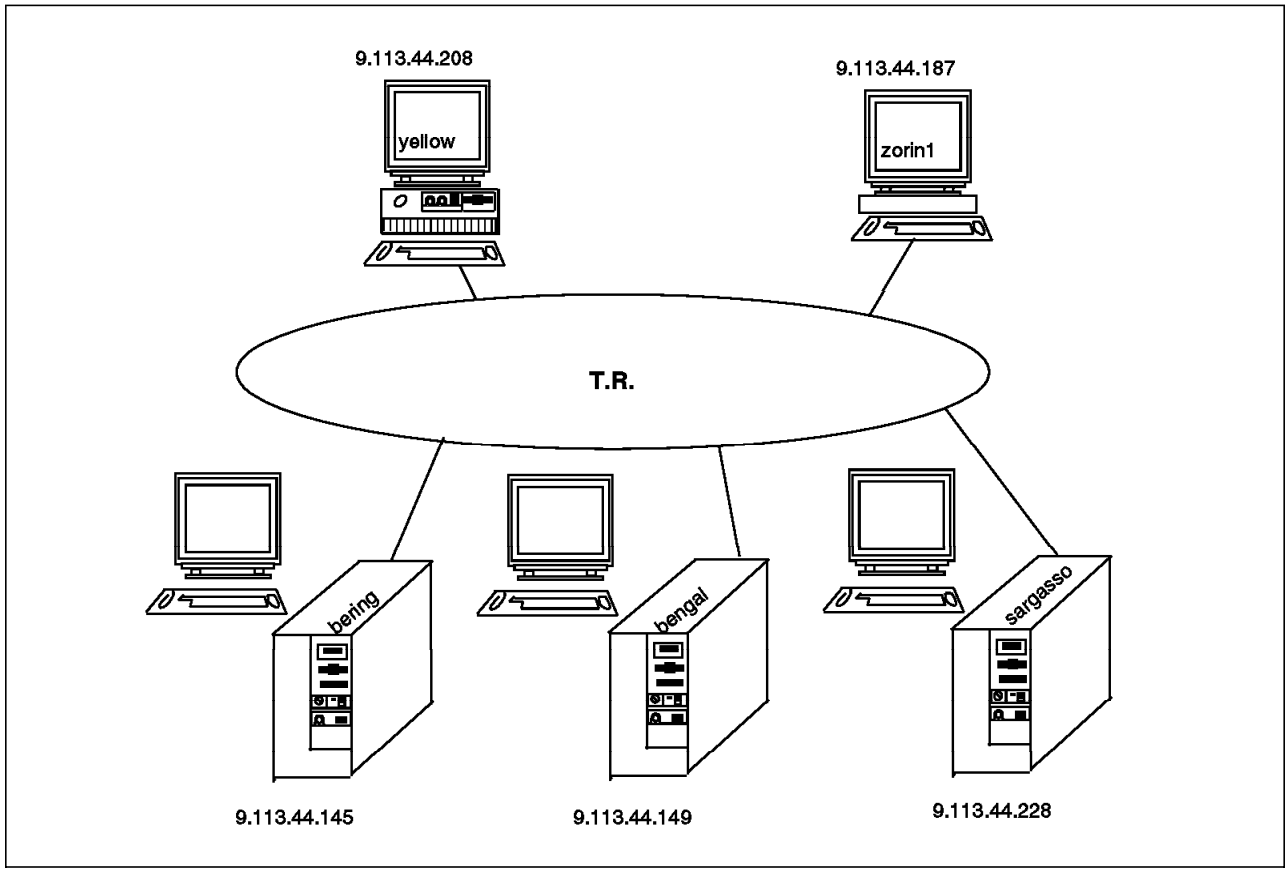

Figure 25. Our Network Topology

Table 2. Our Hosts and Hardware Configurations		
Host Name	**Make / Model**	**Hardware Configuration**
bering	7013 / 52H	2 GB disk, 32 MB memory, 6091-19 display, 8 mm tape, 3.25 inch floppy
bengal	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
sargasso	7013 / 520	400 MB disk, 32 MB Memory, 6091-19 Display, 3.25 inch Diskette Drive, 4/16 MB Token Ring, Colorgraphics Display Adapter
yellow	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	7010 / 130 (X station)	6091-19 display, 4/16 MB Token Ring

3.4.2.1 Displays 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 also have used 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.

3.4.2.2 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* 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 paging 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 hosts. This included the host that supported multiple SDE WorkBench/6000 users, the CMVC server, and multiple CMVC clients.

The amount of disk space required to serve as paging space for a Licensed Program Product (LPP) is usually much larger than the disk space requirement for installing the product. Recommendations for page space are additive, and increasing paging 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 paging space for the first user and 13MB more paging space for each additional SDE WorkBench/6000 user on the host. 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 paging space. Typically, we allocated 100MB of paging space on these hosts.

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 host 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 host, stored all the versioning data for the development source code and related files in AIX journaled file system 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.

3.4.3 Software Topology

Our software topology illustrated in Figure 26 on page 46, shows on which hosts the various client and server portions of the tools and SDE WorkBench/6000 were executed. 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 execute on the host at which the end user executes the X server as much as possible, to minimize display update delay and network traffic.
- Servers should execute on the host with appropriate memory and disk resources.
- Software should execute where prerequisite and corequisite software is available.

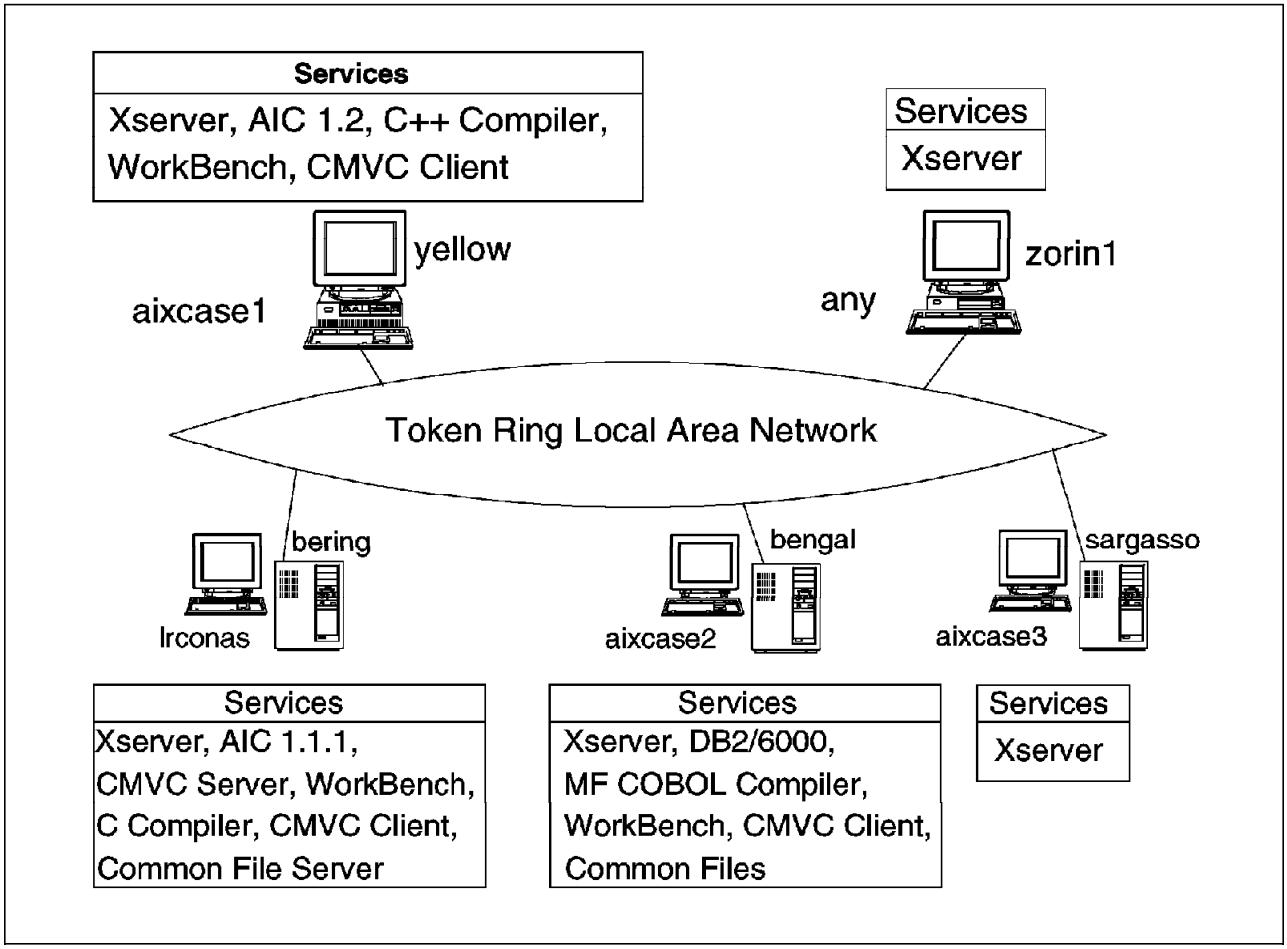

Figure 26. Distribution of Software Services across the Network

3.4.3.1 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 host 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 hosts 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 compatability 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, executed the X server, and remotely executed SDE WorkBench/6000, CMVC, and the other AD tools on other hosts.

Some of the standard LPPs installed on each host 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: C shell, mail handler, etc.

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

Table 3 shows the software configurations installed on the various hosts.

Table 3. 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, XL C 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,
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 1.1, DB2/6000 1.1
yellow	AIX 3.2.4, BOSext1, BOSext2, NetLS 2.1, ADT, TCP/IP, NFS, NCS, Xstation Mgr 1.3, XL C++ 1.1.	X11R5, OSF/Motif 1.2	SDE WorkBench/6000 1.2.2, CMVC 2.1 Client, AIC 1.2, DB2 CAE/6000 1.1, DB2/6000 1.1
sargasso	AIX 3.2.3, BOSext1, BOSext2, TCP/IP, NFS, NCS	X11R4, OSF/Motif 1.1	none

3.4.3.2 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 execute on one computer while the X server it accesses executes on another computer, but both X server and X client can execute 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 execute 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 executing on one host, making input/output requests through an X server that executes on a second host while accessing the CMVC server executing on a third host. The X server always executes on the host where the display is physically attached. An X station is a host that is capable of executing 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 execute at least the X server. They could then remotely execute the tools they needed. Each developer also had occasion to remotely execute some application development products. In subsequent chapters we refer to specific users by their UNIX login names and to specific hosts that are executing applications on their behalf, to illustrate the users′ interaction with their tools and SDE WorkBench/6000. Table 4 identifies the developers′ login names, areas of responsibility, and hosts where their displays were attached.

Table 4. Developer Workstation Assignments

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

3.4.3.3 Network Software

DNS was also configured, but none of our hosts 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.

3.4.3.4 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*.

3.4.3.5 DATABASE 2/6000 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.

3.4.3.6 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 hosts, but most C source code compilation was executed 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 only be installed on HP computers and the compiler capable of generating executable code for Sun platforms would only be installed 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.

3.4.3.7 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 would be upgraded gradually according to business needs.

3.4.3.8 SDE WorkBench/6000

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

3.4.4 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 was established to hold formal releases of the application and provide a working area for application baselines under development. Several file systems were created specifically for CMVC and ORACLE. File systems were mounted across the network to provided 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 will be 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.

3.4.4.1 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 SDE WorkBench/6000, which was executing on remote hosts, access to 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 executed SDE WorkBench/6000.

Refer to *Installing IBM AIX SDE WorkBench/6000 and IBM AIX SDE Integrator/6000* for instructions on supporting distributed data. Figure 27 on page 51 illustrates the NFS mounts used to support distributed data with SDE WorkBench/6000 for this project.

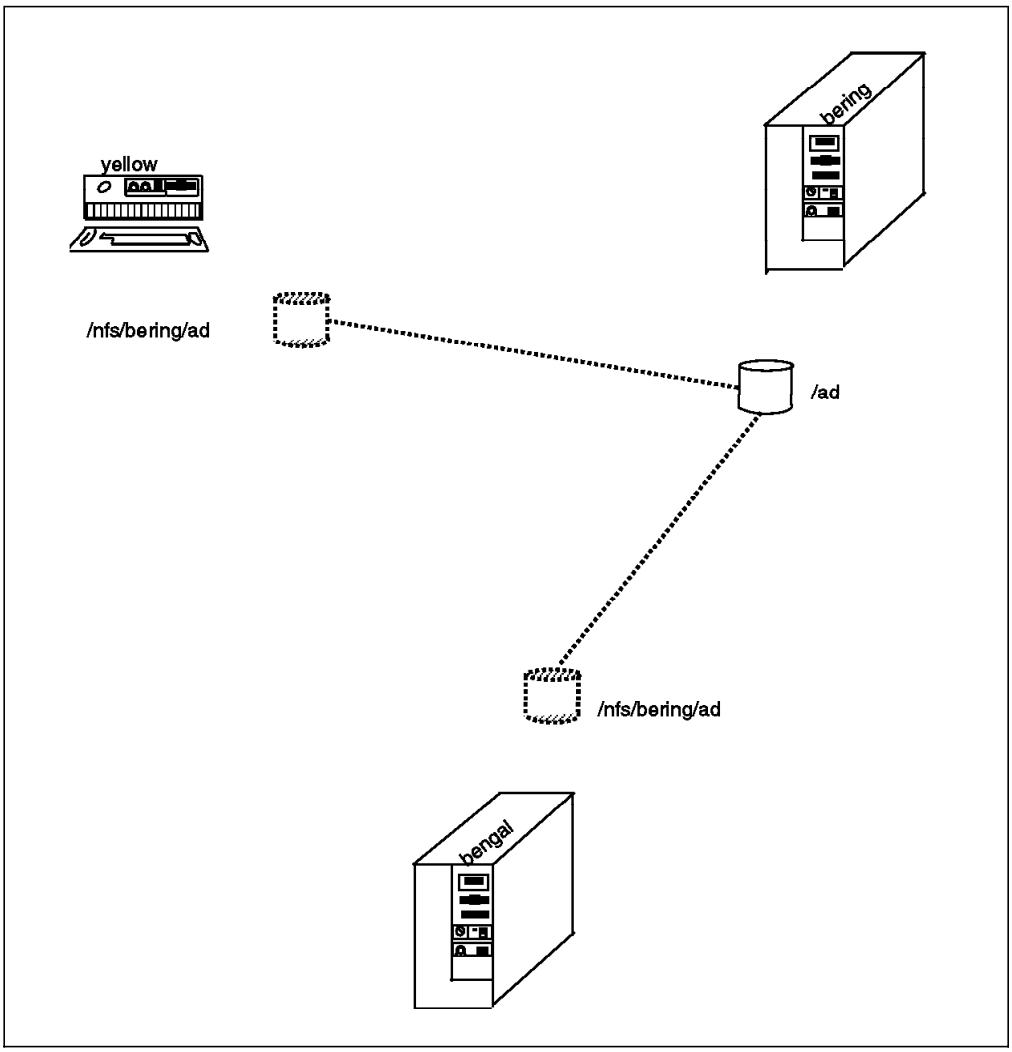

Figure 27. NFS Mounts to Support Distributed Data with WorkBench

3.4.4.2 NFS Mounts for Distributed Execution with WorkBench

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 execute 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 28 on page 52 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.

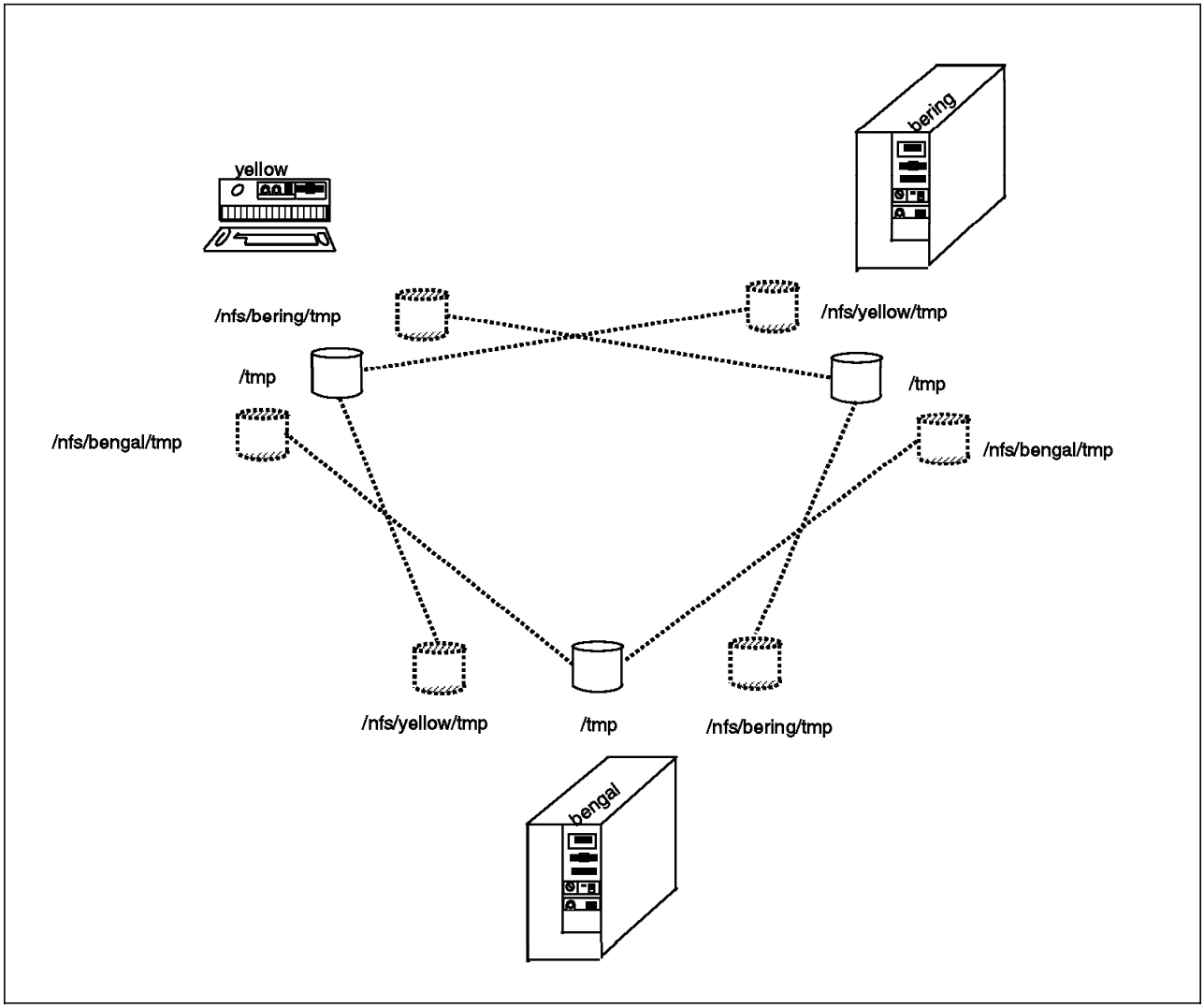

Figure 28. NFS Mounts to Support Distributed Execution with SDE WorkBench/600

3.4.4.3 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 only be one. Also, because many tools and utilities require users to tailor specific files located in their home directory, it can be a lot of work maintaining identical copies of those files on multiple hosts.

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 a each user normally logged in, the default home directory was set to /home/username, and that is where the home directory would really be 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 a different 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 29 on page 54.

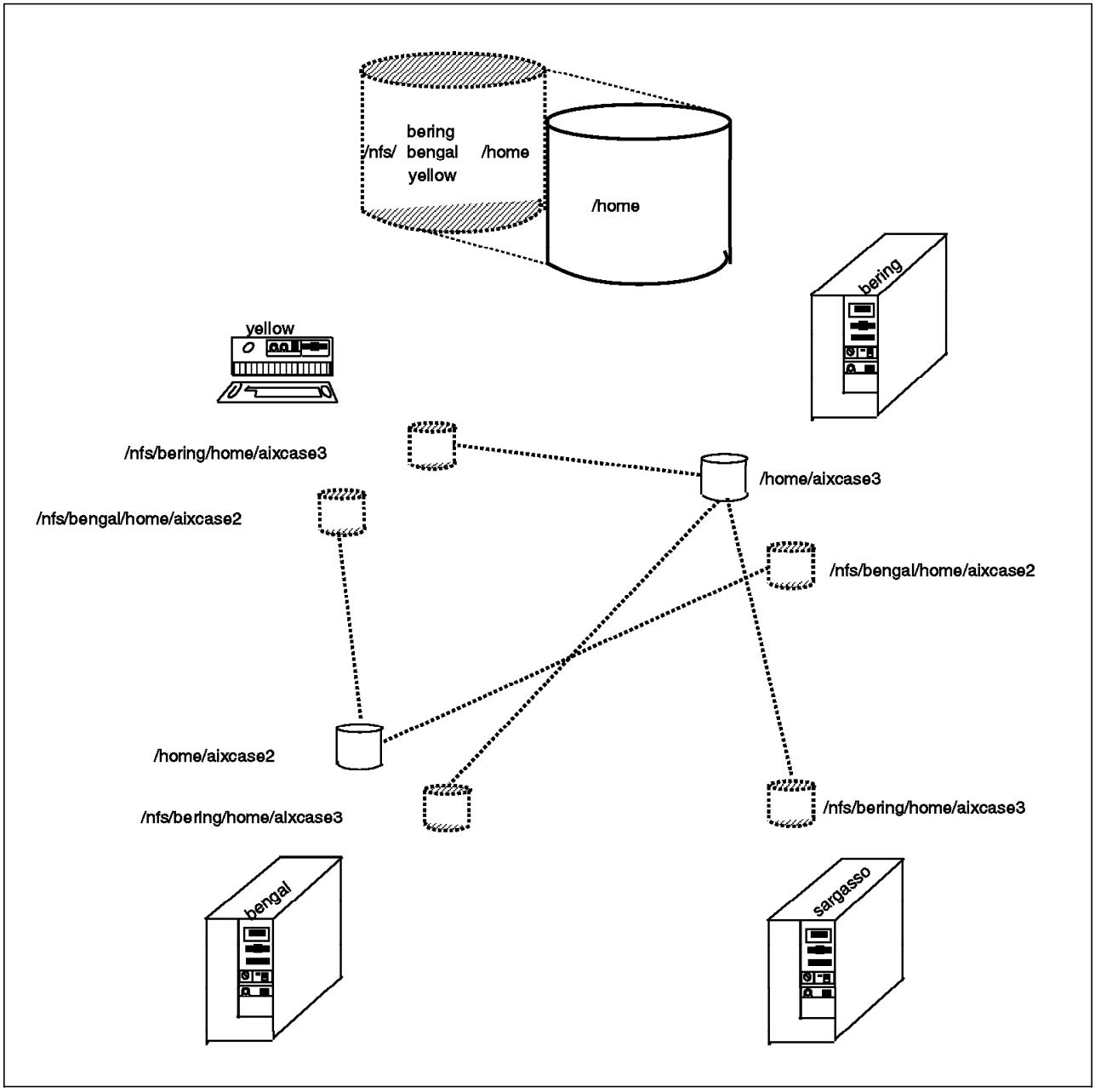

Figure 29. NFS Mounts to Support a Single System Image

3.4.4.4 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 scoped*, so it was unable to interpret remote file path names or path names that began with the SDE WorkBench/6000 convention, which is: `/nfs/hostname`.

The files of concern were all in the common 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 **link** command.

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 the 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 30 illustrates how the local file links masked the NFS mount implied by the path names beginning with /nfs/hostname.

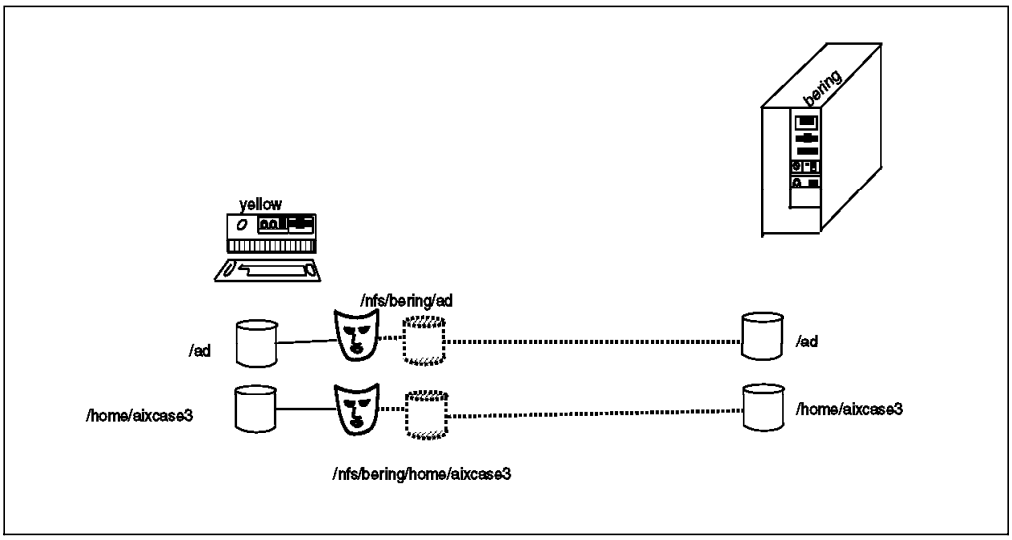

Figure 30. File Systems Cross-Mounted on Our Hosts

3.4.4.5 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 in 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 35 on page 63 and discussed further in 3.4.4.7, "The ProductA Production Release Directories" on page 61.

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.

3.4.4.6 ProjectA Prototype Development File Tree

The *projectA_proto* prototype development file tree is shown in Figure 31 on page 57. 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	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 executes, the X Windows System identifies 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.

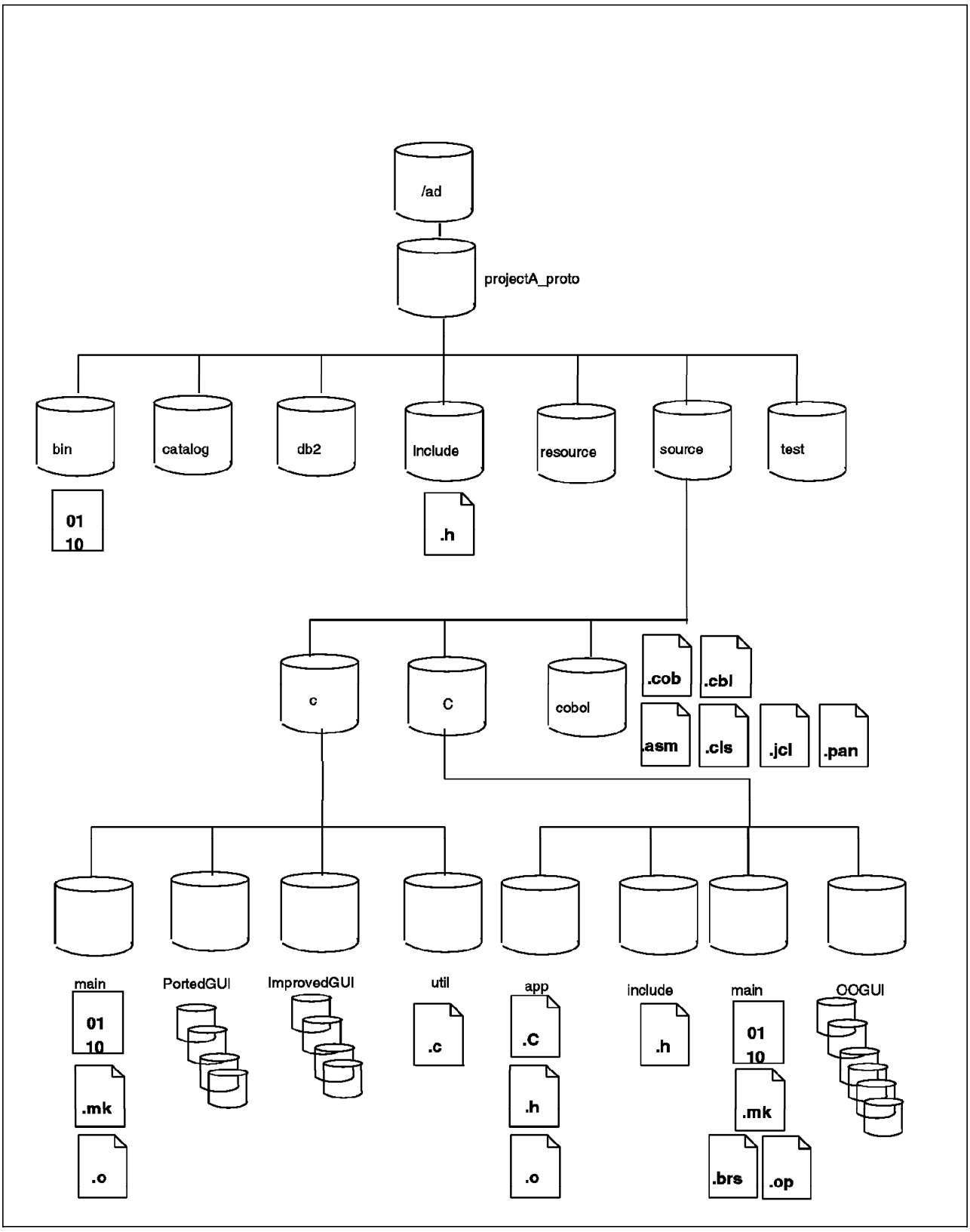

Figure 31. ProjectA Prototype Development File Tree

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 32 on page 59 illustrates the *PortedGUI* directory and Figure 33 on page 60 illustrates the *ImprovedGUI* directory. The C language source directory contained the following directories:

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.

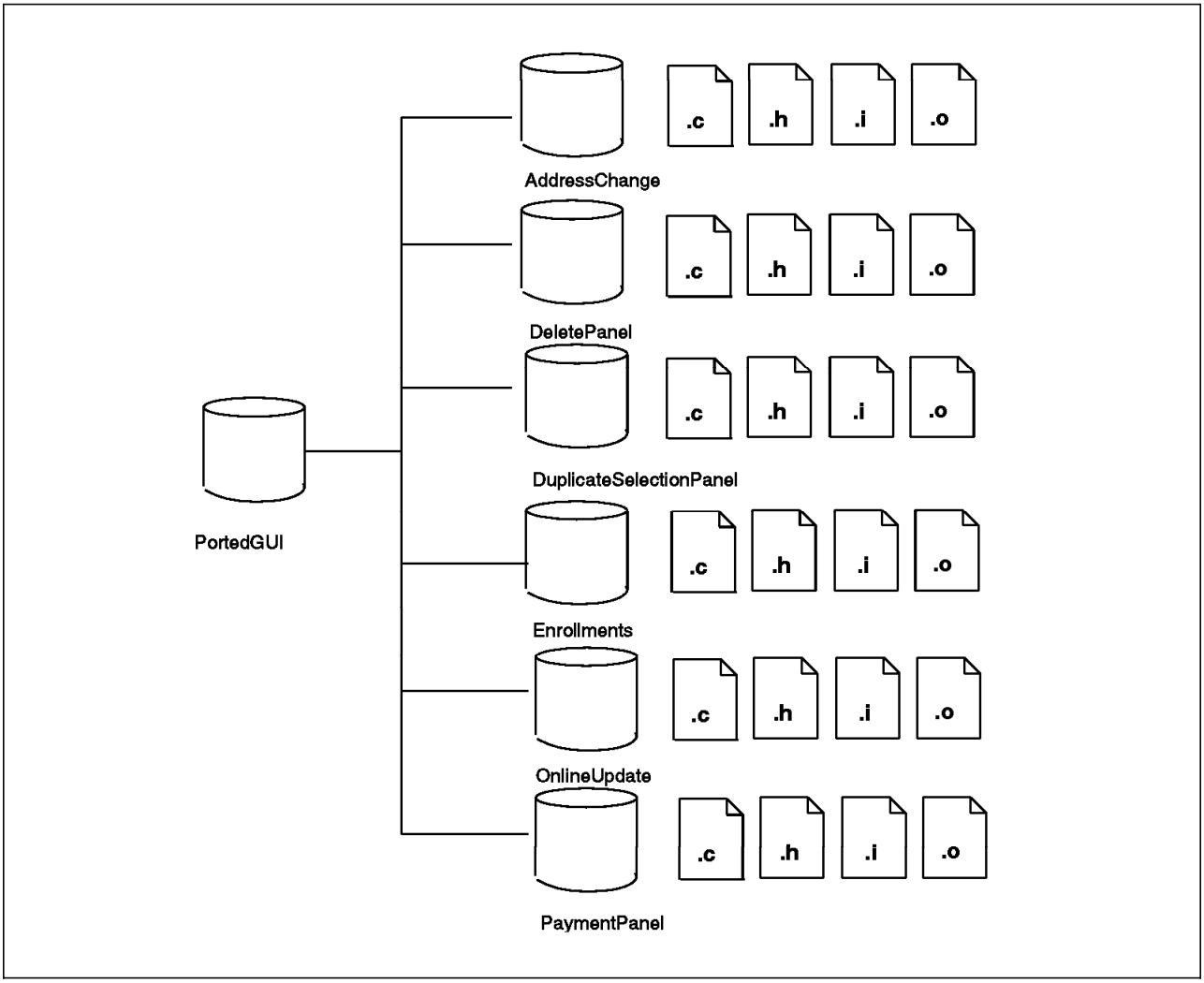

Figure 32. PortedGUI Directory

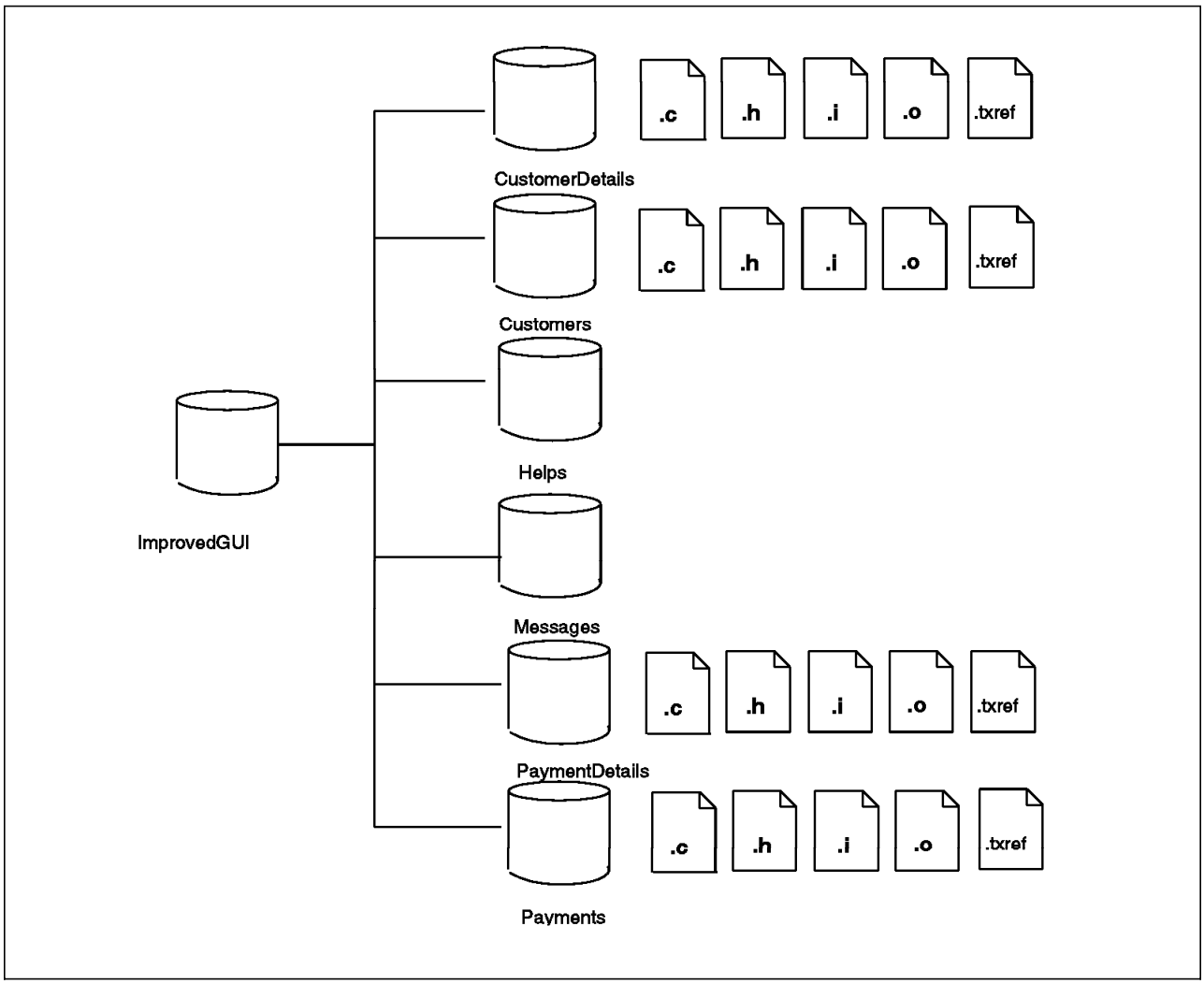

Figure 33. ImprovedGUI Directory

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 34 on page 61 illustrates this directory. The C++ source language directory contained these following directories:

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, which were necessary for access to the DB2 CLI for C.

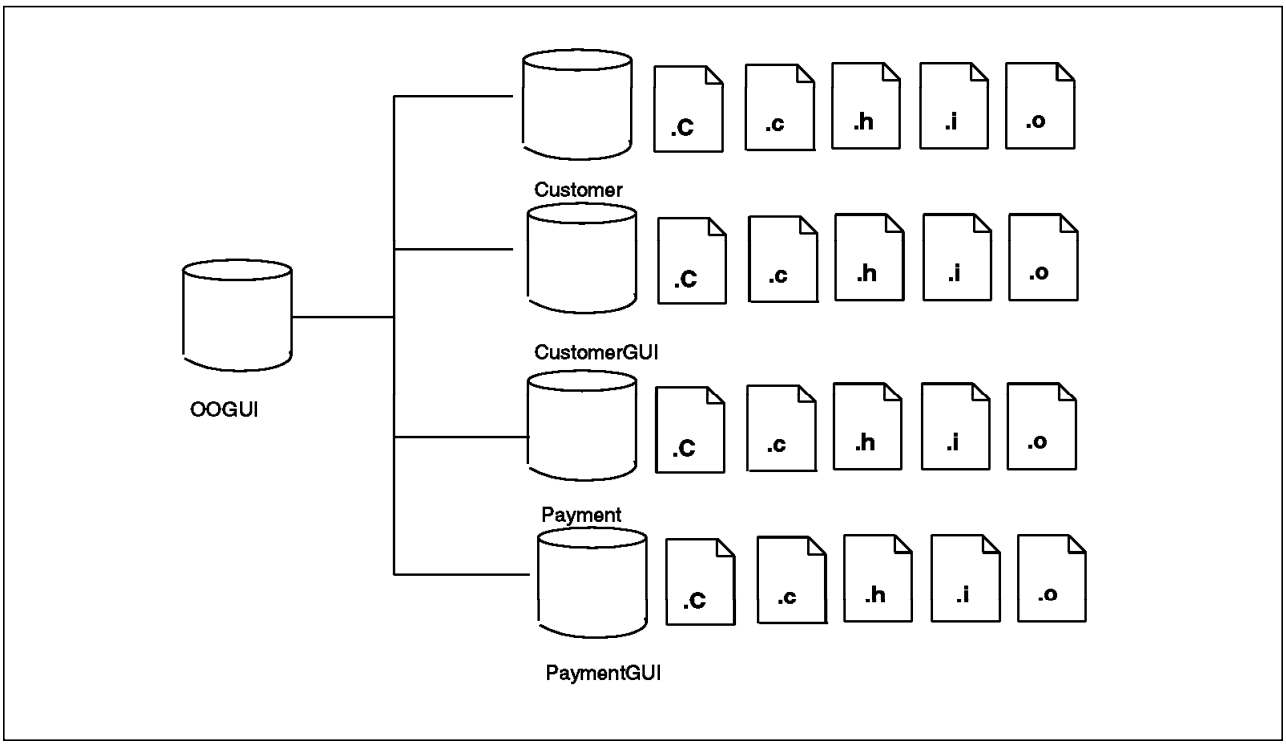

Figure 34. OOGUI Directory

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, Job Control Language (JCL) scripts, Command List (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.

3.4.4.7 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 35 on page 63. It contained the following directories:

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 didn't 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 didn't 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.

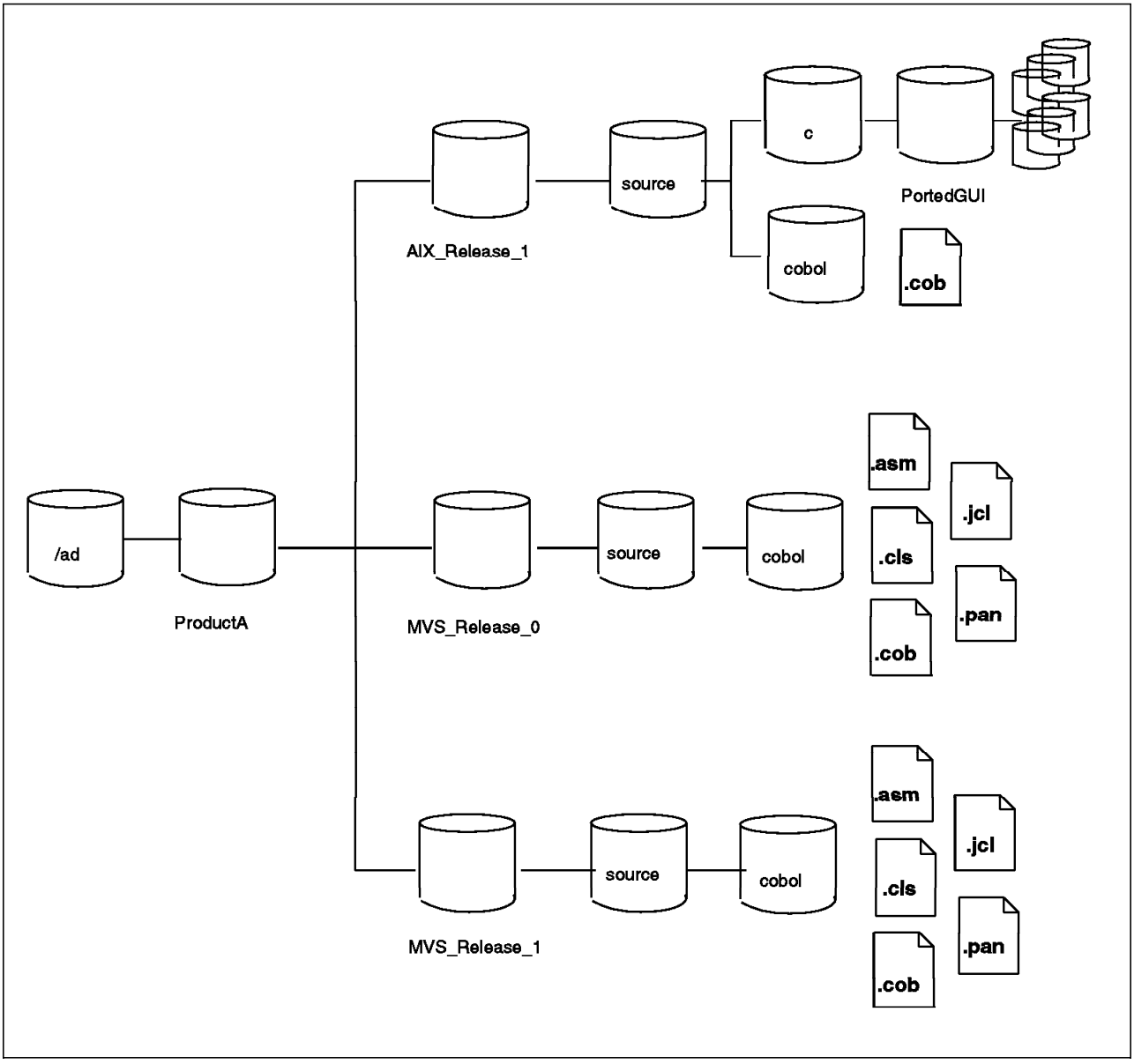

Figure 35. ProductA Production Release File Trees

Chapter 4. Planning for CMVC

In this chapter we offer some general advice on planning to use CMVC, as well as offer some examples of how to apply specific features or functions in CMVC. We also show how a small application development project, described in Chapter 3, "Overview of the Application Development Project" on page 41 and alluded to in Chapter 2, "Discovering CMVC: An New Application Project Is Introduced to CMVC" on page 11, planned for its use of CMVC.

4.1 Why Plan?

The most important thing to understand about CMVC is that you do not need to understand all of it, before you begin using some of it. CMVC is broad in its function, thorough in its implementation, and very flexible. CMVC provides many mechanisms to help you accomplish your SCM goals, but it does not dictate exactly how you should use them, nor does it require that you use them all if you use only some of them. This is one of the distinguishing advantages of CMVC. Because no two development efforts have exactly the same number of requirements, the same degree of complexity, scope of effort, or the same hardware and software resources, how they approach SCM with CMVC varies significantly. Recognizing this, CMVC was designed to be set up, tailored, and utilized according to the needs of the individual project. Therefore, it is wise to plan in advance which features of CMVC you want to use initially, how you want to apply them to your SCM problems, and which features you want to phase in gradually.

The first step in this planning process is to go over the CMVC product documentation carefully. These documents identify and define CMVC objects, such as Files, Releases, and Users, and describe how CMVC users access and manipulate them. In *IBM CMVC Concepts*, you find a description of CMVC concepts, objects, processes, and interactions. Look in *IBM CMVC User's Guide* and *IBM CMVC Commands Reference* for details on how to use individual commands and GUI windows. *IBM CMVC User's Reference* provides a place to look up lists of options, record structures, and field attributes.

However, what you will not find in these documents is the *correct* interpretation of how to map CMVC objects to the real objects of your application development effort. Nor will you find advice on which circumstances call for using or not using a given CMVC object or function. This is because there is no single correct application of CMVC; this will vary with your circumstances.

The second step in planning, therefore, is to compare your thoughts of how to apply CMVC to someone else′s actual experience. Since not everyone can do this, we have written this chapter. It provides a practical example of how to plan for and apply CMVC objects, concepts, and processes to meet the needs of an actual software development project. This chapter also suggests alternative approaches to CMVC that were not used on this project, but are based on other experiences with CMVC.

SCM is not a short term effort, nor does it exist in isolation. SCM responsibilities for an application begin during its development and continue as long as it is in use. The SCM effort on an individual application development project is also part of a larger SCM effort in the development organization. Therefore, original

plans for CMVC are subject to revision as the needs of the project change and as additional projects start up. Your use of CMVC will also evolve as you become more familiar with its capabilities. Generally speaking, CMVC is amenable to this fact of life. But, some decisions about CMVC that you make at the beginning of a project, have a long term impact. This book helps you distinguish between these and other decisions, which you can make tentatively now and the plan to modify as time passes.

CMVC provides a command-line interface client, as well as a GUI client. Rather than refer to specific command and parameter names, and equivalent GUI window and menu items in this chapter, we refer to CMVC actions by a generic name. For example, we refer to the *FileCreate* CMVC action, when we mean either the **File -create** command, or the **Create** selection on the Actions pull-down of the CMVC - Files window. You should refer to the manuals to identify the actual correct spelling of the command and parameter, or window and menu item names.

4.2 Pre-Installation Planning for CMVC

Before you install CMVC, you should:

- Plan your network license requirements and distribution of licenses over your hosts
- Plan your distribution of CMVC client and server software across your hosts
- Identify and define the purposes served by your CMVC families.

4.2.1 Planning Network License Requirements

CMVC makes use of Network Licensing System** (NetLS**). For details on how NetLS works and how CMVC makes use of the NetLS licensing mechanism refer to 6.2, “NetLS Installation and Initialization” on page 163. There are some decisions you must make regarding NetLS; they are primarily questions of network and system administration. These include whether to have more than one NetLS license server, and how to distribute license tokens for various licensed programs among them. The primary decision you must make regarding CMVC and NetLS, however, is how many CMVC license tokens your project will require.

4.2.1.1 What to Consider in Planning Network License Requirements

To plan your CMVC license token requirements, try to determine the maximum number of users who will be using CMVC at any one time. Someone performing SCM functions may need access all day long, while your developers may use CMVC infrequently. Not all developers will use CMVC to the same degree; it may depend upon their specialized role in your project. The project and team leaders may use CMVC a few times daily, while testers and build integrators may use it constantly.

Before a CMVC client issues a request to the server, it requests a license, or a token. After getting it, that CMVC client holds it a minimum of fifteen minutes. (This is CMVC′s default minimum expiration time). If, in the next fifteen minutes, that client issues another request to the CMVC server, that token′s expiration time is extended again by fifteen minutes. So, if you bring the CMVC client GUI up, make and refresh queries, display new windows, and perform CMVC actions every few minutes all day long, you will effectively use one token for most of that

day. On the other hand, if you start up the CMVC client GUI, make a couple of queries, and then leave the CMVC client GUI running, but make no more requests, you relinquish that token fifteen minutes after your last CMVC action. You can increase the expiration time, but you cannot reduce it.

4.2.1.2 Our Network License Requirements

On our project, we had a one-to-one ratio of team members and NetLS licenses for CMVC, because our team members used CMVC frequently. In part, this was because getting acquainted with CMVC required some time and interaction, apart from the application development requirements. However, we found that developers in general made frequent use of CMVC. We ran one NetLS license server on the same machine as our CMVC server, and installed all our CMVC licenses on that server.

4.2.2 Planning CMVC Client and Server Hosts

You must decide on which hosts you want to install CMVC. It is likely that you will have one server and several clients spread about your network. You could, of course, run the clients and server on the same host.

4.2.2.1 What to Consider in Distributing CMVC Client and Server Software

It is a good idea to have the CMVC server execute on a host with plenty of available disk space and memory, and have the clients execute on hosts at which end users login. The CMVC server and the relational database management software on which it depends, use a large amount of disk space for the data they control. They also require a lot of memory, so be sure to allocate sufficient paging space on the server, in addition to the disk space requirements for the library data. *IBM CMVC Server Administration and Installation* states that the CMVC server alone requires 16 Megabytes (MB) of internal memory, and 10 MB of disk. This does not include the disk or memory required for the relational database management system.

Like any X Windows application, the CMVC client GUI can execute on one host and display its output at the X server executing on some other host in your network. If all your users accessed the CMVC client this way, it could place a burden on your network and result in slower response time. In our project, we displayed the CMVC client GUI both at direct-attached consoles on the hosts where the CMVC client GUI executed, and at remote X servers and X stations.

It may be possible to implement multiple CMVC server hosts, which contain parallel data or operate in some coordinated fashion to simulate that condition. CMVC, however, provides no direct support for either coordinating the state of multiple families, or multiple CMVC server databases. Such an arrangement would involve some programming, use of the cron tables, and probably a significant number of user exits, if it is possible. There was no requirement in our project for such an arrangement, so we did not investigate this area.

4.2.2.2 Our Distribution of CMVC Clients and Server

For our project, we chose to place the server on *bering*, one of our RISC System/6000 Model 52Hs, because it had 2 GB of disk, 32 MB of memory, and AIX 3.2.3e. We placed our CMVC clients on all the other hosts, except *sargasso*, which was a Model 520 with minimum hard disk. CMVC Version 2, Release 1, worked well under both AIX 3.2.3e and under AIX 3.2.4, which became available at the beginning of the project and was installed on *yellow*. One user, *aixcase3*,

always executed the CMVC client GUI from *bering* while displaying it at the X server on *sargasso*.

4.2.3 CMVC Families

A family represents a complete collection of CMVC users, components, data, and files that is self-contained and makes no reference to any other components, files or data outside of itself. Data in a family is completely isolated from data in all other families.

4.2.3.1 What to Consider in Choosing CMVC Families

You must determine how many families you will need to create before your team can use CMVC. Generally speaking, an application development organization will not have many families. There will usually be more reasons to include a new development effort in an existing family, than to create a new family for it. If your development effort is entirely separate, however, making separate family for it is appropriate.

For example, your development organization may have several different applications under development and maintenance, and you may want to make a family for each one. If these different applications shared some source code, like a utility subroutine library, then each family would have to maintain its own copy of the source code for this library. But, if these applications were all kept in one family, they could share, or reuse, this common software, while keeping only one copy of the source code. Likewise, if the applications shared a common set of data files, such as NLS error message catalogs, then being in the same family would make the most sense.

New families can be created over time as your needs evolve. Elements of family's data associated with a release can be archived, and removed from the family. If this data is required later, it can be restored to a separate new family. In addition, it is possible to migrate data from one family to another, if you want to create a new family using data already controlled by CMVC in another family.

4.2.3.2 Our CMVC Families

For our project, we concluded that we wanted two families, one to support code in production, and one to support code under prototype development. The two families were named *dev* and *prod*. Our reason for this choice was that we did not expect these two groups to share any files. Files might migrate in one direction from the development realm into the production realm, but development might continue on its own path after that and would not be using production code. We also wanted a CMVC domain in which we could test CMVC itself, and not threaten the integrity of the CMVC data supporting our real project.

4.3 CMVC User IDs and Host Lists

CMVC users have a unique CMVC user ID, which is independent of the UNIX login name.[2] Since users frequently log in to various hosts from other hosts in a distributed environment, CMVC allows the association of a single CMVC user

[2] The term "UNIX login name" refers generically to a login name on any of the UNIX-based operating systems from HP (HP-UX), Sun (SunOS or Solaris), or IBM (AIX), for which CMVC client software is available. Because our project used only AIX hosts, we often use the specific term "AIX login name," instead of the generic term. However, there are no significant

with multiple login names at multiple hosts in a Host List. There is one list per family. For example on our project, the CMVC user ID, *truls*, could access the *prod* family while logged in as the *aixcase2* AIX login name at the *bengal* host, and yet be unable to access *dev* family data under the same circumstances. CMVC permits a given UNIX login name and host name combination to be associated with multiple CMVC IDs, and vice a versa. CMVC also allows you to record an area of interest or responsibility associated with each user.

CMVC employs the concept of a privileged user named the “family superuser.” The family super user can perform any CMVC action, and there are some CMVC actions, which only the family superuser can authorize other users to perform. The family superuser is established by default when the family is created, but additional CMVC IDs can be given family superuser status.

4.3.1 What to Consider in Planning CMVC Users and Host Lists

CMVC is the one application development tool that most people in a development organization will use, so generally speaking, everyone on the team will need a CMVC user ID. You need to determine from which UNIX login names, on which hosts, each team member will access CMVC. You need to identify a single UNIX login name and host at which each team member wants to receive electronic mail.

You also need to determine which data access and CMVC action authority each team member requires. An important factor to consider is which users should have primary responsibility for the project as a whole, for architectural elements of the product, for software quality assurance, for software testing, for documentation, end user evaluation, and of course, software design and development. Identify developers with special duties that cross these areas of responsibility, such as release managers and build integrators. Identify people who should only be able to report problems (defects) and suggest improvements (features). Determine who will perform the roles of SCM administrator, CMVC family administrator, and software librarian. Do not forget managers; they will want the kind of CMVC access that lets them monitor the pulse of the project, but does not burden them with extraneous mail.

Since CMVC user IDs do not need to be identical with UNIX login names, decide what naming scheme you want to use for them. If your development organization is large, with clearly differentiated responsibilities, you may want to use impersonal CMVC IDs, which denote a role in the project. This can save time reassigning ownership of CMVC objects to new users when old users leave. It is easier to modify the host list for the entire family, than to modify, for example, the owner of each component, defect, or release. On the other hand, it may be more difficult to infer the personal identity of the users associated with the CMVC user IDs, if you choose this option. Choosing CMVC user IDs to correspond exactly with the login names works well if all your users have the same login names on most of their hosts. But it can be confusing if they each have multiple names.

Create a list of one word labels that identify your users in terms of the roles they play in your development effort. These labels can have nonspace separator characters, but should not be very long, for example, “user_interface,”

---

differences in function, feature, or client-server interactions among the various CMVC client and server products available on these UNIX-based platforms.

"software_QA," or "product_management." These will be the area of interest or responsibility that CMVC associates with each user. Create a table with one row for each proposed CMVC user ID. Write down this label, the associated host name and UNIX login name, the role you expect the user to play, and the user's electronic mail address. Table 5 shows part of the table for our project.

Table 5. CMVC ID and Host List Plan

CMVC user ID	Role	Login@Host	Mail
projA_lead	*Project A Lead Engineer*	*aixcase4@bering*, *aixcase4@bengal*, *aixcase4@yellow*	*aixcase4@bering*
krt	*AIX Development*	*krt@bering*, *krt@yellow*	*krt@bering*

4.3.2 Our CMVC User IDs and Host Lists

We had a small number of individuals working on this project, but each played several roles. We had three developers, each with a language speciality and a clearly defined role in terms of programming. These members were not only the programming staff, but also the designers, builders, and testers for their portions of the application. We had one member who performed software configuration management, project management, and software quality assurance.

These users normally logged in from a specific AIX host where they would check their electronic mail, but they often logged in from other AIX hosts. We chose to implement multiple CMVC IDs for these users, which reflected the different role they played in the development effort. We also ensured that at least two UNIX login names mapped to each CMVC superuser ID.

The family superuser ID was *Irconas*, which mapped to the login name by the same name. Our developer's UNIX login names were *aixcase1*, *aixcase2*, and *aixcase3*. Our CMVC IDs for these users, when performing the role of developers were *branko*, *truls*, and *krt*, respectively. When these developers performed other roles, they assumed one of these CMVC user IDs: *OOtester*, *MVStester*, and *AIXtester*. The *manager*, *projA_lead*, and *projB_lead* CMVC user IDs mapped to the *aixcase4* UNIX login name.

Figure 36 on page 71 shows the results of a query of all CMVC IDs for the *prod* family as displayed in the CMVC - Users window. This query identifies the CMVC user ID, person, role, and electronic mail address.

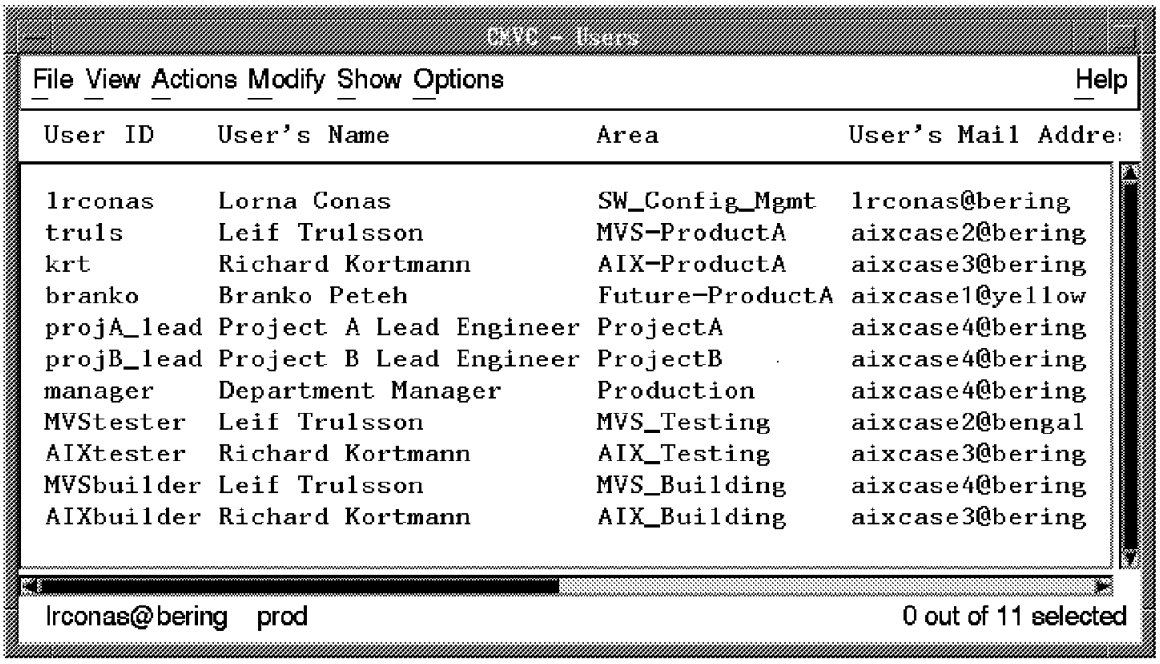

Figure 36. User List for Production Family

Figure 37 maps login names and host names with CMVC user IDs, by showing the results of a query for all IDs in the CMVC - Host Lists window.

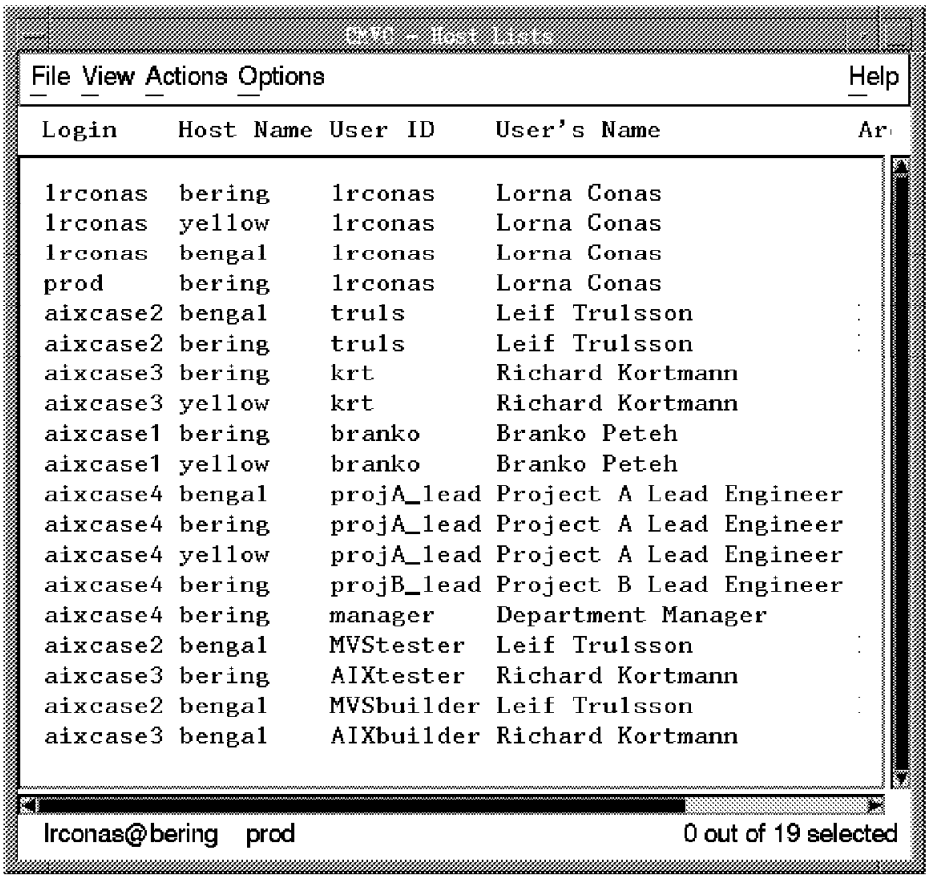

Figure 37. Hosts List for Production Family

Since the CMVC Users record can have configurable fields, potential configurable fields should be examined at this time. We considered several likely uses for additional fields, but decided that most of the information we thought about storing in this record would be redundant with information found elsewhere in non-CMVC departmental databases. For example, we could add fields that would contain a CMVC user's phone number, department, or development team affiliation. This would seem, at first glance, to be useful information, but because the CMVC database is not the authoritative source of this data, it would become out of date quickly. Keeping it up-to-date would be a time consuming.

If information about a CMVC user is not easily obtained from other databases, then it is a good candidate for a configurable field. For example, consider a big project involving developers from several subcontractor companies, where the CMVC database is maintained by the prime contractor. The prime contractor would not keep personnel records for the various subcontractors, and would have no direct way of knowing to which company an individual developer who filed or fixed a defect belonged. In this case, it would be helpful to add a configurable field to the CMVC User record to hold the name of the subcontracting company for which the CMVC user works. As contracting personnel change over time, it may in fact be more important to know the company affiliation, than an individual's name.

We do not recommend creating new fields in the CMVC User record unless the CMVC User record is the only repository of that data. We did not configure any new fields in this record.

4.4 CMVC Family Component Hierarchy

The CMVC "component" is the basic management unit of organization in the CMVC family. A component is used to organize data, to control access to it, and record information about it. It also supports the automatic notification of users of changes in the state of that data.

The components themselves are organized in a hierarchy. The top node is called the *root* component, but its actual component name, and those of its descendant components is yours to define. Each family contains one component hierarchy. The structure can become quite complex as, the layering of children components and, their children components becomes deeper and broader. Any component whose parent is not the root component, can have multiple parent components, as well.

Conceptually, a component is somewhat analogous to a UNIX directory hierarchy. A component not only serves as a means of grouping and organizing data, which is stored in files, but nonfile data that validates and directs CMVC actions is also associated with the component. This meta-data is stored in the tables controlled through the relational database product employed by CMVC. It also defines the component hierarchy. CMVC records a history of all modifications to this nonfile data, as well as a history of all changes to those files grouped by the component. Automatic notification of CMVC actions is defined per component, and these characteristics are inherited by descendent components. Access (the right to perform CMVC actions associated with authority groups) is defined per component, and is inherited by descendent components. However, it can be specifically excluded from inheritance at a given descendent component, as well.

The files managed by a component can be binary or ASCII data, no matter what their content represents or means. The obvious data files are source code files and makefiles, but there are many other appropriate candidates. These include application documentation (source or binary files), engineering drawings in source or intermediate formats, and design and specification tool input or output files. Files not related to the software itself, but to the organization developing the software, are equally appropriate.

4.4.1 What to Consider in Planning a Component Hierarchy

Study the CMVC concepts associated with components. After gaining an understanding about what components do in the context of CMVC, you need to plan a component hierarchy to support your configuration management needs. The following paragraphs outline some factors to consider when planning your component hierarchy.

IBM CMVC Concepts indicates that the component hierarchy can be built to reflect different organizing principles, or a mix of principles. One way to organize a component hierarchy is to have it mirror the application development organizational hierarchy, such as department, section, team, or unit of development. Another hierarchy approach is to reflect the software architecture of the applications under development, such as application, GUI subsystem, communications subsystem, database, or subsystem.

Components can be added and additional parent-child relationships established to overlay notification relationships. There are many possible component hierarchies, which could be implemented for any given situation, but if the best is not immediately discerned, it can be modified later. Components can be reparented, deleted, and renamed, and all their properties can be modified after creation, as well.

One factor to consider in arranging your component hierarchy is that all defects and features must be recorded by component, and the owner of a component becomes the default owner of its defects and features. While CMVC allows you to reassign a defect to another component, and even to change the owner of the defect or feature to another noncomponent owner, the more often you must do this, the more complexity you introduce to the user′s interaction with CMVC.

If you organize your top components according to the major components of your architecture, you might want to continue defining lower level components until you have a component for each assigned unit of work. In most cases, team members opening a defect will be able to correctly identify the component whose owner is most likely to have to implement the correction. When in doubt, the team member opens the defect at a higher level, but this does not happen often. The higher level component owner may be the task leader for the team members owning components below, and can reassign the defect appropriately.

While there is no magic ratio of number of files to a component, there are advantages to breaking up the files into several components. For example, if you have many source files managed by only few components (a flat component hierarchy), you lose the ability to perform detailed analysis on the rate and nature of defects based upon more granular subsets of your application. You might want to be able to compare defect history for the user interface code with that for the data management code and the device driver code. This can easily be done by looking at defects on a component basis, if each of these architectural elements are managed by a separate component.

You do not need to have the components reflect identically the directory tree you implement to support your software build, and in some cases it advantageous not to do so. For example, when you extract the files for a build, you want have a separate directory for every window in the your GUI. You can then extract all the AIC interface files, callback source files, and include files, which are relevant to the particular window to that directory. This arrangement simplifies scanning any one directory to verify that all relevant files are in place or have appropriate dates. You do not need to create a component for each window, however, matching this directory structure. You may be better served by creating a single component for all the files associated with the entire user interface, especially if one developer is responsible for the development and maintenance of the entire user interface. You would have a single component managing all defects and features related to the various panels, and a single person controlling access to all the files.

Another factor in forming the hierarchy, is controlling access to data managed by the components. One approach is to institute a pattern of parallel mini-hierarchies. If you have a project, which has program source and documentation related to those programs, you might maintain the documentation files in one mini-hierarchy, and the source files in another. Both of these descend from a common component representing the project. This organization allows one person, namely the owner of the top-most component in the documentation mini-hierarchy, to have access to all the documentation components. Another person, the owner of the top-most component in the parallel mini-hierarchy, has access to all the program components. The relationship between files for a given document and program element can be made clear by the position of the component in the hierarchy, and by naming conventions.

Component naming conventions can be valuable if thought out in advance and adhered to strictly. They can make generating CMVC reports easier, by allowing users to subset all possible report data by selecting the appropriate components based on some common element of their names. This can be especially helpful where several components, which have common purposes, are distributed randomly about the component hierarchy. For example, components for end-user documentation might be descended from various components managing other components for test data, source code, or specifications. For example, if all their component names have “doc” as the first three letters, it will be easy to distinguish them from components managing other data, no matter where they appear in the hierarchy. In addition, naming conventions can influence how reports sort and order data.

We suggest you make a sketch of the hierarchy, and create a table or matrix in which you make notes about each component as you do your planning. Having identified the components in your hierarchy, add to each component′s line a list of the types of files you want to control with the component and if possible the pathnames in your directory build tree which will be associated with these files. Table 6 on page 75 shows a part of the table for our project.

You may not know all of the components you will need to create at first. Keep in mind, that component owners can create children components as they need them, and the hierarchy can grow arbitrarily wide or deep, as the project progresses.

Table 6. Component Hierarchy Plan		
Component	**Description**	**Path names**
legacy	Contains components for the legacy application on each platform (*produchp1.).*	not applicable
MVS	Contains components for MVS source code (*productA*).	not applicable
COBOL	COBOL sources (*.cbl, *.cob, *.cpy)	source/cobol
ISPF	ISPF panel files (*.pnl)	source/cobol

4.4.2 Our Component Hierarchy

We determined that we needed two families. The *prod* family supported software, which was in production use, and the *dev* family supported development of emerging applications and our reimplementation of the legacy application with object-oriented technology. The root components of the two component hierarchies belonging to these families were named *production* and *development*, accordingly.

4.4.2.1 Our Production Component Hierarchy

We chose to organize our component hierarchy at the top levels to reflect how we were organized to perform our work, and at the lower levels to reflect the execution platforms and language technologies employed. We created a hierarchy of components, which would support the work of an application development organization that had two major application products in the field, and others under development. Figure 38 on page 76 illustrates the component hierarchy we created to support our *prod* family. This figure shows the contents of the CMVC - Component Tree window when we selected the horizontal layout.

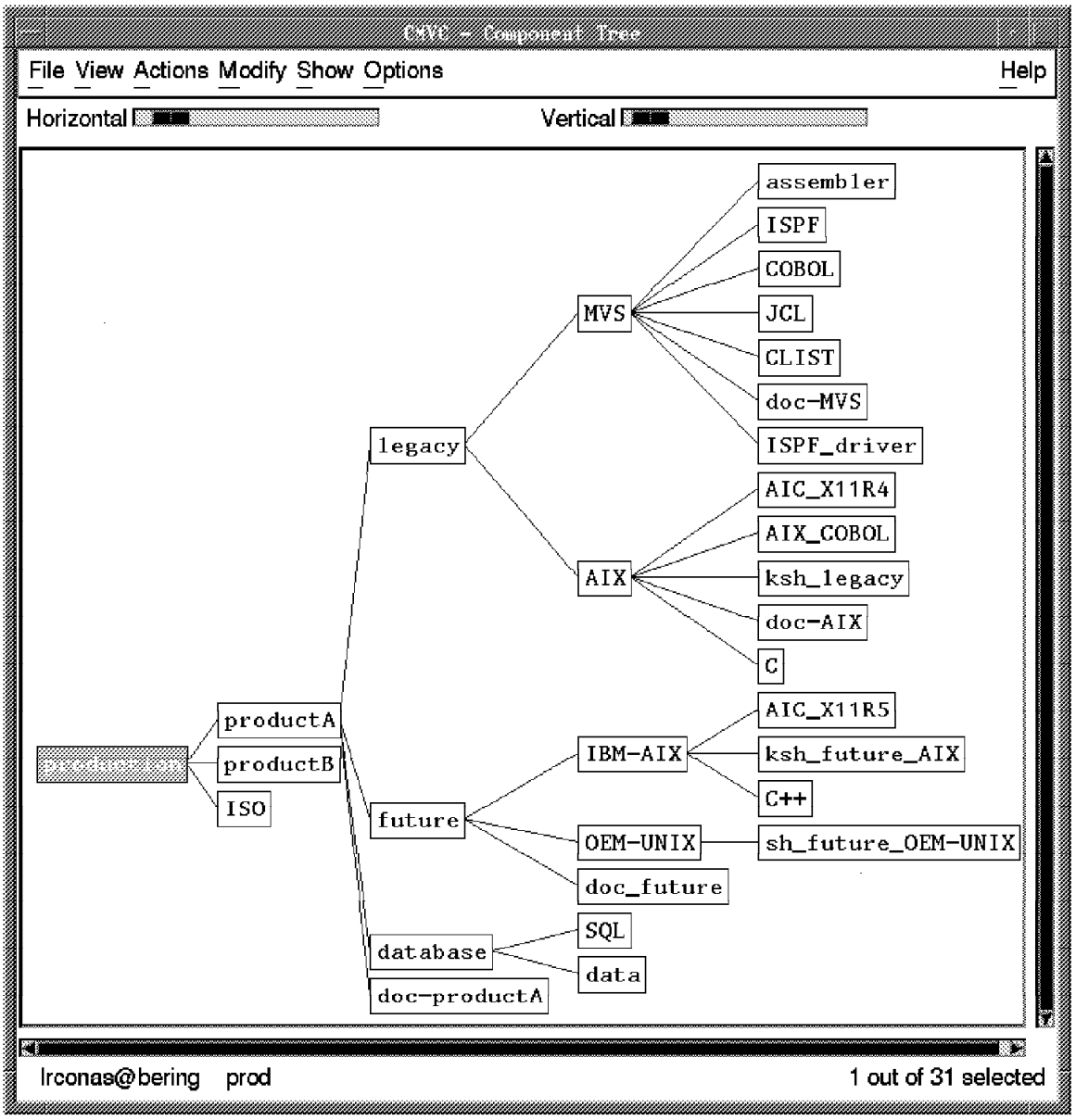

Figure 38. Production Component Tree

The *production* component contained a component for the current software application, *productA*, and a component for a hypothetical second software application, *productB*. Because this level represents the application development organization, the *production* component also contains a component named *ISO*. We intended to use this component to support our organization's efforts at achieving conformance series of quality standards promulgated by the International Organization for Standardization (ISO**), known as ISO 9000. Appendix A, "Implementation of ISO 9001 Using CMVC" on page 167 discusses how CMVC can be utilized to meet some ISO 9000 requirements.

Beneath *productA*, we created the *legacy* component for the original MVS application and its rehosting to AIX. We also created the *future* component for the object-oriented reimplementation of our application. Since both implementations would share a common DB2/6000 database, we created the *database* component at this same level. This would contain the *SQL* for the

scripts required to build and install the DB2/6000 database, *data* for sample data used to create test environments, and *utility* for the source code to utility programs that extracted the data from MVS in a form suitable for inclusion in the DB2/6000 database. There was documentation, which was common to all implementations of the *productA* application. The external interface and high-level requirements, for example, would not change from one platform to the next. For files related to this documentation, we created the *doc-productA* component.

Beneath the *legacy* component we created a component for each platform on which the application would execute, namely *MVS* and *AIX*.

Below the *AIX* and *MVS* components, we created components for each type of source code we expected to develop. “Type of source code” did not mean only compilable programming languages. While C source might be processed by the XL C compiler, some source code modules, such as Korn shell or JCL scripts, would be not be compiled at all. These modules would be interpreted when they were executed. Other source code modules, such as user interface definitions, were input to AIC, a tool, which in turn, would generate compilable source code. Components would contain all versions of the source code modules, regardless of which releases of the application any one version supported. We decided we would have a hierarchy of makefiles, each focusing on processing the modules of a given source code type. So, we decided to store makefiles in the separate language components.

Below the *MVS* component, we created components named *assembler*, *ISPF*, *COBOL*, *JCL*, *CLIST*, and *ISPF_drivers*. The *ISPF_drivers* were C language programs, which worked with an internal tool to emulate the mainframe ISPF function under AIX. We also created the *doc-MVS* component, which contained design and end-user documentation unique to this platform.

Under the *AIX* component, we created components to contain the source code for that platform. The *AIC_X11R4* component was created to contain the interface and callback source code files from which AIC version 1.1 would generate X11R4 compatible C language source code. We did not intend to archive the C language source code, which is generated by AIC, as it is only an intermediate product in the process of generating the GUI. The *ksh_legacy* component was created for the routines that supported installation of the application on the AIX platform. Since we anticipated some commonality in the COBOL source between the MVS and AIX implementations, we created an *AIX_COBOL* component to hold only the COBOL source that was unique to the AIX releases. All common COBOL, plus any source that was unique to the MVS releases, would remain in the *COBOL* component, which was under the *MVS* component. Lastly, we created a *C* component to contain any utility routines, not associated with the callbacks, which replaced assembler routines called in the original MVS implementation.

We followed a similar pattern in creating the component structure below the *future* component. We created a descendent component representing each platform on which the application would execute. During our project, the application would only be implemented under AIX, but we created the hierarchy to reflect the possibility of implementation on other platforms, as well. The *IBM-AIX* component served our present needs, and the *OEM-UNIX* component represented future possibilities. We anticipated that a common set of design

documents might support the object-oriented implementation on multiple platforms, so we created a *doc_future* component to hold them.

Below the *IBM_AIX* component, we created components for each language technology. The *AIC_X11R5* component would contain the interface and callback source code files that AIC version 1.2 would use to generate the C++ language source code for the GUI. Again, we would not archive the intermediate C++ source files generated by AIC. The *C++* component would contain the application source code which was not generated by AIC, but instead was written by hand. The *ksh_future_AIX* component would hold any utilities created to support installation of the object-oriented application on AIX.

The *OEM-UNIX* component would only contain one component, *sh_future_OEM-UNIX*, to hold any platform-unique shell programs necessary for installation under that operating system. We envisioned that the source code for the application and its GUI would port transparently to all OEM UNIX platforms from AIX, or use conditional compilations and/or makefile macros to support any platform peculiarities. We could wait until we knew more before creating additional components for files unique to each platform.

CMVC does not allow any two components to have the same name, so we chose naming conventions for our components, which we hoped would eliminate confusion. Notice that we used both the hyphen and underscore in these names. In hind-sight we see that using only one of these characters would have made typing in the names less difficult. In addition, we failed to use a naming convention that embedded the hierarchy relationships.

Notice that the component hierarchy does not map to the build tree hierarchy, illustrated in Figure 35 on page 63. When our files were placed under CMVC control, however, we specified build tree path names, such as `source/cobol` and `source/c/PortedGUI/AddressChange`. This ensured that when files were extracted to a directory, they were placed in subdirectories based on these path names. For example, if the target directory for a *ReleaseExtract* CMVC action was specified as `/ad/productA/MVS_Release_0`, the cobol source files would be placed in `/ad/productA/MVS_Release_0/source/cobol` and the files for a specific GUI window might be placed in `/ad/productA/MVS_Release_0/source/c/PortedGUI/AddressChange`.

4.4.2.2 Our Development Component Hierarchy

We set up our *development* component hierarchy similarly to our *production* component hierarchy. Because it was primarily an area for testing and getting to know CMVC, it was not as complex. Figure 39 on page 79 illustrates this smaller component hierarchy.

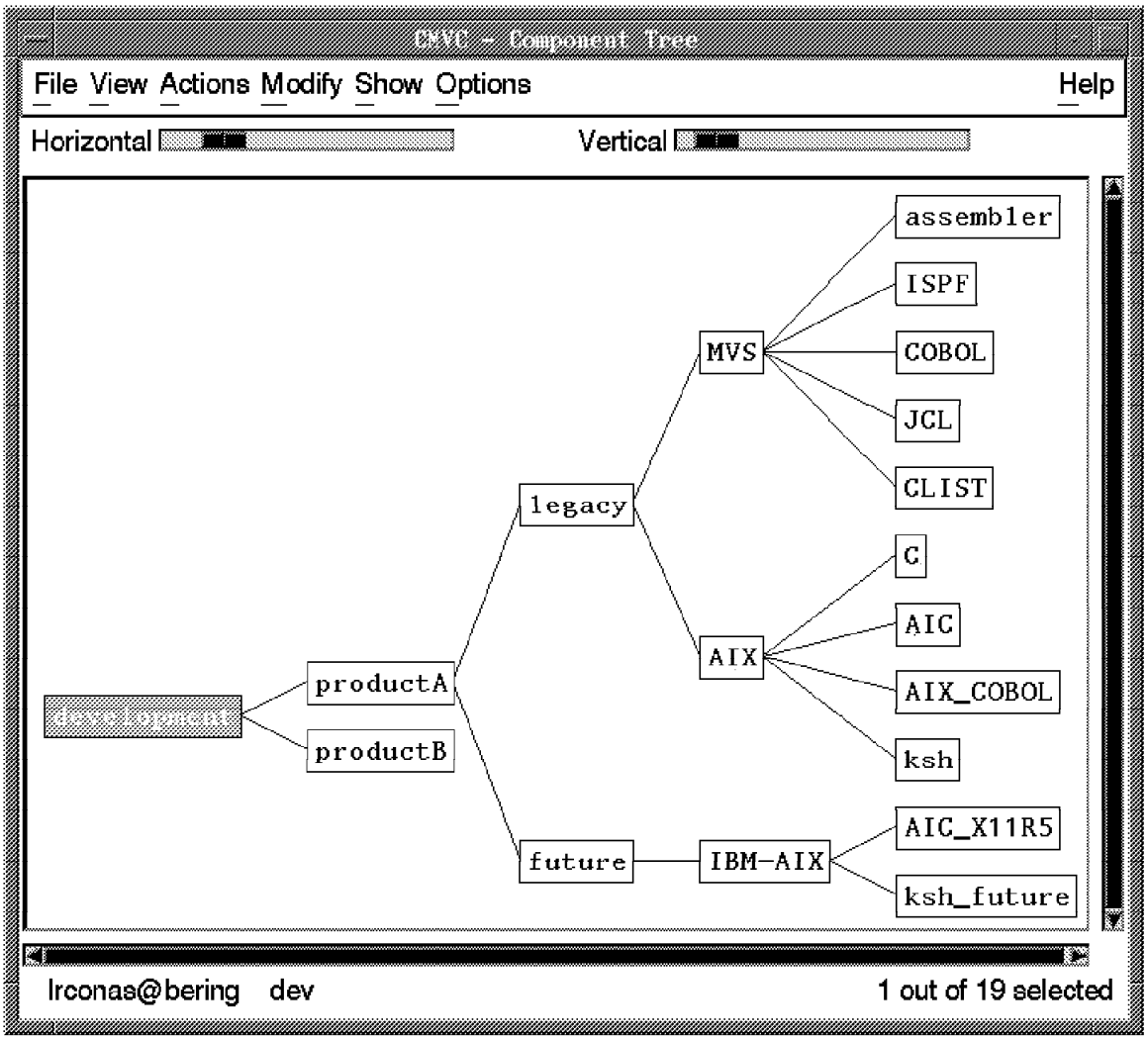

Figure 39. Development Component Tree

4.5 Planning Component Ownership, Access Lists, and Notification Lists

Component ownership is a very important use of CMVC IDs because the owner of a component has implicit authority to perform a wide variety of CMVC actions on that component, the children components, and their files. The default owner of the component is the user who creates it. Ownership may be given to any user by the current owner of a component. No one but the current owner of a component, or the family superuser, can change component ownership.

The component owner can grant explicit authority to other users to perform actions (associated with authority groups) over files controlled by that component. Once authority is granted to another user, that user inherits the same authority over all its descendent components, unless that authority is restricted expressly at a given component. The component owners can remove granted access or restrict any other user's inherited access, provided the component owners have that authority themselves.

A similar relationship exists between component ownership and automatic notification of CMVC actions. A component owner defines the notification list for

a component by placing users in an interest group for that component. These users automatically obtain membership in the same interest group for all descendent components. When a CMVC action occurs, notification is sent automatically through electronic mail to all affected users and owners of affected CMVC entities, such as components and release. It is also sent to users whose names are in the interest list for the affected components, if those lists pertain to the CMVC action.

4.5.1 What to Consider in Planning Component Ownership, Access Lists, and Notification Lists

You must determine the owner for every component in your initial family hierarchy. You need to determine which additional users will require access to these components and, if necessary, modify or create authority groups needed to implement this access. You must also decide if notification lists will be required, and whether the predefined authority groups support your needs.

First, determine the owner for every component. It is best if component ownership reflects the level of responsibility a person has for the files contained in that component. If a single person has overall responsibility for a development effort and several people have been delegated specific responsibility for parts of that effort, it makes sense to have these individuals own the lower level components. The owner of the parent component should be the person with overall responsibility for the project. Determine if a specific CMVC action is implicitly given to a component owner, or must be explicitly granted by means of an access list on that component. Consider the implications of inherited notification and inherited access authority derived from the ownership of a component. See the list of authority and notification for CMVC actions presented in *IBM CMVC User's Reference*.

Then, determine which other users will require access authority to each component. Study the component hierarchy for the implications of assigning individual users to specific access authority groups for specific components at high levels of the component hierarchy. This authority will be inherited at lower levels; consider the appropriateness of this, and modify your plan accordingly. Identify exceptional cases where you might want to restrict this inheritance. Consider actions you might want to restrict to a very few individuals, and map these to the predefined access authority groups. Consider which notifications you may want to broadcast to the widest number of CMVC users, and which you may want to restrict.

With this data, look at the predefined CMVC access authority groups and the CMVC actions they map to, as well as the predefined notification groups. Try to group your users into these categories. If there are cases where you need a class of users whose requirements are not met, define new access authority or notification groups to meet those needs. Be sure you understand which authority is basic to any user, which is implicit to a user who owns or creates CMVC objects, such as components, defects and releases, and which must be explicitly granted.

If you own a component, you can grant a particular access authority to another CMVC user ID that extends to all the files that are managed by the component, but you cannot grant the access authority on a file-by-file basis. Also, that other CMVC user ID inherits the same access authority for all descendent components, regardless of who owns them. It is, therefore, generally better to grant explicit access authority at the lowest applicable level in your component hierarchy.

You must go through a similar evaluation regarding automatic notification to CMVC users of CMVC actions. Refer to *IBM CMVC User's Reference* for a list of users automatically notified when specific CMVC actions occur, and for a list of predefined notification interest groups. If a user is not going to be notified automatically, then you must enter the user's name in a defined interest group in the notification list of the relevant component. While you do not want users receiving more notification mail than is relevant to them, there may be certain actions about which you want all members of the development effort to be notified. In this case, it is easier to place all users in this interest group in the notification list of a component at a high level in your hierarchy. Our experience is that you should not cause too many CMVC notifications to be received at the beginning of your project. As the team members become used to receiving them and learns to react appropriately to them, they will request additional notifications if they are needed.

Over time, as you add components at lower levels, your original access and notification lists for the original components at the higher levels may require revision. You may find that you want to move some interest groups to lower level components, and create new interest groups, as well.

Add a column for proposed CMVC owners, to your matrix of components. Also add also columns for access authorization, access restriction, and notification lists. Table 7 shows part of the table for our project.

Table 7. Component Hierarchy Plan

Component	Description	Interest/User	Authority/User	Access Restriction
legacy	Contains components for the legacy application on each platform (*productA*).	not applicable	not applicable	not applicable
MVS	Contains components for MVS source code (*productA*).	*low*/*branko* *low*/*truls* *low*/*projA_lead* *tester*/*MVStester*	*builder*/*MVSbuilder* *builder*/*krt*	not applicable
COBOL	COBOL sources (`*.cbl`, `*.cob`, `*.cpy`)	not applicable	not applicable	not applicable
ISPF	ISPF panel files (`*.pnl`)	not applicable	not applicable	not applicable

4.5.2 Our Component Ownership, Access Lists, and Notification Lists

The original owner of all our components was the CMVC user ID we created for our team member who had SCM responsibilities, *lrconas*. Ownership of the root component, *production*, was changed to the *projA_lead* CMVC user ID. This belonged to our project leader, who delegated authority to create and delete components to another person in the department, but continued owning this component to ensure receipt of mail about these activities when they occur. The *projA_lead* was also made owner of the *ISO* component. This component was used to manage data related to ISO compliance efforts, and to manage defects representing any deficiencies found during an ISO audit of the department. This component was also used to record deficiencies or features created to ensure changes in the documented procedures for the department.

The *productB* component was given to the *projB_lead* CMVC ID, which would be assigned at a later date. The ownership of the remaining high level components,

productA, *future*,*legacy*, *database*, and *doc-productA* was given to the *projA_lead* CMVC user ID.

We chose to make the *truls* CMVC user ID, which belonged to our developer with COBOL expertise, the owner of the *MVS* component and its descendent components. We also gave ownership of the *SQL* and *data* components (descending from the *database* component) to *truls*. We chose to make the *krt* CMVC user ID, which belonged to our developer with AIC expertise, the owner of the *AIX* component and all its descendent components. We gave ownership of the *IBM-AIX*, *OEM-UNIX*, and *doc_future* components, and their descendent components, to the *branko* CMVC user ID, which belonged to our developer with C++ expertise. Figure 40 on page 83 shows the list of components and their owners for the *prod* family.

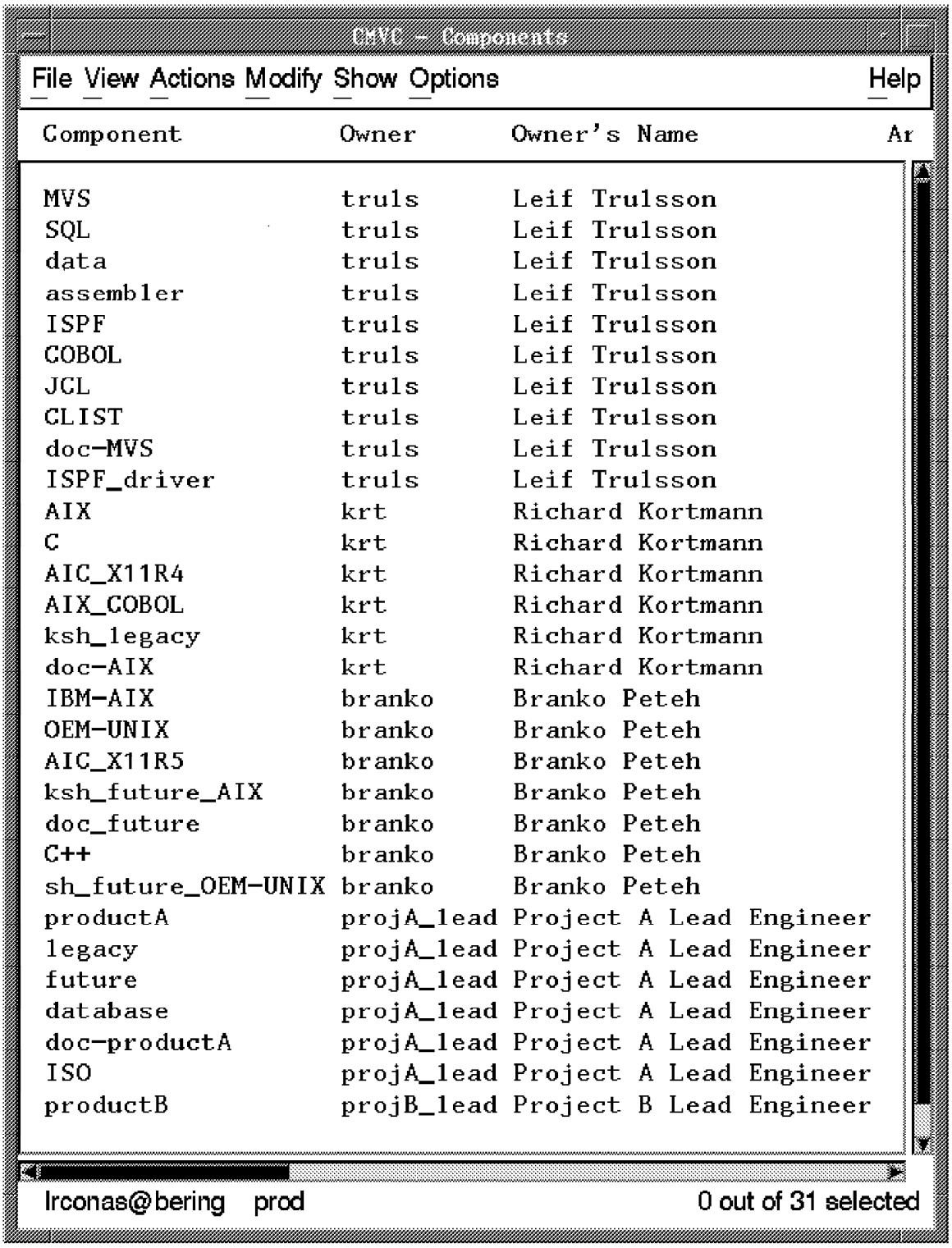

Figure 40. List of Components and their Owners for prod Family

The owners of the top-level components were not burdened by many automatic notifications, as their components were fairly stable. The owners of the lower-level components, where files, defects or features, and releases were being actively manipulated, saw considerably more automatic notification mail. Because all users would want to know about major events, such as release creation, we added the users who were not owners of release managing

components to the *low* interest group for those components. We placed the COBOL expert in the *developer* interest group for the AIX_COBOL component. Figure 41 on page 84 shows which CMVC users were placed in interest groups for components in the *prod* family.

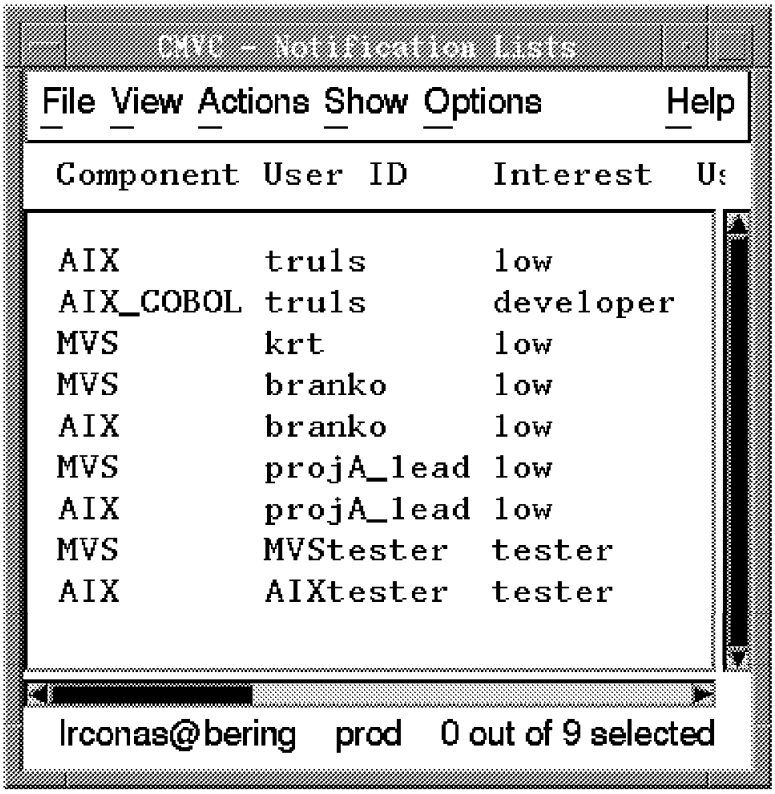

Figure 41. Notification Lists for prod Family

Figure 42 on page 85 shows how we mapped our users to access authority groups for specific components to accommodate these situations.

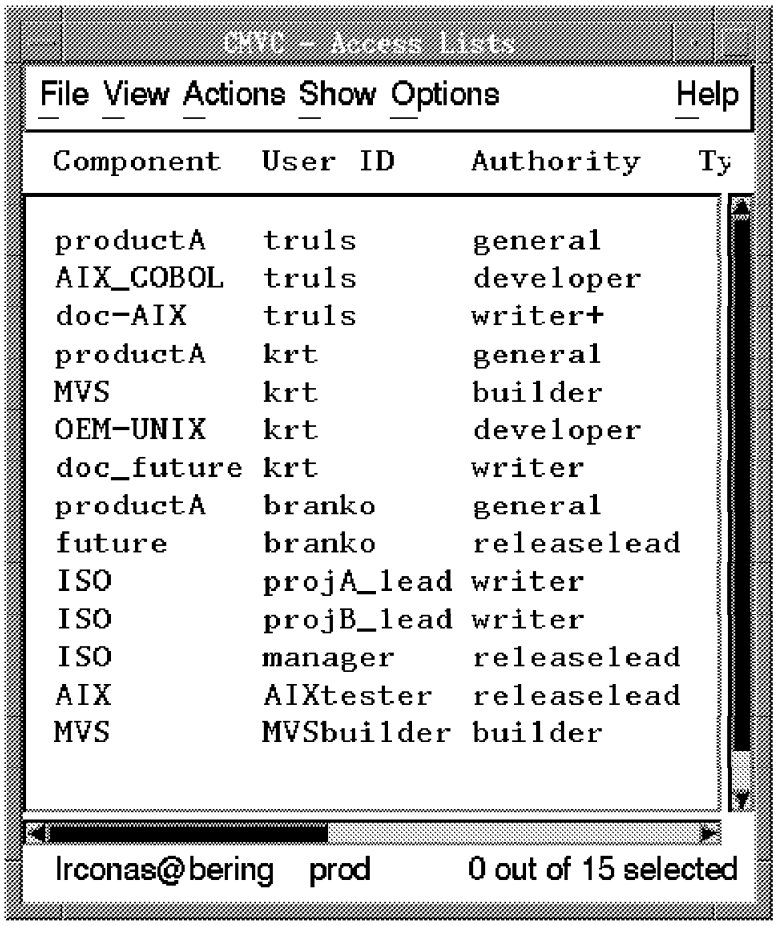

Figure 42. Granted Access Lists for prod Family

4.6 Planning for Files

Dealing with files is straight forward. File versioning and file actions are well documented. The main points to consider at the planning stage are path and file naming conventions, and potential uses for configurable fields.

4.6.1 What to Consider in Planning for Files

Planning the file and directory naming conventions is an important first consideration in planning for CMVC files. Understanding the features CMVC offers with respect to files, and the relationships between files and releases, is also important in planning for files. Finally, even though you can configure new fields in the CMVC File record at any time, the many uses for configurable fields should be considered early in your project.

4.6.1.1 Planning File and Path Naming Conventions, and Build Directory Structure

Plan the file and path naming conventions you intend to use, and the build directory structure (file tree) you want to use when building your application. CMVC records a relative path name which is prepended to the file name when a version of the file is extracted from CMVC. This relative path name is then prepended with an absolute path name generated from either your present

working directory or a specified directory when you perform a CMVC action, such as *FileExtract* or *FileCheckOut*.

Another reason you want to plan out the build directory structure is to segment the files involved in the development effort, so your team members need only deal with files relevant to the work they are doing. You may want testers to have only test cases and related data files in the directory hierarchy they can manipulate. You may not want your writers storing their files in the same directory as the application code. Remember that the hierarchy of directories in which your files are placed when you use them, does not have to map directly or even indirectly to the component hierarchy you create to manage them in CMVC.

The combination of relative path name and file name identifies a unique file to CMVC. CMVC commands use the flags **-relative** and **-top**, or the CMVC environment variable `CMVC_TOP`, to convert absolute path names to the relative path names known to CMVC. Use of these bears close examination as it may not be all that intuitive to the novice CMVC user.

Combining the relative path name and base file name to create a unique file name means that you can have multiple files with the same base file name managed by a single CMVC family. As long as the files are associated with different relative path names, they are different files to CMVC. This requires that you plan under what circumstances you will allow multiple files to have the same name, and plan a directory hierarchy to support this.

One reason you might have multiple files with the same name is to support an application across multiple platforms. There may be modules that are named identically and perform the same role on each platform, but contain radically different, platform-specific source code. These modules are linked selectively with the bulk of the common code to create platform-specific executables.

There are also instances where a file's base name determines how it is used, but not what specific data it contains. For example, the UNIX `Makefile` files are usually named identically, but distributed across the build directory hierarchy, which usually models the application architecture. The `Makefile` file in each subdirectory contains entirely unique build instructions that are pertinent only to the source code extracted to that subdirectory.

4.6.1.2 Planning Relationships between Files and Releases

Do not confuse multiple files that have the same name with multiple concurrent versions of the same file. This is referred to as a single file being "shared" by multiple releases. A shared file is one file that has parallel branches of version history.

To illustrate this concept, assume you are naming the versions of a file with two digits, incrementing the second digit with each new version, such as 1.1, 1.2, and 1.3. This file, which has only one "most recent" version, is associated with a single release. Now, assume that for another release, you want to have a version of the file, which was identical to version 1.2, but with some changes that were not in version 1.3. To do this, you would start a parallel branch of version history, naming the new version: 1.2.1.1. If you continue to make changes to each branch, you might soon find the most recent version on the main branch is named 1.5, while the most recent version on the other branch is named 1.2.1.3. Now you have two most recent versions of this single file. Each is associated

with a different release. When planning your application releases, you will want to consider how to best use this feature.

There is another variation of the relationship between a file and a release. You may want the same version of a single file to be defined in multiple releases. This is referred to as a file that is “common” to multiple releases. Common files are the most efficient way to reuse source code, but when you identify a file as common, you need to evaluate whether the component which manages it is still appropriate. It may be useful to have a separate component to manage files that are common to multiple releases. This is particularly important if the file is common to releases of different applications. Appendix D, “Hints and Tips for Using CMVC” on page 193 gives some examples of shared and common files.

Such relationships may not be immediately evident at the beginning of a development effort. But, where development is spring-boarding from previous work, as was the case with our project, many such relationships can be identified and planned for in advance.

4.6.1.3 Planning Configurable Fields for the File Record

The File record is configurable, so you should consider early on if there are additional characteristics of a file in your environment that need to be recorded and manipulated. There are several circumstances that dictate the use of configurable fields for files.

One circumstance is where you need to query and report based upon various classification criteria that are not readily discernible from the file name or path name conventions or component relationships. Various candidates for a configurable field of this type may be found in data you want to record in a module header.

For example, you may need to record the fact that certain software files were contributed from sources outside your company. These files might contain “free ware,” “share ware,” or code developed by subcontractors, yet are managed by the same component. Since you cannot distinguish the origin of the source code files by the name of the managing component, you need a configurable field in the File record in which you can record it.

Or, you might want to identify the source code language contained in the files. You cannot distinguish between a C++ and a C language include file, by file naming convention, because both types of file names end in “.h.” You can create a configurable field in the File record, store an indication of the source language there, and easily query it in reports.

You may also be need to identify files according to an arbitrary classification of the data they contain. For example, you may have files containing documentation source which represent different types of documents, such as design notation, end-user documentation, or requirements specifications.

Another circumstance is where you want to record additional state or status information about files that is independent of the file versioning information, but related to your process model. For example, you may want configurable fields in the File record in which to note the dates when it passed unit design review, unit code review, or software QA review. Alternatively, you may want to record the fact that a version of the file had entered a significant baseline. This fact can be easily retrieved, if it is recorded automatically in the File record at the time the file is linked to the CMVC release that represents that baseline.

Finally, you may want to do automated processing of all files, and need to record some data related to that processing, on a file-by-file basis. Rather than create a separate database with this information, you could create a new field in the File record and store it there. One possibility is that you want to do some software quality analysis on the current version of all files in a given release, and record the quality metric in the File record.

4.6.2 Our Use of Files

We had a lot of existing code to start with for this project. We also had a clearly articulated plan of development for our initial migration and the object-reorientation of the application. We wanted to experiment with configurable fields, so we made a point of creating one for the File record and exploring its use in conjunction with a user exit.

4.6.2.1 Our File and Path Naming Conventions, and Build Directory Structure

We knew we wanted our three developers working fairly independently, so we structured the directories to support a division of the files according to the division of labor, the languages and tools, and the releases we anticipated. Our component hierarchy classified files by target platform, source language, and division of responsibilities, but the two were not identical. Our decisions and their rationales area are described in detail in 3.4.4, “File System Topology” on page 49.

4.6.2.2 Our Relationships between Files and Releases

When planning our releases, we maximized reuse of source code, so we identified several examples of both common and shared files. The releases for the AIX and MVS platforms had common and shared files managed by the *COBOL* component. The AIX releases had shared and common files managed by the *AIX_X11R4* component, also.

4.6.2.3 Our Configurable Fields for the File Record

We created a LOC field to store the number of lines of code in the program source files. We referred to *IBM CMVC Server Administration and Installation* for instructions on implementing configurable fields. To ensure that the LOC count was up-to-date, we created user exits, which would calculate and record the count whenever a file was created or updated. Our use of this new field and the related user exit are described in 2.2.3, “Project Manager Asks about Implementing Quality Metrics” on page 19.

In addition, when you consider a configurable field for the File record field, remember that you may have multiple parallel versions of the file in use, and that one field may not be enough to accommodate them all. If the field defines a type of data that is likely to be the same for all versions, such as source code language, one field is acceptable. For our project, a single LOC count field is useful only if there is a single current version of this file in all releases. If we had multiple current versions, we would need a release-specific field, and our user exit would need to determine what field to update when it executed. Clearly, the number of configurable fields defined for the File record would evolve as new releases were defined.

4.7 Defects and Features

Defect and feature processing is critical, not only because it enables you to maintain the quality of the application you develop, but also because the reports generated from this data can provide insight into many other management issues concerning budgets, schedules, and productivity. The most important things to decide early about defect and feature processing, is how to map the CMVC model to your actual problem tracking process. It is also good to plan on the possible addition of configurable fields to the Defect and Feature record before you start generating them. You should also plan any modification to the choices lists for Defect or Feature record fields, such as Prefix, Priority, and Severity. You can also automate the defect and feature number generation.

We experimented with the process model, making it evolve over time. We wrote a user exit to generate and record our defect and feature numbers. However, our project made a very simple use of Defect or Feature processing. We do not show a use of new configurable fields, or the many possibilities of report generation in this book.

4.7.1 What to Consider When Planning Defects and Features

In many cases, you have a preexisting means of recording, reporting, and processing reports of defects in your applications or requests for improvements and new features in them. While the CMVC process model is flexible and configurable, it takes some studying to determine the best way to mirror an existing process with it. If you do not have an existing process, the main decision is how rigorous a process you do need. The case for configurable fields in the Defect and Feature record follows similar reasoning. If you are currently using some form of manual or automated problem tracking mechanism, you should study it to see if configurable fields are required. If you do not yet have problem tracking, you may not have any reason for additional, configurable fields in these records.

4.7.1.1 Planning Configurable Component Process Labels

Defect and feature processing is governed by the component associated with the defect or feature. When the component is created, a choice of predefined processes is made. This initial choice can be changed later, as long as there are no active defects or features at that time.

You have flexibility in choosing the name you use for a component process. A component process is essentially a label for a unique combination of Boolean values in a matrix, in which Defect and Feature form the rows, and the two subprocesses, *Design-Size-Review* (DSR) and *Verify* form the columns. Therefore, the *prototype* process represents a “NO” in every intersection of this matrix (no *DSR* for Feature, no *DSR* for Defect, no *Verify* for Feature, no *DSR* for Defect), as shown in Table 8.

Table 8. Prototype Process Definition

Used For	*DSR* **Subprocess**	*Verify* **Subprocess**
Defects	NO	NO
Features	NO	NO

If the labels supplied with CMVC for the processes you choose to use are not meaningful in your environment, you can create redundant labels, which are

more meaningful. For example, you might create a *design* process that is identical to the *prototype* process. *IBM CMVC Server Administration and Installation* contains instructions on changing the component process labels.

4.7.1.2 Planning Configurable Component Process Selection

You can have one group of components governed by one process and another group governed by a different process. You can also configure your process differently over time, to reflect different phases in your development effort. A sample scheme for evolving component processes that uses only the process labels shipped with CMVC, is shown below:

- During your requirements gathering phase, you create a component mini-hierarchy that contains requirements documentation. Because you do not want any defect or feature processing against the files in these components, you create the components in this mini-hierarchy with the *prototype* process.
- After the requirements are defined, you begin designing your application. You want to exert control over changes to the requirements data, but not over the rapidly evolving design data. You change the component process for the requirements components to *default* process and you create a new component mini-hierarchy that contains design documentation. Each of the design components has the *prototype* process.
- Once you begin coding, you change the design components′ process to *default* and you create the code component mini-hierarchy using the *prototype* process for all these components.
- After all the code files managed by a given component pass unit test, you change that component′s process to *default*.
- Ninety days before your deadline for delivering, you change all the requirements, design, and code components to the *preship* process, to ensure that all new features or defects against any component are reviewed for impact on your delivery schedule.

However, it can be important that you ensure that component processes are used uniformly across, a project or a department. A grand scheme is only valuable if it is rigorously enforced. Ensuring that component processes are defined as you plan them to be, may require automation in the form of user exits.

Figure 43 on page 91 shows the list of predefined processes shipped with CMVC. The left column lists the process names and the right column shows the associated subprocesses. This list is displayed if you select **On Process...** from the Help pull-down of any CMVC window.

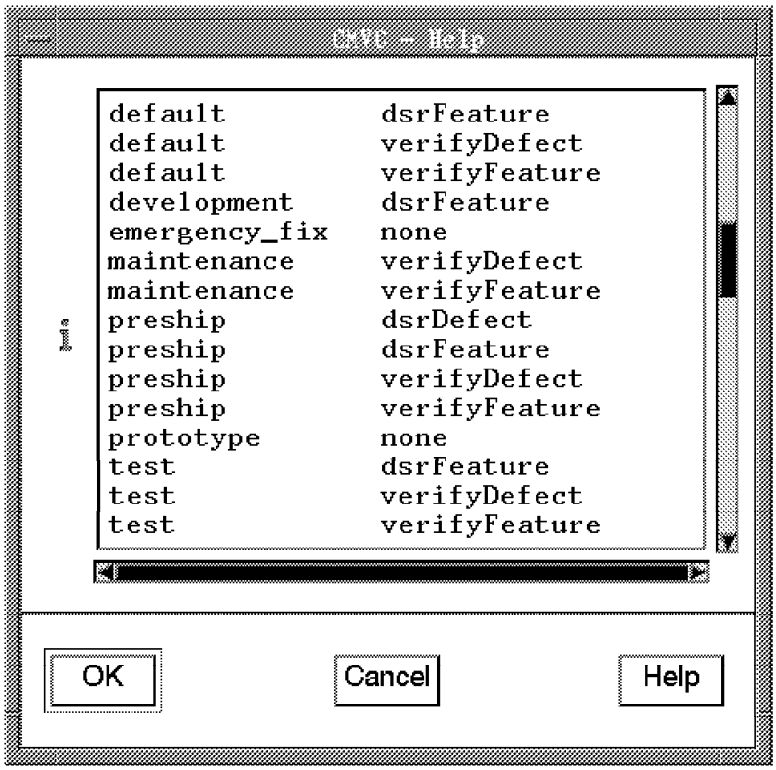

Figure 43. Component Processes Shipped with CMVC

You will probably want to update your list of planned components with an annotation about the process definition for each component. You may need multiple columns, if the process changes over time.

4.7.1.3 Planning Defect and Feature Number Generation

Defect and feature numbers are treated by CMVC as a single set, that is, no defect or feature can have the same number as any other defect or feature in a family. They need not actually be a number, because they are treated as a string. You may not find CMVC′s default numbering scheme appropriate for your needs. For example, you may find that you need to use defect or feature numbers generated by another problem tracking system. You may want to automate the selection of the number, so you can embed other information in it. You may need to intermix numbers originating from several sources, but validate user-supplied numbers against certain criteria. Some mechanism for ensuring unique and meaningful defect and feature numbers needs to be implemented prior to allowing users access to CMVC.

4.7.1.4 Planning Configurable Fields and Choices Lists

If you have a preexisting problem tracking process, you probably have the equivalent of a Defect and Feature record on paper or in a computerized record. You will need to map the types of data you currently record to the fields in the Defect and Feature record. You will also want to map the range of values you accept in any specific field to the choices list associated with it. You may find that an existing field serves your purpose, by simply changing the choices list. *IBM CMVC Server Administration and Installation* contains instructions changing the choices lists. You may also find you need to create additional fields. Defects and Feature records share the same record template in CMVC, but the user interface requires fewer input fields for a feature than a defect.

You may find that a static mapping fields or choices is not sufficient. Not only can the set of fields you need evolve, but you may find it useful to modify the choices list for a specific field as your project enters different phases of the development life cycle. For example, you may find that a set of six Priority values will work in the early development phase, but as you near a delivery deadline, you must convert these to one of just two values, indicating that the problem must either be solved before the deadline, or deferred to another release. After you pass the deadline, you may need to convert these values again when you create new tracks for them in another release. This is a good time to plan tools to automate this type of transition, too.

4.7.1.5 Planning Changes to Access Authority for Defect or Feature Processing

CMVC recognizes and provides for various checks and balances in the development process, however, they may not meet all of your needs. This is something to consider when planning to use CMVC.

For example, in some projects, only a software configuration manager, Software QA representative, or software librarian, as representatives of a Software Change Review Board (SCRB) may be authorized to move a defect or feature through the *DSR* states. An access authority group defined for only those CMVC actions, is not shipped with CMVC. However, the defect owner has implicit authority to perform these actions. You might want to change this by defining a new authority group and calling it *software_librarian*. You could then explicitly deny authority for these actions to the component owner, and explicitly grant it to the librarian, for all components that use the DSR subprocess.

4.7.2 Our Use of Defects and Features

Whether we should have used a feature or defect to initialize our baselines, or to address the changes we made in the code to migrate it to AIX, may be a philosophical question. We chose to use defects for the most part, when we established new product baselines or made changes to existing files. When we created new function, such as when we introduced the first GUI, we chose to use a feature. Our basis for these decisions was arbitrary, but in your circumstance this decision may be more serious.

Figure 44 on page 93 shows the details of a feature opened to authorize work that modernized the GUI of our ported application.

Figure 44. Feature Information Window

Figure 45 on page 94 and Figure 46 on page 95 show the contents of a defect opened to bring our AIC callbacks and interface files under CMVC control for our initial AIX release. The Defect record is long, so this defect is in two figures. The position of the scroll bar on the right side of the window shows the relative position of data. This defect contains some information that resulted from our decisions on the component process selection, the defect number generation, and the choices field of the *defect acceptance* field (labeled “answer”in this window). These decisions are described in the remainder of this section.

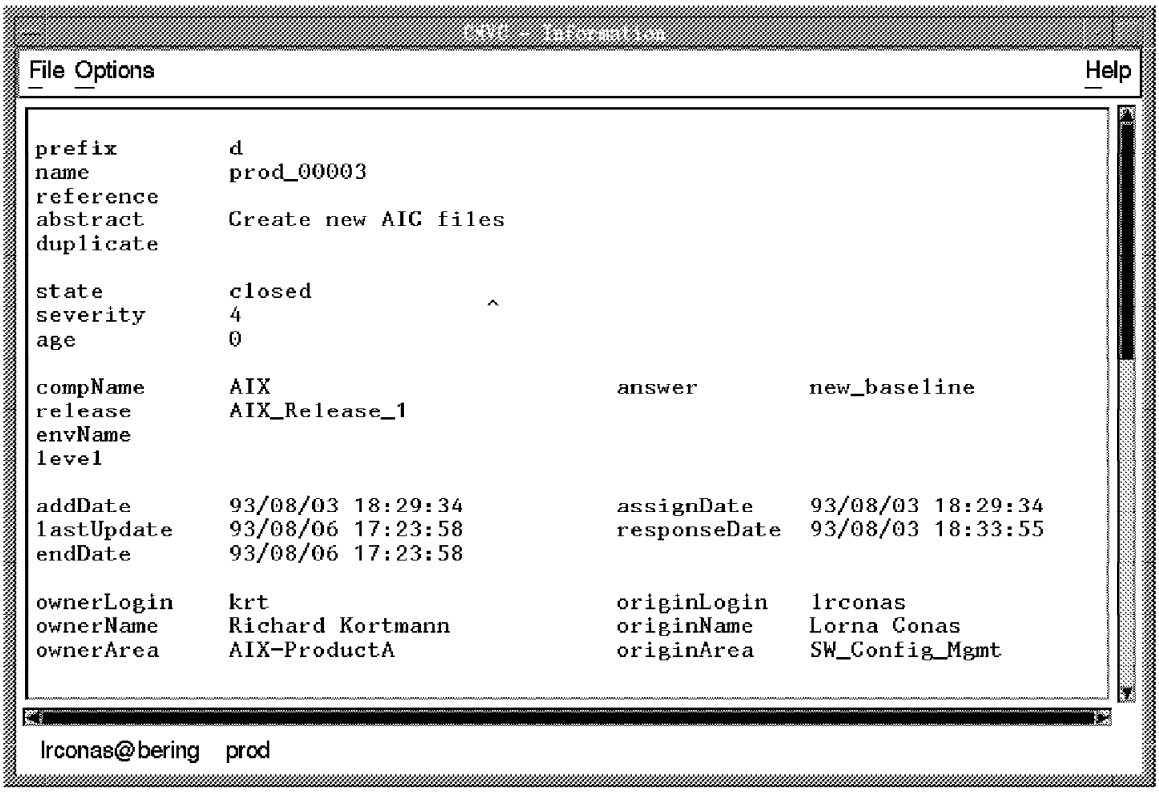

Figure 45. Defect Information Window (Top Half)

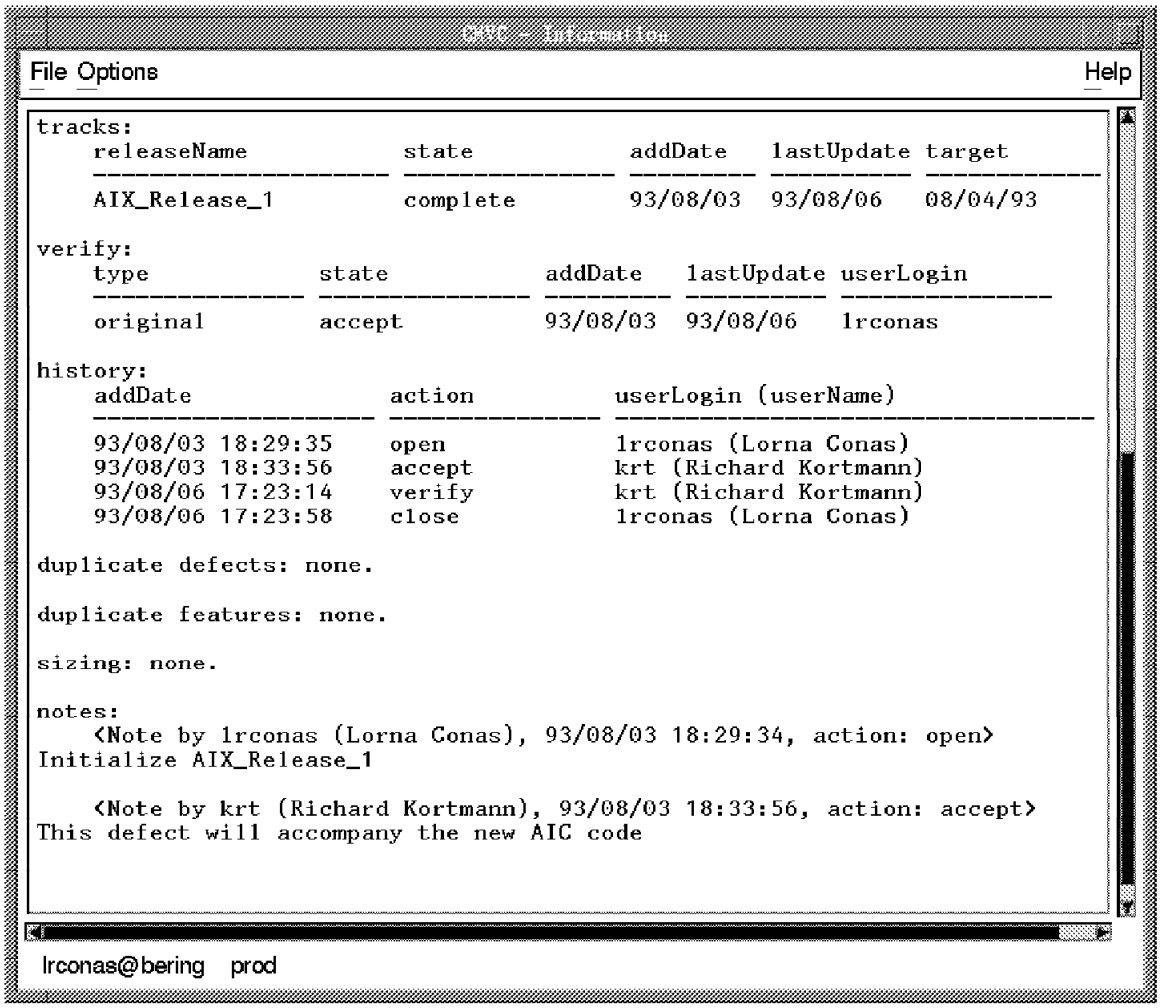

Figure 46. Defect Information Window (Bottom Half)

4.7.2.1 Our Configurable Process Selections

Our project was not large and it did not have formal baselines and complicated issues to manage. We chose a simple combination of configurable component process selections across all our components.

We decided that the *maintenance* process was adequate initially for all components. This process required verification of both features and defects, but it did not require the *DSR* subprocess. In the case of the components containing COBOL, where we had a working application, and wanted to track carefully changes made to it for the migration. We chose this process for the components containing newly developed code also, so we might gain experience with defects and features, even though we did not need such a rigorous control over that code.

We moved components to the *preship* process after a baseline had been tested and evaluated, because we wanted to force the developers to slow down long enough to consider the impact of any new changes they introduced into the application baseline. We left unchanged components, which contained no files, or which represented documentation that would not be updated with each of the software releases.

Table 9 on page 96 shows a list of all the components in the *prod* family with the component process definitions as they began, and as they changed after the baselines were established and evaluated.

Table 9. prod Family Component Processes

Component	Original Process	Final Process
MVS	*maintenance*	*maintenance*
SQL	*maintenance*	*preship*
data	*maintenance*	*maintenance*
assembler	*maintenance*	*preship*
ISPF	*maintenance*	*preship*
COBOL	*maintenance*	*preship*
JCL	*maintenance*	*preship*
CLIST	*maintenance*	*preship*
doc-MVS	*maintenance*	*maintenance*
ISPF_driver	*maintenance*	*preship*
AIX	*maintenance*	*maintenance*
C	*maintenance*	*preship*
AIC_X11R4	*maintenance*	*preship*
AIX_COBOL	*maintenance*	*preship*
ksh_legacy	*maintenance*	*preship*
doc-AIX	*maintenance*	*maintenance*
IBM-AIX	*maintenance*	*preship*
OEM-UNIX	*maintenance*	*preship*
AIC_X11R5	*maintenance*	*preship*
ksh_future	*maintenance*	*preship*
doc_future	*maintenance*	*maintenance*
C++	*maintenance*	*preship*
productA	*maintenance*	*maintenance*
legacy	*maintenance*	*maintenance*
future	*maintenance*	*maintenance*
database	*maintenance*	*maintenance*
doc-productA	*maintenance*	*maintenance*
ISO	*maintenance*	*maintenance*
productB	*maintenance*	*maintenance*
production	not applicable	not applicable

4.7.2.2 Our Defect and Feature Number Generation

We automated the generation of defect and feature numbers, primarily as an experiment in using user exits. We chose to use a single sequence of numbers for both defects and features, and to embed the family name in the numbers. Our defect or feature numbers took the form of dev_0005, for the *dev* family, and prod_00012 for the *prod* family. We also could have created two sequences of numbers, one for features and one for defects.

Refer to C.2, “User Exit to Generate Defect or Feature Number” on page 185 for a detailed discussion of this.

4.7.2.3 Our Configurable Fields and Choices Lists

We did not like all the choices lists shipped for *defect acceptance*, *priority* or *prefix*. However, we only took time to modify the *defect acceptance* choices. We added a choice labelled “newbaseline,” which indicated that the defect and changes related to it were for the purpose of establishing a new baseline. Figure 47 shows the complete list of choices after this modification.

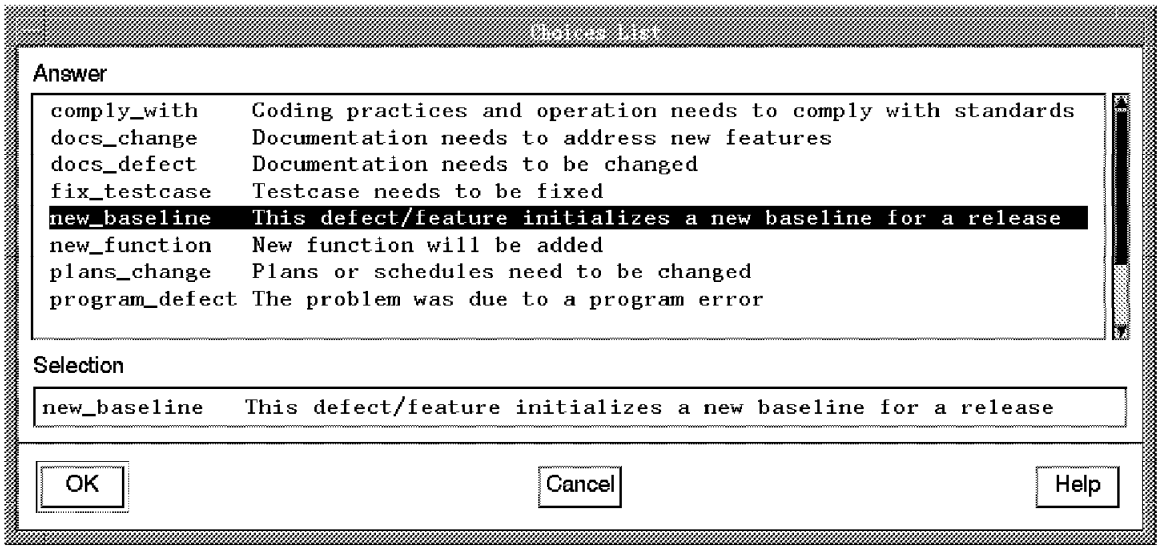

Figure 47. Defect Answer Choice List Customization

4.8 Releases, Levels, and Tracks

There are clear and precise rules relating to CMVC releases, levels, and tracks, and their relationships to components, files, defects, and features. There are many ways to make use of these CMVC objects in the context of software configuration management, but not all of them will be obvious to every CMVC user, and many will seem contradictory.

4.8.1 What to Consider in Planning Releases, Levels, and Tracks

A CMVC “release” is a means of grouping versions of files in the family. A CMVC release can be used to identify the exact version of all the files that comprise an application at a significant point in time. It can be used to establish an application configuration baseline. A release can also be used to represent a horizontal subset of an application, such as an end-to-end build of a functional thread (a test driver). It can be used to identify a vertical subset of an application, such as the user interface or batch processing subsystem. It also can be created to identify the set of the end-user documentation, which corresponds to a release of the code. A release can be used to represent one of several customer-specific or platform-specific builds of a complete application at one level of function.

In part, this flexibility is due to the fact that CMVC recognizes no relationship between any two releases, nor imposes any restrictions on what names you give releases, so long as they are unique. You can, however, indicate to your users

that releases are sequential, parallel, or related in some other way by how you name them.

A CMVC “level” identifies a set of “tracks,” which are being incorporated into a release. It can be used to divide and organize sets of file additions and changes, so they can be incrementally extracted from CMVC to be integrated and tested. This subsetting can be horizontal or vertical across the application, or merely subsetted by sequential arrival, such as a weekly compendium of fixed defects. However, a level can be used to identify a platform-specific build, because multiple levels can contain a given set of tracks. Levels can be named as you wish, and CMVC imposes no relationships between levels of a given release.

A track is CMVC′s method of indicating that the set of file changes or additions implementing a given defect or feature are to be integrated into a specific release. There is relatively little ambiguity about how a track is used, but use of tracks is optional and varies with the chosen release process.

When a release is created, a release process is chosen for it. A release process is a name for a predefined combination of release subprocesses. These subprocesses govern how a track (representing a single defect and feature in a single release) is integrated into the application. The release subprocesses are likely to be used when:

Subprocess	**Reason to Use**
track	You need to trace changes in the release to specific features or defects.
approval	You need rigid control over what changes enter the release and want to ensure that one or more persons authorize it.
fix	You want to ensure that the component owner of the affected files reviews the changes pertaining to a track (and therefore, a defect or feature).
level	You want to create updates for a release according to groups of defects or features.
test	You want to ensure that a release is tested in several environments, such as versions of the operating system, or separate platforms, or by independent testers.

Deciding if and how to use these subprocesses is a task of early planning, if you can readily identify some of the releases you will use. The process governing a release, like a component process, can be modified after the release has been created, adding to the options you have in using releases. You might decide that you require only *track* during early integration and test of a release. But as you near the deadline to ship, you might decide to place extra scrutiny on which changes entered the release, and change the process to include *approval*. The release may grow to an unmanageable size and force you to include *fix*, *level*, and *test* so more people are involved in verifying the correctness of the changes.

CMVC ships a list of predefined release processes, but the CMVC family administrator may modify these or create new release process definitions. The definition represents a combination decisions to include or exclude each of these subprocesses. Like component processes, new definitions may be created,

which are redundant with preexisting labels. See *IBM CMVC User's Reference* for a list of the shipped CMVC release processes.

Files and releases have in common the fact that each is managed by a component that determines notification interest and access authority. A release is managed by a component, but can include files from any component in the family. Again, there is room for creativity in choosing the components to manage your releases. You may choose to create high level components whose job is just to manage releases. You may choose to attach a release to any component owned by the CMVC user who will be responsible for that release. You may choose to associate the release with a component that represents the application architectural subdivision.

A release has an owner, who is independent of the owner of the component, which manages the release. But, a component owner, in fact, does not have implicit authority to create any release. Explicit authority to create and modify releases must be granted by the family superuser. Once granted, the user can create releases managed by that component or by a descendent component. A release owner shares certain access authority over the files contained in the release′s managing component, and inherits this authority over all descendent components. The release owner does not have the same access authority over files belonging to components,which are not descendents of its managing component unless explicit authority is granted to the release owner.

Generally, you will want the release to be managed by a component beneath which descend many of the components that will someday contain these files. If you intend to define files in components that do not descend from the release′s managing component, plan to explicitly grant *releaselead* or equivalent authority access, to the release owner over these components (or some set of components beneath which they all descend). You may have to weigh the decision between making a convenient single explicit grant of access authority at some high level, versus the more tedious multiple grants at the lower levels, if not all descendent components are relevant to the release. You may need to deny access authority at specific lower level components. Update your component hierarchy planning matrix accordingly.

4.8.2 Our Releases, Levels, and Tracks

Our application development effort required the definition of several CMVC releases, each of which was constructed by integrating several CMVC levels. These releases were named:

- *MVS_Release_0*
- *MVS_Release_1*
- *AIX_Release_1*
- *AIX_Release_2*
- *OO_Version_1*.

The *maintenance* process involved *track* subprocess, which we felt was essential to maintaining control over the application baseline. It also included the *fix*, *level* and *test* subprocesses. We felt it important to identify every file change that joined a release in terms of the managing component; therefore, we included the *fix* subprocess. We also felt that levels would make it easier for the developers to work independently and integrate portions of the baseline separately. For example, in the *AIX_Release_1* release, it made sense to integrate the main module and the GUI in one level and test them calling stub programs. We did

not choose the *approval* subprocess, as we did not intend have a separate review board. Design decisions were made as a team, but deciding which tracks went into which levels was handled by the single developer responsible for each separate release.

The *MVS_Release_0* release consisted of the source that produced the currently running MVS application. This release included files added to the *assembler*, *ISPF*, *COBOL*, *JCL*, and *CLIST* components, which comprised the actual application, and files from *ISPF_driver*, which supported AIX testing of this MVS code. These components all descended from the *MVS* component, which itself descended from the *legacy* component. All of these components were owned by user *truls*, and it was an easy decision to have the *MVS* component manage this release, and give ownership of the release to the *truls* CMVC user ID.

We opened a defect to authorize the file changes, and created a track to associate the file changes authorized by this defect with the *MVS_Release_0* release. We created this release to use the *maintenance* release process. Files were placed in this release using the *FileCreate* CMVC action, referencing the track. The original files were taken from the *projectA_proto* portion of the common development tree. For build testing of the release, we extracted these files from CMVC to the `/ad/productA/MVS_Release_0` directory and, then uploaded them for compilation and execution under MVS. Refer to Figure 38 on page 76 for a review of the *prod* family component hierarchy.

As we explored the issues in using AIX as a maintenance platform for MVS targeted code, as well as the issues of making AIX the application′s new target platform, we proposed that for a period of time, we would want to execute functionally identical versions the application on both platforms. While there would be many differences in the user interfaces of these two versions, the code COBOL application logic would remain the same in both. The strategy we proposed involved using as much common (truly identical) COBOL code in the two platforms′ versions as possible. This required some minor restructuring of the COBOL compilation units on the MVS platform, to isolate the common code in modules which could be copied at compilation time into the COBOL source code of either the AIX or MVS version. The *MVS_Release_1* release was created to contain these changes in the MVS COBOL source code.

The *MVS_Release_1* release contained only files descended from the *MVS* component. The *MVS_Release_1* release was owned by the same CMVC user ID that owned *MVS_Release_0*, and it used the same release process. Files were initialized in this release by means of the *ReleaseLink* CMVC command, creating this release essentially from the current version of all files in the *MVS_Release_0* release. We opened defects to accommodate changes to the COBOL source code that is described in the preceding paragraph. We created tracks to associate these defects with the file changes we wanted in this release. We broke the common link between the two releases on those files that would have different versions in each release. The original source of the new versions came from the portion of the common development tree descending from *projectA_proto*. We extracted the new versions of the files in this release to the `/ad/productA/MVS_Release_1` directory, and then uploaded for compilation and execution under MVS.

The *AIX_Release_1* release established the initial AIX baseline. This release contained files controlled by components descended from the *AIX* component (*AIC*, *AIX_COBOL*, *C*, and *ksh*) as well as some files from the *COBOL* component

descended from the *MVS* component. This release was initially owned by the *krt* CMVC user ID that also owned the *AIX* component, and its release process was defined at the start, as *maintenance*.

We created defects and tracks for the initialization of the COBOL, AIC, and main routine source modules. The first files we integrated into the release were the AIC interface and callback source modules, which had been tested independently of the COBOL source modules. We created a level to integrate the AIC code in the release, first. Then, we brought the files under CMVC control using a defect and track. We linked additional files this release from the *MVS_Release_0* release, through the *FileLink* CMVC action, citing the same defect and track. This required that we update several callback source files, referencing yet another defect and track. We then extracted the release to the /ad/productA/AIX_Release_1 directory for integration and final testing. When testing was completed, we closed all defects. Then we changed the component process for all related components to *preship*. The release process was changed to *preship* also, to ensure that no features or defects entered this baseline without careful evaluation. We also changed the owner of this release to the *AIXtester* CMVC user ID. We intended to freeze this release after it was tested and proceed with a new AIX release.

We created the *AIX_Release_2* release to define the AIX baseline after some improvements were made to the GUI. These improvements made the GUI more object-oriented. This was done with the transition to an object-oriented implementation in mind. The new release was owned by the *AIXtester* CMVC user ID and used the *maintenance* process. It included all the files defined for *AIX_Release_1*. We created it by linking it to the *AIX_Release_0*. We opened defects to accommodate the changes in the AIC interface files. We checked out, modified, and checked in these files to resolve the problems cited in the defects. We created tracks so we could integrate these file changes into levels of the release. Then we extracted the files to /ad/AIX_Release_2 and tested. Once all the levels of the release tested successfully, we upgraded its process to *preship*.

We created the *OO_Version_1* release, by extracting the AIC files from the *AIX_Release_2* release to new development directories that were not those defined by the path name CMVC associated with these files. This illustrates another way in which files might migrate from one release to another. We did not check these files back in as new or parallel versions of existing files, but created them as entirely new CMVC files, by associating them with these new directories (path names) in development directory. We managed these new files by the same component as the previous ones, because they were the same type of source code files.

We took this approach, because it was less work to modify the Improved GUI source files to create the object-oriented GUI than it would have been to create all new files. When we were done, however, these files had changed so much that it would not have made sense to treat them as simply a newer, parallel version of the old files. For example, the interface files were modified so that the callbacks were placed inside the interface, instead remaining as separate compilation modules.

AIC generated the C++ source files for the object-oriented GUI from these AIC interface files. We created the brand new C++ modules in this directory too. We created the new CMVC files using the *FileCreate* CMVC action. This release was owned by *OOtester*, and used the *maintenance* release process. We extracted

the files of this release to /ad/00_Version_1 where the release was tested. Again, when this release was successfully tested, we upgraded its process to *preship*.

Figure 48 shows which releases we established and which component and user owned them during the early part of the project.

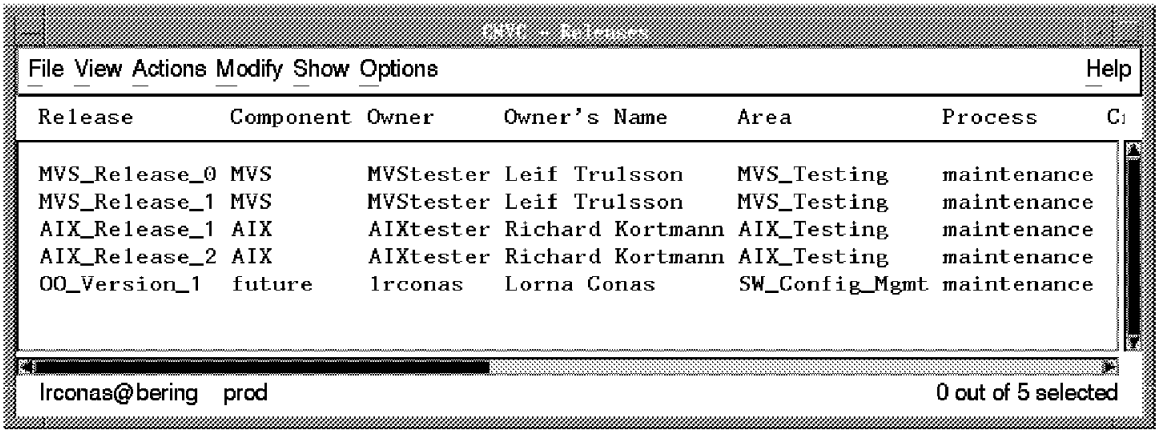

Figure 48. List of Releases with Components, their Owners, and Processes

We found that this scheme of release ownership and component ownership forced us to place explicit *builder* access authority for the integrators at the components that were not descended from components that they personally owned. This was true also, when we granted *releaselead* authority to *AIXtester* and *MVStester*. Level actions were covered by these authority groups, and were not inferred automatically with *releaselead* authority, nor with component ownership.

Chapter 5. Using CMVC

In this chapter we show you how to use the CMVC graphical and command-line client interfaces to accomplish the software configuration management goals described in Chapter 2, “Discovering CMVC: An New Application Project Is Introduced to CMVC” on page 11 and Chapter 4, “Planning for CMVC” on page 65. This chapter gives relevant sequences of CMVC commands and GUI interactions that tailor the CMVC server and create your CMVC objects, such as users, hosts, components, rele and files, for this specific project. This chapter also discusses the relationships and interactions among these CMVC objects.

5.1 First Things First

The sections that follow offer some general advice to the new CMVC end user on getting started with CMVC.

5.1.1 Background Reading

The first thing you should do before you start using CMVC, as with planning, is to consult the CMVC product manuals. You will find that the complete details of all of the CMVC windows and commands are documented in the *IBM CMVC User's Reference* and *IBM CMVC Commands Reference*. These documents define each command or GUI window separately. They identify the prerequisites to the CMVC actions associated with the command or window, the possible command parameters and options, or menus and menu options, and expected results. Each command or action is discussed in isolation, however. These documents do not provide scenarios involving the correct sequences of commands (or GUI actions) that achieve typical SCM goals in a practical context.

IBM CMVC Concepts provides a comprehensive look at what you want to accomplish with CMVC, and how CMVC concepts, objects, and actions enable you to do this. This document provides invaluable insight by diagramming the relationships among the defect and feature states, track and level states, and subprocesses and track states. This document, however, gives no specific details about which CMVC actions involving specific windows or commands should be performed first, second, third, and so forth.

IBM CMVC Server Administration and Installation describes many of the things you need to plan and execute when installing and beginning to use CMVC. It goes into great detail and addresses many system administration issues that we did not investigate at all during this project.

5.1.2 Becoming Familiar with the CMVC Client GUI

Before trying to accomplish anything in CMVC, we recommend that the new user spend some time becoming familiar with the GUI. “Part 1. Using the CMVC Graphical User Interface” in the *IBM CMVC User's Guide* does a good job of introducing the window hierarchy, the parts of windows, the common windows and other graphical elements that recur throughout the GUI. The sections that follow describe our experience in using the stand-alone GUI and from tools that are integrated with SDE WorkBench/6000.

5.1.2.1 CMVC Window Queries

One thing to be constantly aware of is that the information shown in the GUI windows depends at all times on the contents and timing of the last query issued for that window. A query is issued and the window contents are updated, under the following circumstances:

- The default query is issued when you initially request the window to be drawn.
- The last query issued is reissued if you cause the window to be refreshed explicitly by selecting **Refresh Now** on the View pull-down).
- The last query is issued if you have toggled Auto Refresh on—**Auto Refresh** on the View pull-down in the current window—and subsequently perform an action *in the window* that causes the current display to become out of date.
- A new query is issued if you select **Open List...** from the File pull-down, specify query parameters, and select **OK**.
- A query for the window is issued because of a selection you made in another window on the Show pull-down; for example, **Tracks** on the Show pull-down of the CMVC - Releases window causes an update in the CMVC - Tracks window showing all tracks related to the release highlighted in the CMVC - Releases window.
- The default query is issued if the window comes up as a result of selecting a CMVC - Tasks window action item.

While the window is being displayed it will not be updated by actions you take in other windows that modify the database or by database changes caused by other users since the last query was issued. Another thing to remember is that deiconifying a window does not cause it to be refreshed either. **Refresh Now** or **Auto Refresh** must be selected from the View pull-down in each window to update the contents of the window. Taking these actions in one window does not have influence over other current or future windows.

In an environment where many users are frequently using CMVC, you should refresh your screen frequently if you depend on the accuracy of its contents. If no objects are listed, or if those objects you had hoped to see are not present, it does not necessarily mean that they do not exist. It may mean only that the query may not be designed as you expect it to be. It may also mean that there is no default query being issued at all. If you just performed an action that should have updated the window contents, it may simply mean that you have not toggled the **Auto Refresh** on for that window, or that you did not perform the action in this window.

We recommend that if you are a new user, you should explicitly set a default query in every window, which will initially match any and all possible objects that could be listed in that particular window. As you become more familiar with CMVC and its GUI, and as the project database grows, you will arrive intuitively at better default queries. For instance, initially select all defects, sorted by date as the default query in the CMVC - Defects window. Later, you will find that a query on all open defects on components owned by you is more useful, if you are a developer. In many environments, the component owner will be the developer responsible for fixing defects in the code managed by that component. If you are a casual end user, but not a developer, you may prefer a default query for all defects you originated.

To set the most recently issued query as the new default query for a window, select **Current Query** in the File pull-down, and then **Save As Default**. This updates the `.cmvcrc` file in your home directory.

To perform a query in a CMVC window follow these general directions. To issue a new query in a window, select **Open List...** on the File pull-down. In the resulting dialog box, you see an input field for each field in the record which is displayed in the window. The combination and labels on the fields displayed change from window to window, because each CMVC window is a means of viewing records in a particular table. The dialog box is labelled `Open XXXX List` where "XXXX" represents the name of the CMVC window from which you selected **Open List**. The button on the left offers a selection of SQL operators, such as `in`, `not in`, or `like`. Making a choice of SQL operator and then filling in the input field next to a given label ensures that the records retrieved meet certain criteria.

For example, assume you are looking at the Open Component List dialog box. If, next to the field labeled Components, you select the SQL operator **like** and you enter the value `AIX%` in the input field, you are indicating you want all records with a value in the component field, which begins with the string "AIX." The character "%" is used in SQL queries, like the "*" is used in UNIX commands, to match a pattern of 0 or more arbitrary characters.

The button on the right, **Sort**, offers a choice of how to sort the resulting records with respect to the fields in the records. You can indicate you want sorting done first, on one field, second, on another, and third, on yet another. To do this, select **Sort** next to a field. Then select **First** from the menu that pops up. Select **Sort** next to another field, and select **Second** from the pop-up menu associated with that input field. If necessary, select **Sort** next to a third field, and then select **Third** from the pop-up menu there.

To continue our example, you might indicate first that you want records sorted by the value in the *component owner* and then all records with an identical *component owner* value sorted by the contents of the *component name* field. Having made your selections and filled in the data fields on this dialog box, select **OK** to execute the query. The dialog box closes, and the resulting list of records is displayed in the original window.

5.1.2.2 Tasks Window Tailoring

We recommend that you put some of these default queries on the CMVC - Tasks window as new tasks if you do not have standardized CMVC - Tasks window tailoring for different types of users. We recommend a way to use the `.cmvcrc` file to standardize CMVC - Tasks window tailoring in Appendix H, "Tailoring CMVC Windows for Different Types of Users" on page 217. You may want to revise the CMVC - Tasks window as you become more familiar with CMVC.

The tasks defined in the CMVC - Tasks window, as it is shipped, do not all work immediately.

Figure 49 on page 106 shows the CMVC - Tasks window as it is shipped. These tasks provide a template query, which you may need to modify, before it will work. To view the template query, execute the task and wait for a window to be displayed. In that window, select **Open List...** from the Actions pull-down. In the Open List dialog box that is displayed, select **Command Box**. Select the last query shown in the scrollable history pane to display that query in the text entry

area. Edit it, replacing the generic words with a legitimate value. For example, replace “yourRelease” or “yourComponent,” with a valid release or component name in your family, then select **OK**.

To add a new query to the CMVC - Tasks window, make that query in the appropriate window. Select **Current Query** in the File pull-down, and select **Add to Task List...**. You will be asked for the descriptive text, which should be placed in the CMVC - Tasks window to trigger this query.

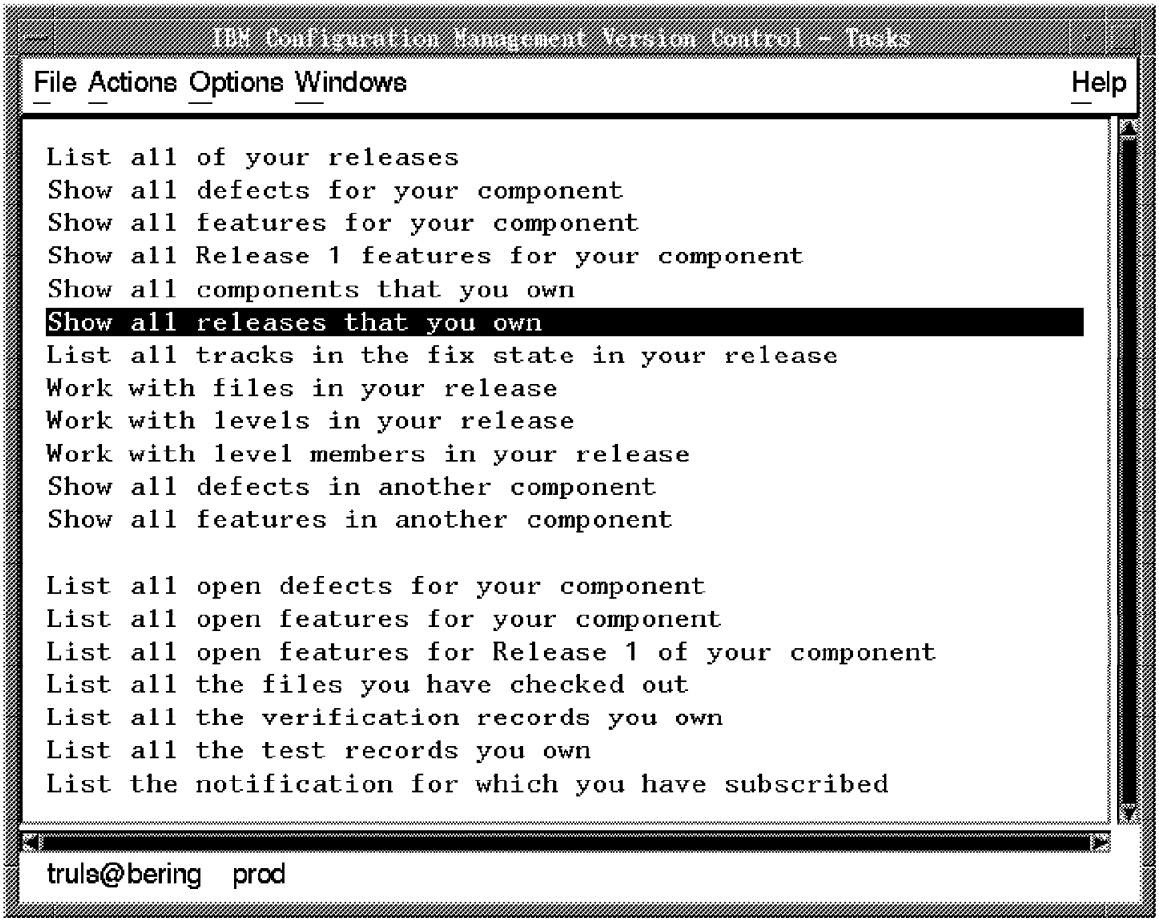

Figure 49. CMVC - Tasks Window As It Is Shipped

Figure 50 on page 107 shows new actions added to the CMVC - Tasks window that correspond to the suggested default queries for a new user on this project. You can put non-CMVC actions into this window. Notice that we have added the action of listing the files and directories in the home directory (**ls**). Some CMVC - Tasks window customizations are described in more detail in Appendix H, “Tailoring CMVC Windows for Different Types of Users” on page 217.

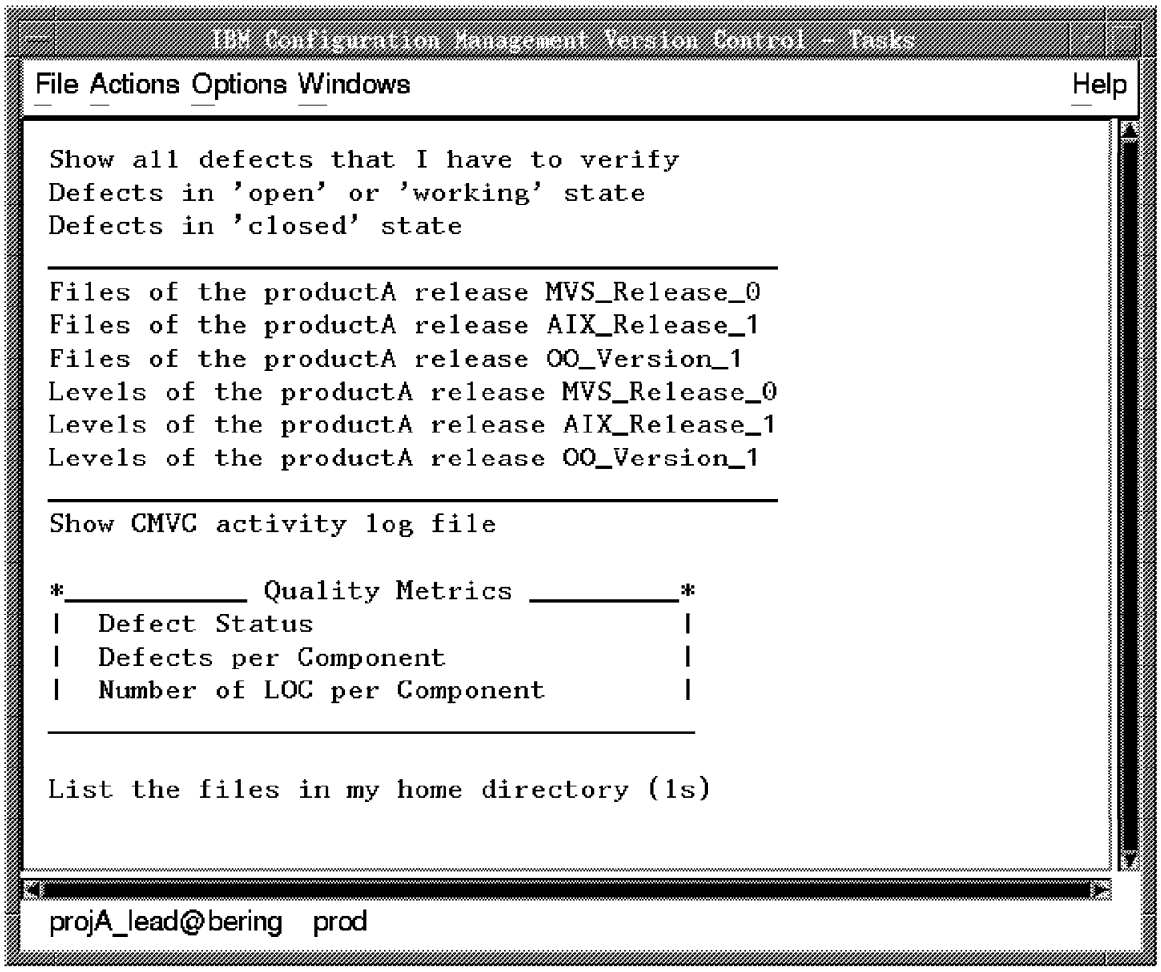

Figure 50. New Tasks Added to the CMVC - Tasks Window

5.1.2.3 CMVC and SDE WorkBench/6000

There are several things to know about to ensure a productive use of CMVC from tools integrated with SDE WorkBench/6000. Unfortunately, they are not documented in one place but are spread out over several chapters in several CMVC manuals. To save you time struggling with cross references, we recommend that you check for chapters relating to SDE WorkBench/6000 in the following documents:

- *IBM CMVC Client Installation and Configuration*
- *IBM CMVC User's Reference*
- *IBM CMVC User's Guide*.

Refer to Appendix E, "CMVC and SDE WorkBench/6000" on page 207. for details on how to use the integrated CMVC client GUI from various tools integrated with SDE WorkBench/6000.

5.1.3 Becoming Familiar with the CMVC Command-Line Interface

Before trying to accomplish anything in CMVC, we recommend that if you are a new user, you spend some time becoming familiar with the command-line interface described in the *IBM CMVC Commands Reference*. The sections that follow describe our experience in using the command-line interface.

5.1.3.1 CMVC Commands

For each CMVC object, except for the family, CMVC provides one command, for example: **User**, **Component**, **File**, or **Host**. Each CMVC action and each object attribute corresponds to a command flag. For example, if you want to create the file *ibmbsel.cob* managed by the component *COBOL*, and used by the *MVS_Release_0* release, you should issue the command shown in Figure 51.

```
File -create ibmbsel.cob -component COBOL -release MVS_Release_0 \
-description "report production"
```

Figure 51. CMVC Command Example

When you need to know the exact command syntax enter only the command name without any flags or parameters. Figure 52 shows the result when the **User** command was issued without flags or parameters.

```
Usage: User -create -login Name -address Name -family Name
           [-name Text] [-area Name] [+super] [-become Name] [-verbose]

       User -configInfo -family Name [-become Name] [-raw]

       User -delete Name ... -family Name [-become Name] [-verbose]

       User -modify Name ... -family Name { -login Name -name Text
           -address Name -area Name [+super | -super] } [-become Name]
           [-verbose]

       User -recreate Name ... -family Name [-become Name] [-verbose]

       User -view Name ... -family Name [-long] [-become Name] [-verbose]
```

Figure 52. CMVC Command Online Help Example

You do not need to enter the full flag word; however, you must enter enough of the beginning of the word so that the command can discern your flag from other legitimate flags. For example, the command shown in Figure 51 could be simplified as shown in Figure 53.

```
File -cr ibmbsel.cob -co COBOL -rele MVS_Release_0 \
-des "report production"
```

Figure 53. CMVC Simplified Command Example

5.1.3.2 CMVC Client Environment Variables

When using the command-line interface, you can omit certain parameters if you have set up the corresponding CMVC environment variables. Table 10 shows the relationships among the environment variables and the command flags.

Table 10. CMVC Client Environment Variables and Command Flags

Parameter	Variable	Command Flag
Family name	*CMVC_FAMILY*	**-family**
CMVC user ID	*CMVC_BECOME*	**-become**
The name of the component managing data on which you work most of time	*CMVC_COMPONENT*	**-component**
The name of the release on which you work	*CMVC_RELEASE*	**-release**
The directory from which CMVC can search the files	*CMVC_TOP*	**-top**

5.2 Project Manager Asks about Setting up the SCM Environment

Let us now return to our project as it was described in 2.2.2, “Project Manager Asks about Setting up the SCM Environment” on page 15 where the system administrator created an SCM environment with CMVC. The actions performed by the system administrator are described in the sections that follow.

5.2.1 Creating a CMVC Family

The system administrator creates CMVC families and their related databases using the **mkfamily** and **mkdb** CMVC server commands, which are documented in *IBM CMVC Server Administration and Installation*. These commands are CMVC server commands; they must be executed on the CMVC server’s host. They cannot be executed through the GUI. You must already have the RDBMS, NetLS, and CMVC Server installed before you can create CMVC families. Refer to Chapter 6, “Installing CMVC and Supporting Databases” on page 161 for additional information about installing Oracle, NetLS, and initializing CMVC.

5.2.2 Modifying Choices Lists, Authority, Interest, and Processes

Before running the **mkdb** CMVC server command, you can modify the choices lists, define new authority or notification groups, and define new component or release processes that are different from the defaults shipped with CMVC. The specific mechanism for doing so is described in detail in *IBM CMVC Server Administration and Installation*. If you do not make any modifications during initial configuration but need to later, you should use the **chcfg**, **chcomproc**, or **chrelproc** CMVC server commands. As specified in the manual, you may need to stop and restart the CMVC server daemons to perform these changes.

In our project, the company’s development methodology dictated the use of CMVC default authority and notification groups, as well as the default component and release processes. It also required modifying the choices list for reasons to accept a defect.

To do this, we edited the `config.ld` file to add a value to the *answerAccept* configuration type. The file changes consisted of adding the value

"new_baseline" and the corresponding descriptive text: "This defect/feature initializes a new baseline for a release." After that, we executed the **chcfg** command. We then copied the customized `config.ld` to our company's development methodology directory, `/usr/lib/CMVC`, to be used as a template.

5.3 Project Manager Asks about Implementing Quality Metrics and Project Practices

Sections 2.2.3, "Project Manager Asks about Implementing Quality Metrics" on page 19 and 2.2.4, "Project Manager Asks about Implementing Project Practices" on page 22 show how CMVC was configured and customized during our project to implement company policies and project practices. The CMVC server commands used by the SCM administrator on our project are described in the sections that follow.

5.3.1 Configuring Fields

You can customize your family by implementing configurable fields for the Users, Files, Defect, or Feature records. In section 2.2.3, "Project Manager Asks about Implementing Quality Metrics" on page 19, the QA representative asked about implementing quality metrics computed from the number of lines of code (LOC) in your project. The project manager decided to configure the File record to have an *LOC* field. It is best to make this modification before the family is created (before running **mkdb**) by importing the configurable field from an existing family. To do this, you copy the files found in the `configField` subdirectory of the home directory belonging to an existing family to the same subdirectory in the home directory of your new family and then run the **mkdb** CMVC server command without using the **-d** flag. You can also change it later using the **chfield -object File** or **chfield -object File -source** CMVC server commands as documented in *IBM CMVC Server Administration and Installation*.

5.3.2 Configuring User Exits

If you are setting up a family, it is a good time to configure any user exits. You do not need to do this before the family and database are created, but it is better that you do it before your users access CMVC.

In section 2.2.3, "Project Manager Asks about Implementing Quality Metrics" on page 19, the SCM administrator added a new field to the File record to contain the *LOC* count and created a user exit program that computes the LOC count and fills in the field. The SCM administrator then ensured that this user exit program would be called each time a CMVC file is created or checked in to CMVC.

In section 2.2.4, "Project Manager Asks about Implementing Project Practices" on page 22, the project manager decided that the defect and feature numbers should be automatically generated and inserted in newly created defect and Feature records. The project manager also decided that a specific module header should be added to each new C source file when the CMVC file was created. The SCM administrator created two user exit programs to do this. Appendix C, "User Exit Samples and Suggestions" on page 183 shows the codes for both programs.

5.4 Project Manager Asks about Starting Up CMVC Client

In section 2.2.5, “Project Manager Asks about Starting Up CMVC Client” on page 23, the project manager started the CMVC client GUI for the first time and defined for this family the CMVC user IDs that each of the team members would use.

5.4.1 Starting CMVC Client GUI

To use the CMVC client GUI, you need to know:

- Family name: *prod*
- Server host name: *bering* (optional)
- TCP/IP port number: *1222* (optional)
- Your CMVC user ID.

You can ignore the server host name, if the family name aliases the host server name in the /etc/hosts file on your host. Likewise, the TCP/IP port number is not necessary input, if it has been added to the /etc/services file on your host.

The first time you start the CMVC client GUI the Set Family dialog box is displayed. You should fill in the Family and User ID fields. CMVC will fill in the User ID field with your UNIX login name by default, but you can override this entry. When you select **OK** this dialog box closes. You can change your entry in both Family and User ID fields by selecting **Set Family...** or **Set User ID...** from the Options pull-down of any CMVC window at any time. Figure 54 shows how to fill in fields of the Set Family dialog box if neither the /etc/services file nor the /etc/hosts file on your host has been updated for your CMVC family.

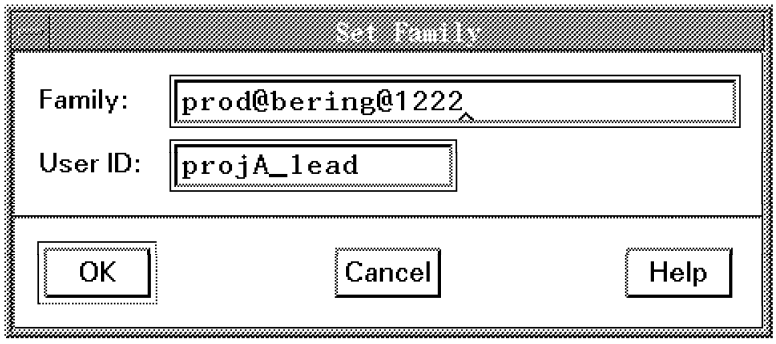

Figure 54. Setting Family Name and User ID When /etc/hosts and /etc/services Files Have Not Been Updated for the Family

Figure 55 on page 112 shows how to fill in the Set Family dialog box fields if both the /etc/services file and /etc/hosts file have been updated for your CMVC family.

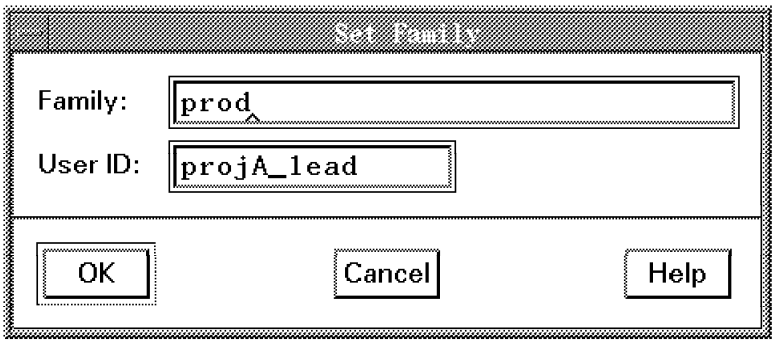

Figure 55. How to Set Family Name and User ID from CMVC Client GUI When /etc/hosts and /etc/services Files Have Been Updated

5.4.2 Creating CMVC User IDs and Host Lists

The CMVC family superuser ID, as mentioned in section 5.2.1, “Creating a CMVC Family” on page 109, is created automatically while running the **mkdb** CMVC server command. This command also adds a host list entry for the superuser and sets values for the *CLIENT_HOSTNAME* and *CLIENT_LOGIN* environment variables in the `.profile` file for the family’s UNIX login name. All other CMVC user IDs must be created subsequently. For our project, the CMVC superuser ID was *lrconas* and it was associated with the *lrconas* AIX login name and the *bering* host.

Only the CMVC family superuser can create or delete other CMVC user IDs, by default. However, the family superuser can grant superuser authority to another CMVC user ID. In our project the SCM administrator gave the project manager, *projA_lead*, superuser authority for this purpose.

To create new CMVC user IDs, display the CMVC - Users window. Select **Create...** from the Actions pull-down and fill in the User ID (CMVC user ID), the User’s mail address, the User’s full name, and the User’s area. You can also give a new user superuser privileges for your family, by selecting the Grant superuser privilege push button. Then, select **OK** if you are creating only one CMVC user ID, or **Apply** if you are creating more than one CMVC user ID and want this window to remain displayed. Figure 56 on page 113 shows the Create User dialog box and the CMVC - Users window.

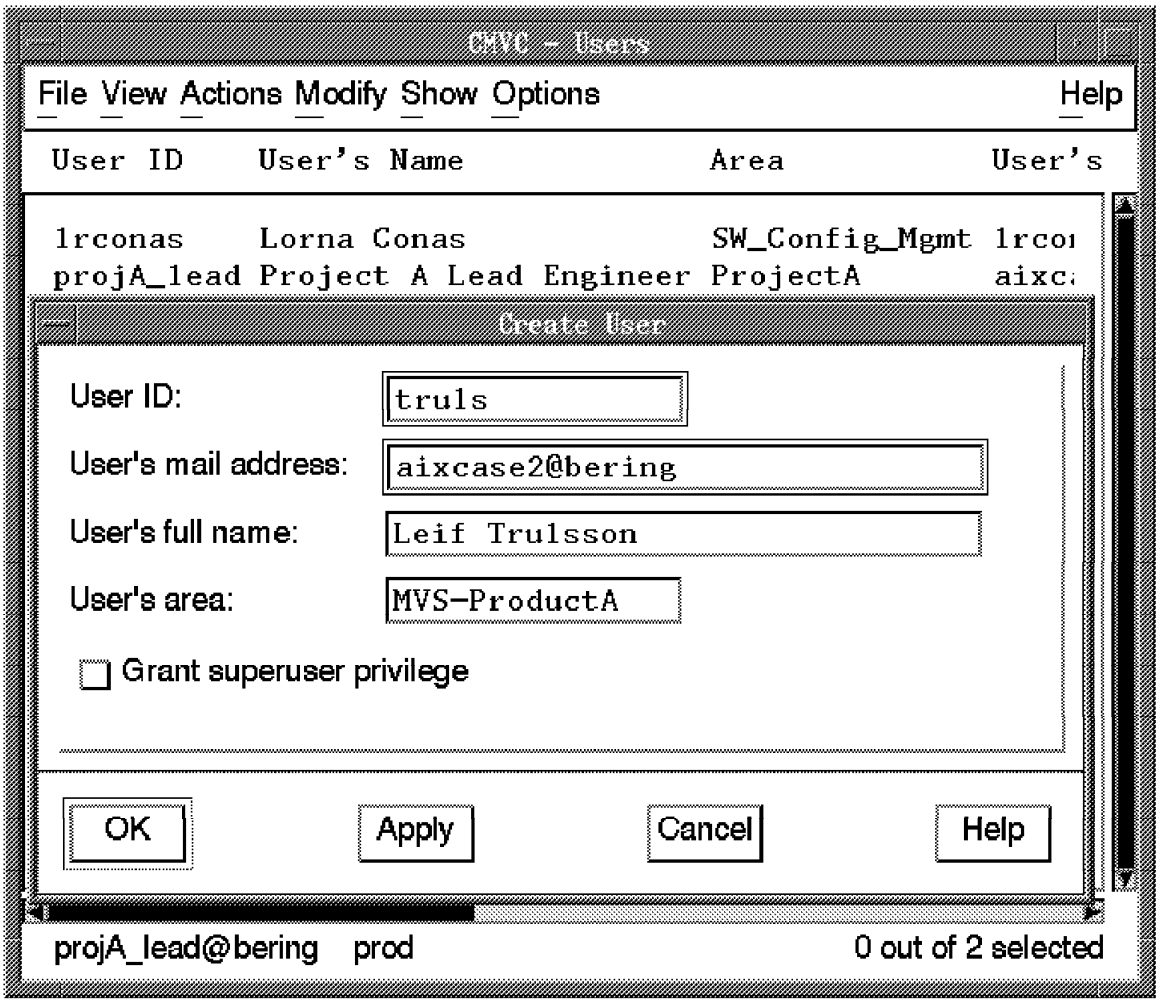

Figure 56. Creating a CMVC User with the CMVC Client GUI

You can also use the **User** command to create new CMVC user IDs. Figure 57 shows the CMVC command to create the same CMVC user created in Figure 56. This command requires that you already have set up the CMVC_FAMILY and CMVC_BECOME environment variables to *prod* and *projA_lead*, respectively.

```
User -create -login truls -address aixcase2@bering \
-name "Leif Trulsson" -area MVS-ProductA
```

Figure 57. Creating a CMVC User with the User Command

Immediately after creating the CMVC user IDs, create a Host List entry for each. If CMVC cannot find a valid Host List entry, the CMVC user ID is not allowed to access the CMVC family. The family superuser must create a CMVC user ID's first Host List entry. A CMVC user ID can create or delete additional Host List entries for itself only. The family superuser can create or delete any Host List entry for any CMVC user ID.

You can create Host List entries in two different ways with the CMVC client GUI:

- Highlight a User ID in the CMVC - Users window and then select **Add Host...** from the Actions pull-down
  or

- Select **Add...** from the Actions pull-down on the CMVC - Hosts window.

Figure 58 shows the Add Host dialog box and the CMVC - Users window.

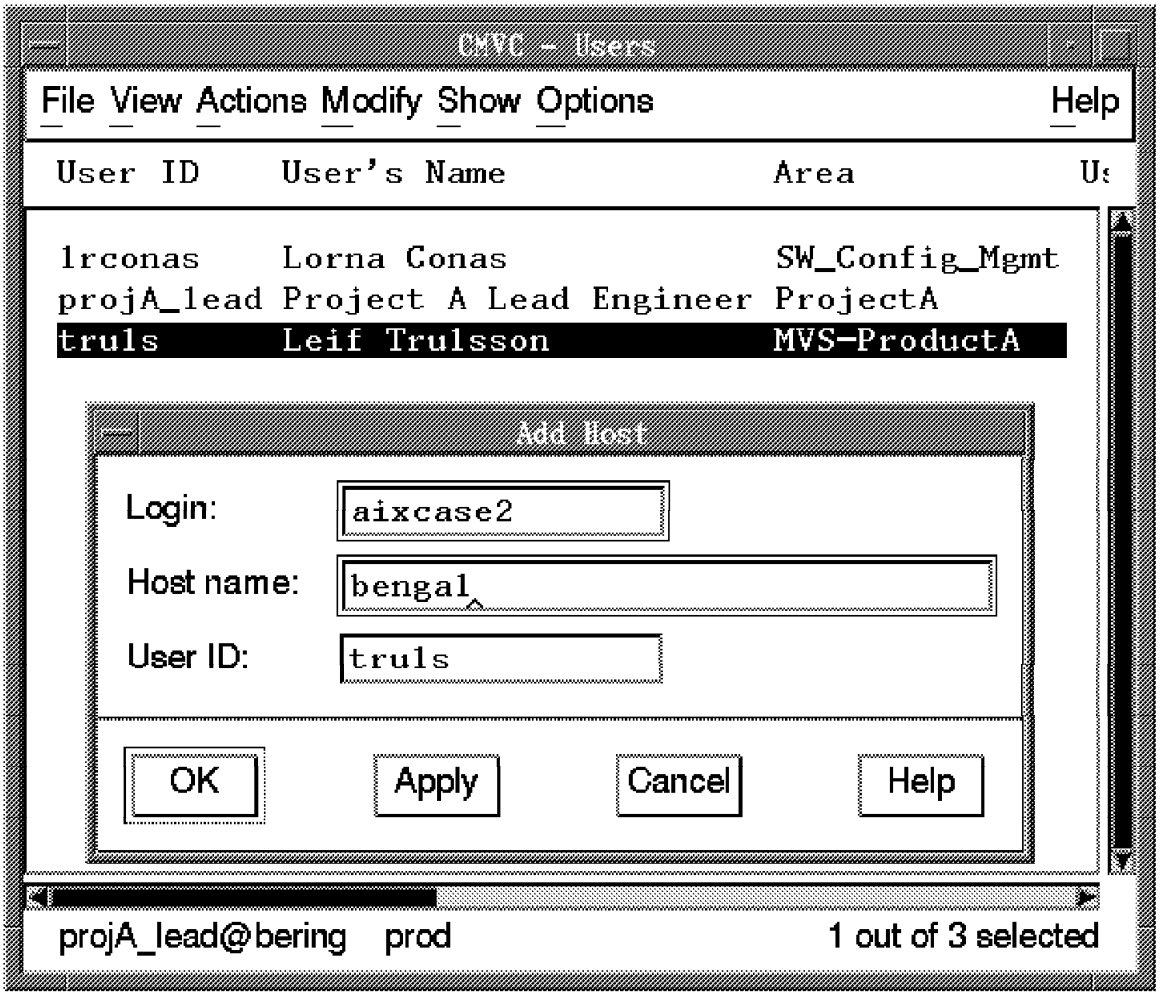

Figure 58. Creating a Host List Entry with the CMVC Client GUI

You can also use the CMVC **Host** command to create a Host List entry as shown in Figure 59. The Login and Host fields shown in Figure 58 are concatenated to a single attribute in the **Host** command. That is, the **-create** attribute shown in Figure 59 takes a compound value composed of the host name separated from the login name by the "@" character. The User ID field from the Add Host dialog box corresponds to the **-login** attribute.

```
Host -create aixcase2@bengal -login truls
```

Figure 59. Adding a Host List Entry with the Host Command

You can create several Host List entries for each of your CMVC user IDs, because the users might be expected to execute the CMVC client GUI from a number of hosts and login name combinations.

5.5 Project Manager Asks about Project Organization in CMVC

In section 2.2.6, “Project Manager Asks about Project Organization in CMVC” on page 24, the project manager learned how to organize CMVC family data to support the project’s requirements using releases, components and the component hierarchy. The following sections that follow describe the CMVC actions involved, which included:

1. Building a CMVC component hierarchy
2. Creating CMVC releases
3. Entering access list and notification list entries for certain components
4. Opening a defect to authorize bringing the existing application source code files under CMVC control and creating a release to represent that existing application baseline on MVS.

5.5.1 Creating and Manipulating Components

After creating CMVC user IDs and Host List entries, you should create your component hierarchy. The easiest way to work with components is to display the CMVC - Component Tree window.

You display this window by selecting **Component** from the Windows menu. The Windows menu can be popped up by pressing the right mouse button while the graphical cursor is in any CMVC window. The Windows pull-down on the CMVC - Tasks window also produces this menu. Further, if you are using SDE WorkBench/6000, you can display this menu using the Windows pull-down in the WorkBench - Development Manager window.

The CMVC - Component Tree window shows you your component hierarchy in a graphical manner, illustrating the hierarchical relationships in either a horizontal or vertical layout. This window is useful for browsing the component hierarchy and for manipulating components and their relationships. Using the graphical cursor, the left mouse button, and the pull-down menus, you can create, delete, unparent, and reparent components to manipulate the component hierarchy.

You can also modify how much of the component hierarchy is displayed in the CMVC - Component Tree window by pressing the right mouse button. This mouse button behaves differently depending on object the graphical cursor is “pointing at,” or appears to be hovering over, when the button is pressed. When the cursor is over a figure representing a component (a “node” in the hierarchy), the right mouse button causes the Node menu to be displayed. The Node menu allows you to expand or suppress various levels of the hierarchy. A new user may want to have all levels of the hierarchy visible, but an experienced user may be more comfortable hiding subnodes. When the cursor is not over a component, but in the background space, the Popup Menu is displayed by pressing the right mouse button. The Popup Menu gives convenient access to several common CMVC actions and enables you to display the Windows pop-up menu.

Figure 60 on page 116 illustrates the horizontal layout of the CMVC - Component Tree window and the Popup Menu. The CMVC - Component Tree window is showing part of the component hierarchy for the *production* family.

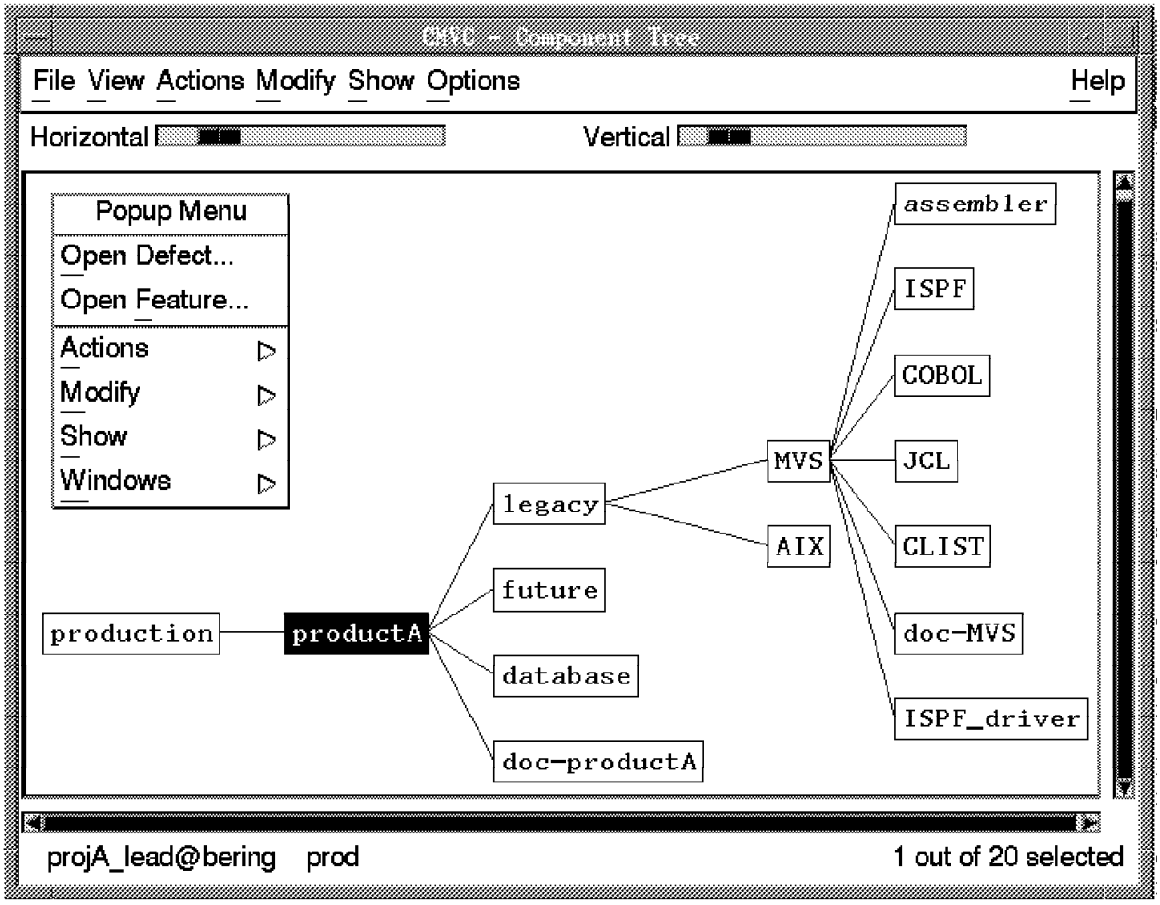

Figure 60. CMVC - Component Tree Window Horizontal Layout with Popup Menu

To change the layout from vertical to horizontal, or vice versa, select **Layout** from the View pull-down on the CMVC - Component Tree window. Figure 61 on page 117 illustrates the CMVC - Component Tree window's vertical layout and the Node pop-up menu. This figure shows the same portion of the *production* family component hierarchy as is shown in Figure 60.

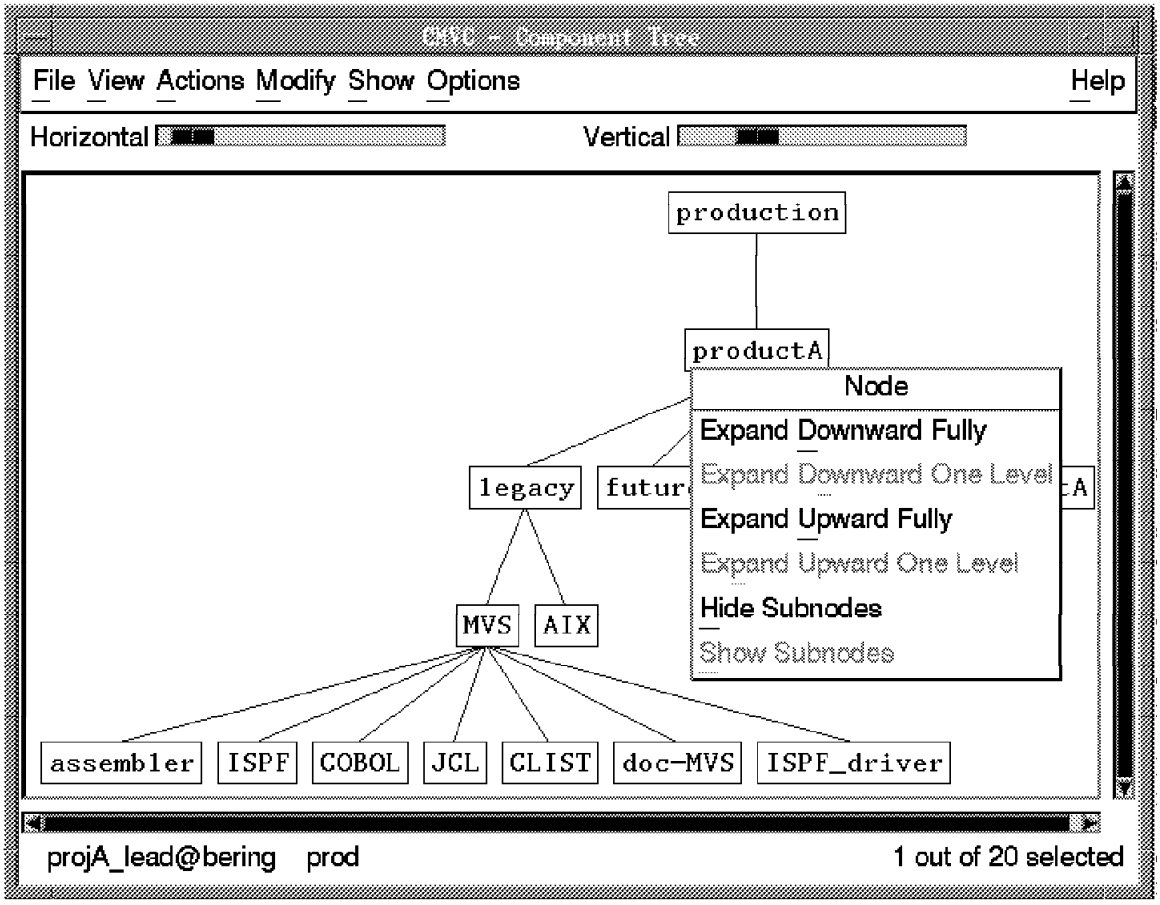

Figure 61. CMVC - Component Tree Window Vertical Layout with Node Menu

From the CMVC - Component Tree window you can display the CMVC - Components window, which lists the full record for all component records. You can also display this window by selecting **List...** from the View pull-down menu of the CMVC - Component Tree window. This window uses a table format and shows many details that are not shown in the CMVC - Component Tree window. This window works like other CMVC windows because you issue queries to determine which records are displayed. One or more components highlighted in this window can later be “imported” to fill the Component field in other CMVC window dialog boxes using the **Import** button. This window is like the CMVC - Component Tree window in that you can create and delete components and manipulate component relationships with it. However, it is a difficult mechanism for showing the hierarchical relationships among the components.

You should fully plan your component names because every component created and later deleted is remembered by CMVC forever, and its name becomes unusable for a later component. However, do not panic if you change your mind, or make an error. Components can be deleted, if they currently manage no files, and all of their properties can be changed, including their names.

You can create all of the initial components, assigning component ownership to specific CMVC user IDs. The owners of these components can later create additional lower-level (child) components as needed. Note that you do not need a defect or feature to justify the creation or deletion of a component.

When the CMVC - Component Tree window is first invoked, it shows you a topmost-level component for the family, named *root*. This was created for you by CMVC when your family was created. As a first step, you can rename this component to match your family name. To rename this component, highlight the *root* component with a single click of the left mouse button and select **Name...** from the Modify pull-down. This displays a dialog box. Fill in the new name and select **OK**. In our project we changed the *root* component name to “production.”

Because the default query shipped with CMVC for the CMVC - Component Tree window looks for a component whose name is “root,” you must also change the default query for this window. To do so, we selected **Initial Component...** from the File pull-down and entered `production` in the resulting dialog box.

With your root component highlighted, create new components parented by it. Select **Create...** from the Actions pull-down. The new component is displayed descending from (parented by) the *production* component. Continue creating your component hierarchy using the **Delete...**, **Recreate...**, **Link**, **Unlink**, and **Reparent** selections from the Actions pull-down.

If you find you must move a component so it descends from a different component (**Reparent**), or link it to another component so it has multiple parent components (**Link**), you highlight the child component and select the relevant menu item. A dotted line then extends from the currently highlighted component to the cursor. Place the cursor over the target component and press the left mouse button to complete the move or link. While it is not important in which order the sibling components appear on this diagram, you will notice that they are displayed in the same order in which they are created, and they cannot be rearranged afterward.

When creating a component, a dialog box is displayed. You must enter the component name, the component process from a list of predefined choices, the owner, and a brief description of the component. If you are going to assign ownership at this time, you should first display the CMVC - Users window and highlight the correct CMVC user ID. This enables you to use **Import** to bring this user name in without having to type it. **Import** is useful in this particular dialog box, if you intend to create several components all owned by the same user.

You can find **Import** on many dialog boxes. It appears next to one or more input fields representing CMVC objects. Using **Import** saves time and precludes typing errors. Learning how to make use of **Import** in the various windows takes some time, but it is useful once you have mastered it. We will attempt to guide you in selecting windows in an order that makes the use of **Import** practical. However, at times it may seem like less trouble to type in the value rather than create the conditions to use **Import**. Figure 62 on page 119 shows the screen captured while creating a component.

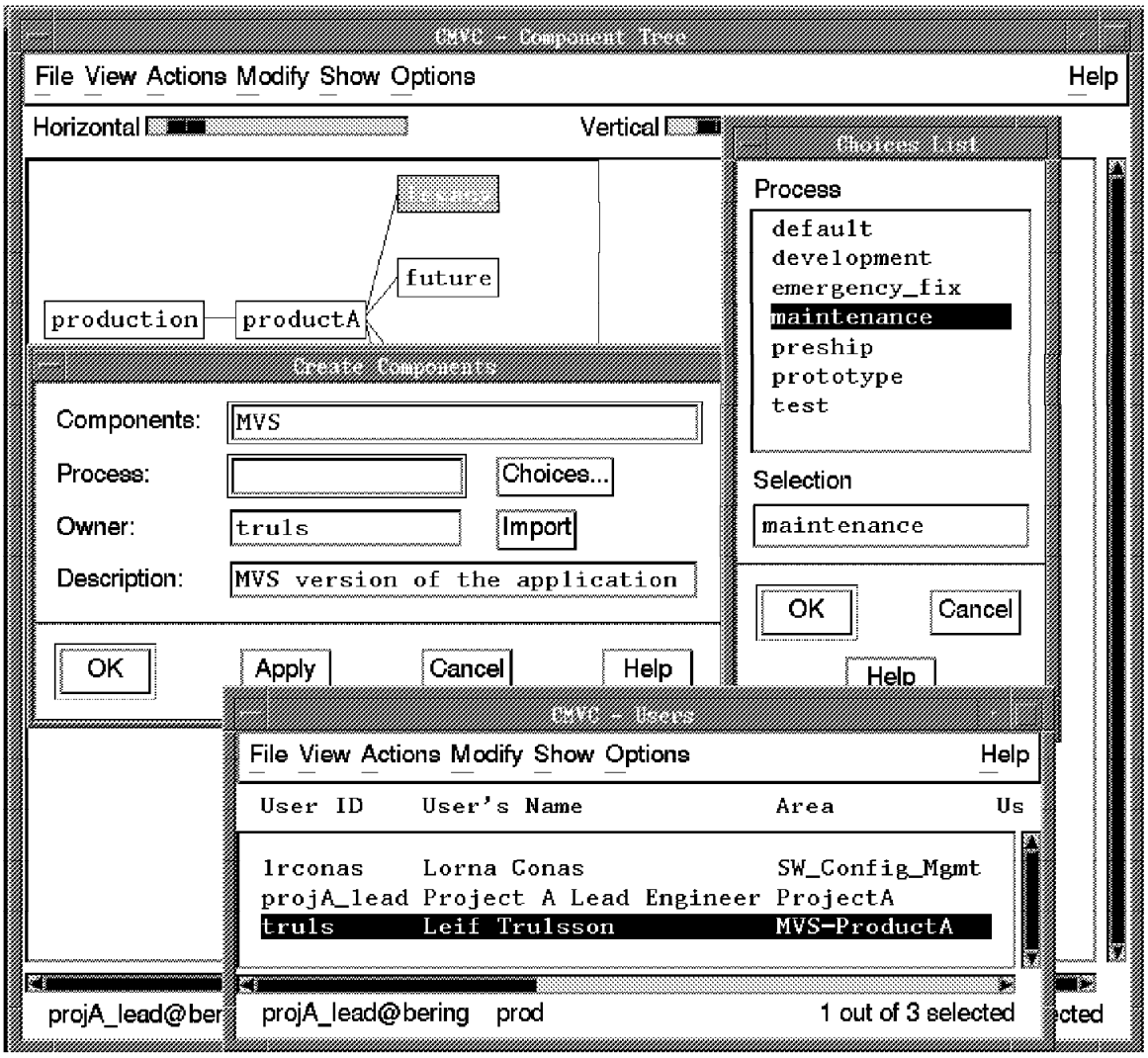

Figure 62. Creating a Component from the CMVC - Component Tree Window

You can also use the command-line interface to create a set of components. Figure 63 shows the CMVC **Component** command.

```
Component -create MVS -parent legacy -owner truls \
-process maintenance \
-description "MVS version of the application"
```

Figure 63. Creating a Component with the Component Command

5.5.2 Creating a Release with the Test Environment

Using a CMVC user ID with superuser privilege, create the releases that are managed through the components. When you create a release, you assign its ownership to the CMVC user ID that is responsible for managing the release.

To create the release, display the CMVC - Users window and initiate a query to find the record for the user who will own the new release. Highlight that CMVC user ID and display the CMVC - Component List window. Issue a query to display the component that should manage this release. Highlight this

component and then display the CMVC - Releases window. Select **Create...** from the Actions pull-down of this new window. Use **Import** to fill in the Component and Owner fields. Select the release process you want from the list displayed when you select **Choices...**. Now, enter the release name, description, and other fields, if appropriate. The Approver, Environment, and Tester fields may not be necessary depending on the process you are selecting for the release. Since we chose the *maintenance* process, we had to enter both Environment and Tester fields. Select **OK** to create the release. The release owner is automatically notified by mail that a release has been created.

If you do not know which subprocesses are included in a particular release process that is listed in the Choice List window, select **On Process...** from the Help pull-down. Figure 64 shows the various dialog boxes to create a release, as well as the CMVC - Help window, which shows the subprocesses included in the *maintenance* release process.

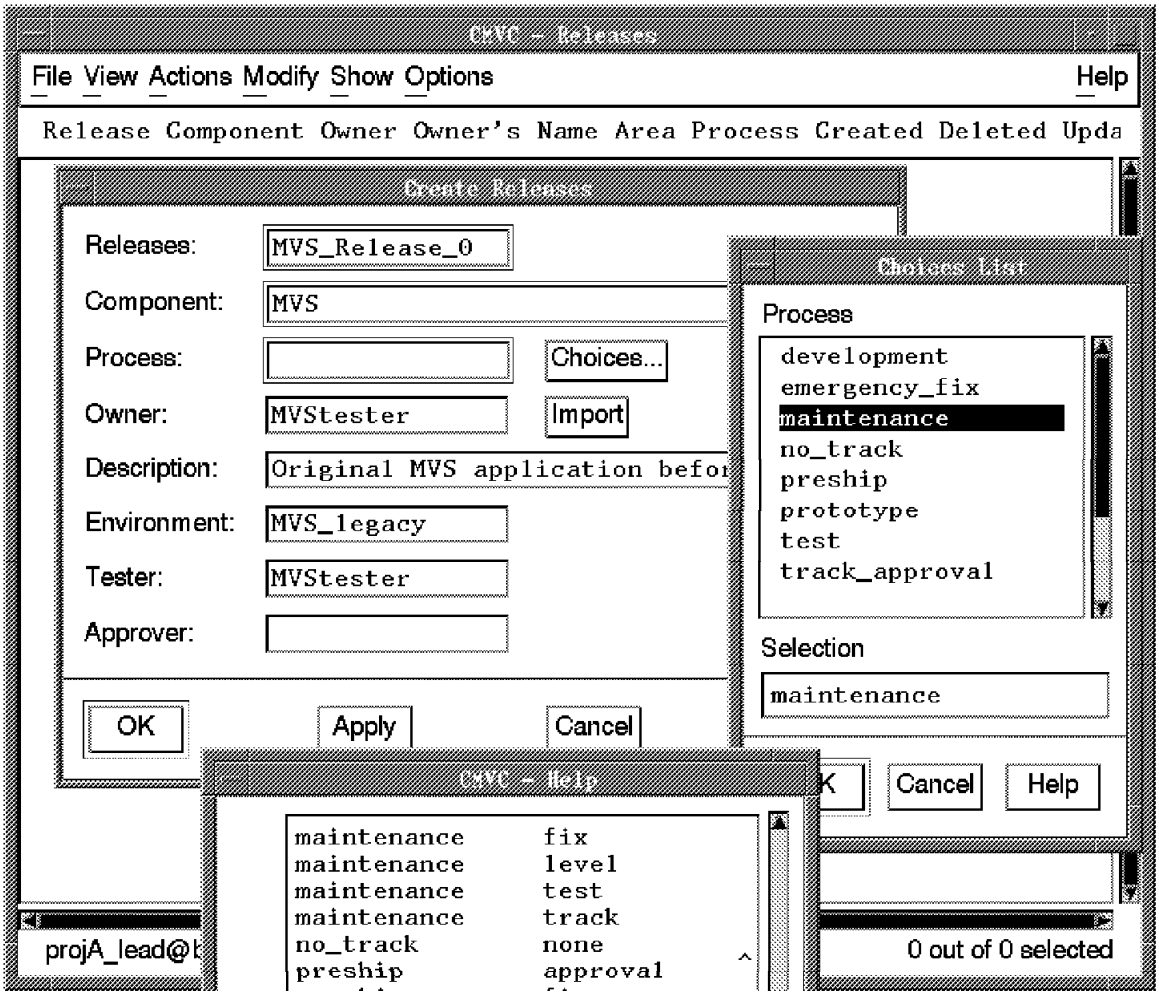

Figure 64. Creating a Release from the CMVC Client GUI

You can also use the CMVC command-line interface to create a release. Figure 65 on page 121 shows the CMVC **Release** command.

```
Release -create MVS_Release_0 -owner MVStester \
-process maintenance -component MVS \
-description "Original MVS \
application before migration effort began"
```

Figure 65. Creating a Release with the Release Command

5.5.3 Setting Access and Notification Lists

Once you have organized a framework to control your data (according to your requirements) through components and releases, you want to control access to this data based on CMVC user ID and have various people informed automatically about actions performed during the development effort. To do this, you create an access list and a notification list for certain components.

A component owner can implicitly perform many actions on objects, such as files, that are managed by that component. For example, a component owner has implicit authority to check in, check out, or rename a file. The *IBM CMVC User's Reference* lists how implicit authority to perform CMVC actions is derived.

Authority to perform other CMVC actions can be granted explicitly by associating a CMVC user ID with an authority group for a particular component in that component's access list.

To place a user in an access list for a component, display the CMVC - Users window and issue a query to display the users you intend to put in the list. Now display the CMVC - Component Tree window and highlight the component whose access list you want to modify. Next, select **Access List...** from the Action pull-down and wait for the Add Access dialog box to be displayed. Enter the fields in the dialog box by selecting **Import** to bring in the preselected CMVC user IDs. Now, select **Choices...** to display the list of the valid authority groups. Select the desired group(s) and then select **OK** in the Choices List dialog box. To create the entry in the access list, select **OK** in the Add Access dialog box. There are other paths to this same dialog box; using the CMVC - Component Tree window offers just one.

Figure 66 on page 122 shows the CMVC client GUI windows and dialog boxes displayed when creating both an access list entry and a notification list entry.

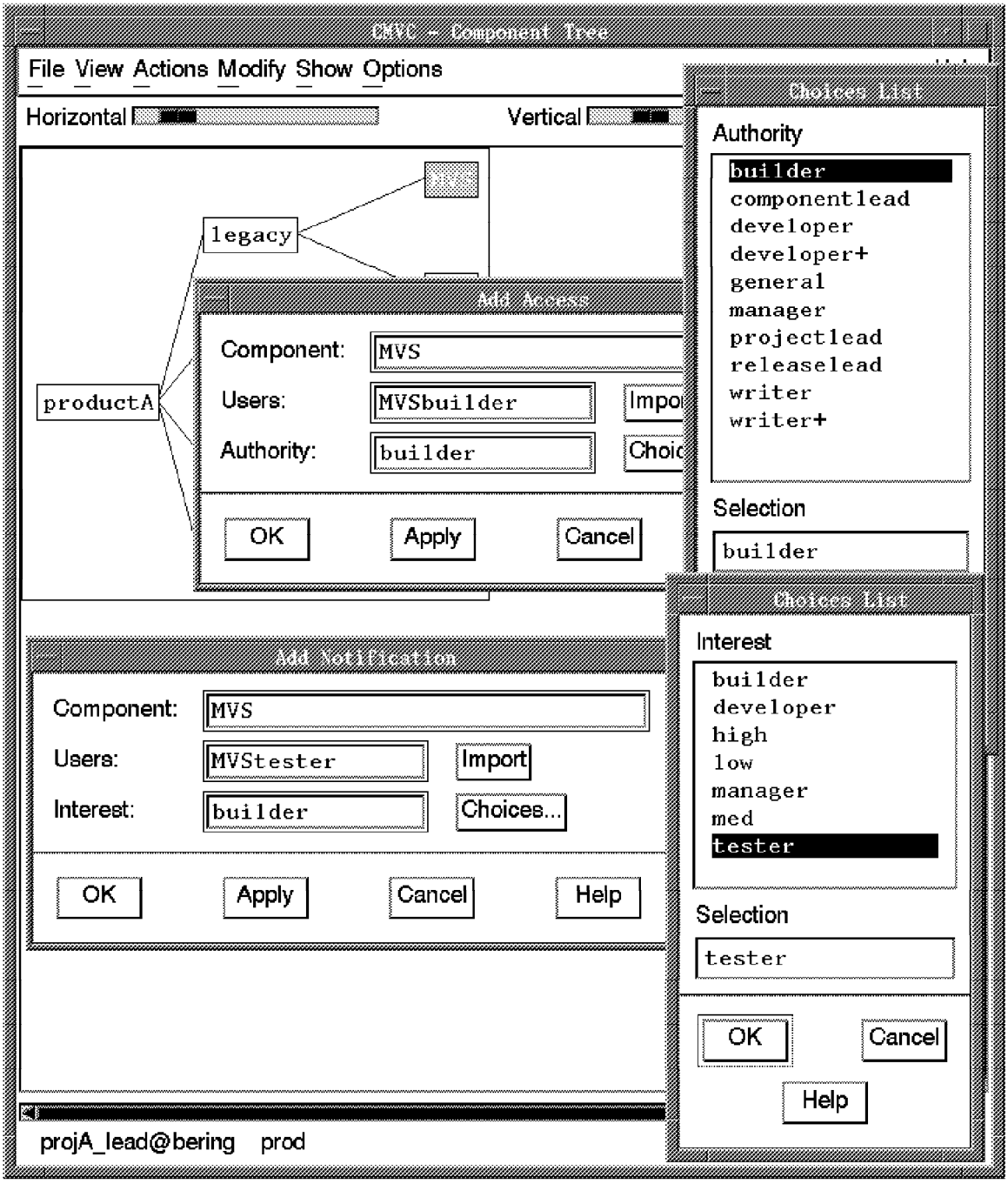

Figure 66. Adding Access List and Notification List Entries through the CMVC Client GUI

You can also use the CMVC command-line interface to add entries to the access list. Figure 67 shows the CMVC **Access** command.

```
Access -create -component MVS -login MVSbuilder -authority builder
```

Figure 67. Adding an Access List Entry with the CMVC Access Command

The authority group defines a set of actions to be authorized. You refer to the group name, instead of naming each authorized action individually. You can

refer to the combination of actions defined by the authority group *releaselead* as *releaselead* authority.

You can retrieve the list of actions associated with a given authority group through the CMVC client GUI. To list the actions of an authority group, select **Authority Actions** from the Show pull-down in the CMVC - Access Lists window and then enter the authority group name. You can also use the command-line interface to interact with Action Lists. Figure 68 shows the CMVC **Report** command.

```
Report -view authority -where "name='builder'"
```

Figure 68. Using CMVC Report Command to List Actions for an Authority Group

The authority of a user that is explicitly granted at a component is inherited downward recursively through the component hierarchy. At descendent components, additional authority can be granted to the same user. For example, on our project we explicitly granted the *krt* CMVC user ID the *general* authority for the component *productA*. This user ID then inherited that authority over its descendent components, including the *MVS* component. The *krt* CMVC user ID was also explicitly granted the *builder* authority for the component *MVS*. Therefore, this user had all the authority defined for both authority groups at the component *MVS*.

You must study the implications of granted authority and its inheritance so you can minimize the effort you spend manipulating access lists. For example, you might only have to grant authority to check in and check out files to a given user at one high level component, if all the files that user needs to access are managed by descendent from that one component. Once granted at the high component, the authority is inherited by that user over all of the descendent components.

You probably want to grant *general* authority to all project team members at a fairly high component, so they can view defects, features, files, and other CMVC objects in your family. The *developer* authority group grants authority to perform all of the CMVC actions you would want a developer to do. Therefore, you may also want to grant developer authority to all your programmers at a fairly high component, so they can work on a wide variety of files in your application.

You can restrict the explicitly granted authority of a user to any given component where the user inherits this authority from an ancestor component. This restriction of authority prevails only at the one component; the inheritance of the granted authority prevails over descendent components. Restriction is also handled by means of authority groups, only now they represent sets of actions *not* authorized for the user on objects managed by this component.

You should also study the subtle interactions that occur when inherited authority interacts with explicitly restricted authority. Take the example of a user who implicitly inherits only *general* and *developer* authority for a component in your hierarchy. There is no other explicitly granted authority at this component for this user. If the component owner decides to restrict that user ID's authority to perform *developer* actions, all actions belonging to this one authority group are now restricted for the user ID, unless they are defined in another authority group for which this user ID is authorized at this component. There are some actions, which are authorized by *general* authority and by *developer* group. These

remain enabled for this user ID, but those actions that are defined only for the *developer* authority are now disallowed for this user ID. For example, this user ID can still view a file, but can no longer check out (and therefore, check in) a file managed by this component. Table 11 shows the actions authorized by these two authority groups as they are shipped with CMVC. It also shows which actions are effectively restricted in this case.

Table 11. Inherited Actions Restricted and Remaining

Developer Authority Restricted	**General Authority Inherited**	**Restricted Developer Action**
CompView	CompView	
DefectView	DefectView	
FeatureView	FeatureView	
FileAdd		FileAdd
FileCheckOut		FileCheckOut
FileExtract		FileExtract
FileRename		FileRename
FileView	FileView	
LevelView	LevelView	
NotifyCreate	NotifyCreate	
ReleaseExtract		ReleaseExtract
ReleaseView	ReleaseView	
TrackView	TrackView	
UserView	UserView	

Some authority derives implicitly because of an action taken by a CMVC user, especially if that action resulted in a new CMVC object, such as a defect, file, or release, being created. Authority to perform actions on that object may derive from the ownership of the new object, while authority to create the object and perform other actions on it may have derived from an inherited authority. Do not jump to conclusions about how an authority is derived, and what these authorities imply. Merely owning a release, for example, does not give a user all of the authority associated with the *releaselead* authority group for the component that manages that release. If one user created a release and allowed it to be owned by another user, that second user does not thereby gain sufficient authority to create a second release managed by that component. Likewise, you might be the owner of a defect by virtue of owning the managing component, but that does not give you authority to cancel it. That authority derives from the action of creating the defect.

A notification list works similarly to an authority list, in that they are both associated with a component. Notification is like authority in that a user inherits membership in interest groups at lower level components from higher level components. Likewise, notification of a given CMVC action depends on implicit and explicit factors. An implicit factor might be that a CMVC user is affected by the action. For instance, when a family superuser changes ownership of a release, the new owner of the release will be notified. Notification might also result because a CMVC user ID was explicitly added to the notification list for the component that manages the CMVC object created or affected by the CMVC

action. This is also referred to as “subscribing” to an interest group for a component.

In our example project, the project manager might decide to be informed whenever a new file is brought under control of a particular component. This would require his placing his CMVC user ID on the notification list for that component, referencing the appropriate interest group.

Notification is also like authority, in that new interest groups can be defined. *IBM CMVC User's Reference* lists how implicit notification is determined and which interest group definitions are shipped with CMVC.

Manipulating interest groups and notification lists is similar to manipulating authority groups and access lists. To place a CMVC user ID in a notification list for a component, display the CMVC - Users window and issue a query to display the users you intend to put in the list. Next, display the CMVC - Component Tree window and highlight the component for which you want to modify the notification list. Then, select **Notification List...** from the Action pull-down and wait for the Add Notification dialog box to be displayed. Fill in the fields in the dialog box by selecting **Import** to bring in the preselected user IDs. Select **Choices...** to display the list of the valid interest groups. Highlight the group(s) then select **OK** in the Choices List dialog box. Finally, select **OK** in the Add Interest dialog box.

Figure 66 on page 122 shows the CMVC client GUI windows and dialog boxes when creating both an access list entry and a notification list entry.

To display the set of actions defined by an interest group, select **Interest Actions** from the Show pull-down in the CMVC - Notification Lists window, then enter the Interest Group field.

You can also use the CMVC command-line interface to view the set of actions defined by an interest group. Figure 69 shows the CMVC **Report** command used for this purpose.

```
Report -view interest -where " name='tester'"
```

Figure 69. Using the CMVC Report Command to List all CMVC Actions Associated with Interest Group

And, you can use the CMVC command-line interface to interact with notification lists. Figure 70 shows the **Notify** command.

```
Notify -create -component MVS -login MVStester -interest tester
```

Figure 70. Using the CMVC Notify Command to Add a Notification List Entry

5.5.4 Opening a Defect to Accompany Files in the Initial Release

To authorize bringing the MVS source files under CMVC control with reference to the first release of our product, our project manager opened a defect on the *MVS* component. Typically, a defect is used to report a problem and a feature to make an enhancement. Creating files to initialize a baseline (release), does not

fit neatly into either of these categories. But, our project required that we use one or the other to authorize creating new CMVC files, so we used a defect.

Any CMVC user may open a defect or feature on any CMVC component without special authority. You can open a defect or feature using pull-down menus in the CMVC - Defects window and the CMVC - Components window. You can also open a defect or feature using the CMVC - Component Tree window. To use the CMVC - Component Tree window, highlight the component that will manage the defect or feature and select **Open Defect...** from the Actions pull-down to display the Open Defect dialog box. Enter one or more names in the Release field and the Remarks field. If you have lengthy remarks, you can use the editor by selecting **Edit...**. The Abstract, Level, Environment, and Reference fields are optional. You cannot ignore the Prefix or Severity fields. To choose a prefix or a severity, select **Choices...** next to the field whose list of the valid input values you would like to display. This will cause a Choices List dialog box to be displayed.

Your SCM administrator can set up default values for any fields by changing the family's `config.ld` file and executing the command **chcfg**. Figure 71 on page 127 shows the defect report form used by our project. This form can be customized by your SCM administrator with the command **chfield -object Defect**. Refer to *IBM CMVC Server Administration and Installation* for more information on these two CMVC server commands.

To open the first defect for our project, we displayed the CMVC - Component Tree window and expanded the component hierarchy to display the *MVS* component. We opened the defect against that component. We entered *MVS_Release_0* in the Release field. We did not enter date in most of the optional fields, because these fields are more meaningful for defects dealing with real problems, than for baseline initialization. We selected **d** for the Prefix field and **4** for the Severity field. We did not enter a defect number because we arranged to have the number automatically generated by a user exit program. Refer to 5.3.2, “Configuring User Exits” on page 110 for more about that.

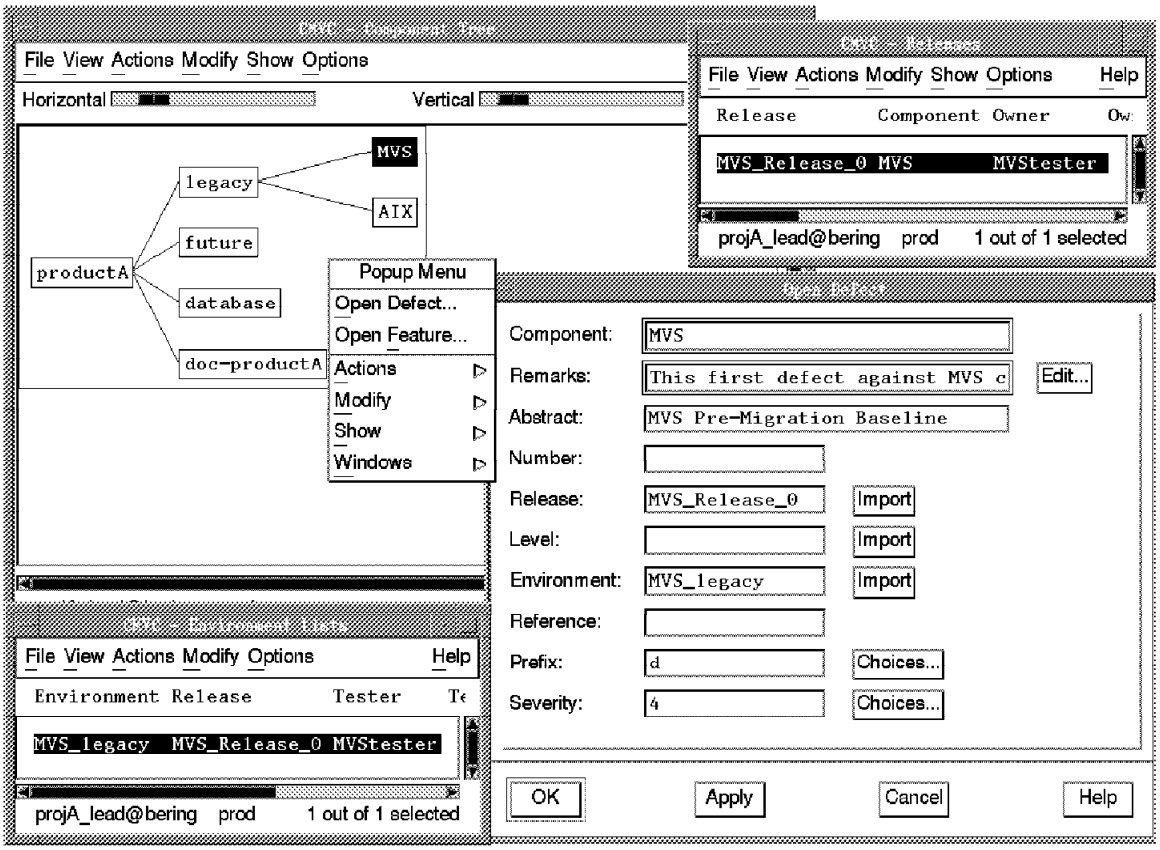

Figure 71. Opening a Defect from the CMVC Client GUI

You can also use the CMVC command-line interface to create a defect. We do not show an example of the CMVC **Defect** command. Having numerous parameters, it is not convenient to use. However, you can use it from batch programs to automatically open a defect under certain circumstances, such as when automated testing of your application fails.

A problem you have with the automated generation of the defect number is that CMVC displays a wrong defect number in the CMVC - Information window at completion of the defect opening operation. For example, our project's first defect was named *prod_00001* by the user exit program, but the CMVC - Information window displayed the following message: "A new defect was opened successfully. The new defect number is 1.."

Figure 72 on page 128 is the first in a series of illustrations with which we hope to show the relationships among certain CMVC actions and the CMVC objects affected or created by them. This figure shows the relationship between the first defect you opened and its managing component. The box representing this defect also indicates the "state" of this CMVC object immediately after the defect creation. States of CMVC objects are like the teeth in the gears of CMVC. As states of objects change, particular CMVC actions become possible or impossible. The states that defects, features, levels, tracks, and releases pass through are described in *IBM CMVC Concepts*.

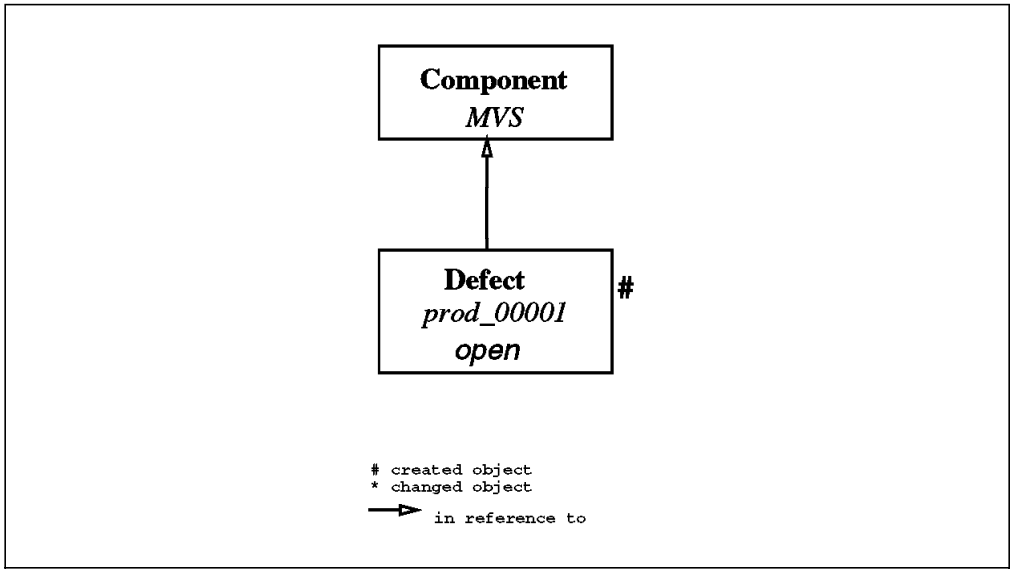

Figure 72. Defect and Component Relationship after a New Defect Is Opened

5.6 Project Manager Asks about Reporting

In section 2.2.7, “Project Manager Asks about Reporting” on page 27, the project manager wanted to generate some reports about our SCM environment. One way to do this with CMVC is to build a query by selecting **Open List...** from the File pull-down from the appropriate CMVC window. Refer to section 5.1.2.1, “CMVC Window Queries” on page 104 for more information on building a query. The contents of the CMVC window can then be printed by selecting **Print...** from the File pull-down.

You can also generate reports with the CMVC command-line interface. The CMVC **Report** command is particularly useful in shell scripts, which can be run on a regular basis or on demand. You can also add execution of these to the task list displayed in the CMVC - Tasks window.

The CMVC client provides sample shell scripts that extract data from CMVC and format them. *IBM CMVC User's Reference* lists and describes those shell scripts located in the `/usr/lpp/cmvc/samples` directory. You can adapt those programs to meet your needs. Figure 73 on page 129 shows a part of the output of the **defectReport** shell script used to generate a global defect report showing tracks, test records, approval records, and fix records.

```
Detailed Defect Report
pre name            compName        state    origin   owner    sev age abstract
--- --------------- --------------- -------- -------- -------- --- --- ------------------------------------------------------
  d prod_00007      AIC_X11R4       working  lrconas  krt      4   157 Renaming COBOLGUI-COBOL integration changes
        < Track: state = integrate  release = AIX_Release_1  >

  d prod_00009      AIX             working  lrconas  krt      4   157 COBOL changes to include COBOL copy code.
        < Track: state = fix  release = AIX_Release_1  >
  d prod_00010      COBOL           working  krt      truls    4   156 Removing original COBOL files
  d prod_00011      MVS             working  truls    truls    4   156 MVS_Release_1 baseline COBOL, CLIST, assembler, ISPF
        < Track: state = fix  release = MVS_Release_1  >

            <      Fix: state = active      component = assembler       release = MVS_Release_1   >
            <      Fix: state = active      component = ISPF            release = MVS_Release_1   >
            <      Fix: state = active      component = COBOL          release = MVS_Release_1   >
            <      Fix: state = active      component = JCL            release = MVS_Release_1   >
            <      Fix: state = active      component = CLIST          release = MVS_Release_1   >

  d prod_00012      COBOL           verify   krt      truls    4   0   File name correction for AIX_Release_1
  d prod_00014      COBOL           working  truls    truls    4   156 Parallel changes related to AIX_Release_1

        < Track: state = fix  release = MVS_Release_1  >
```

Figure 73. Output of the Defect Report Sample Shell Script

5.7 Developer Asks about Components and a Release for the Original MVS Baseline

The MVS developer needed to bring the original MVS source files under CMVC control in 2.2.8, “Developer Asks about Components and a Release for the Original MVS Baseline” on page 30. The MVS developer was owner of the *MVS* component and therefore was able to create child components below it representing the types of source code in the MVS application. This action ensured that the project could track defects against the various types of source code separately. The CMVC actions taken by the MVS developer in creating a mini-hierarchy of components are no different from those described in 5.5.1, “Creating and Manipulating Components” on page 115.

5.8 Developer Asks about Bringing MVS Source Files under CMVC Control

In section 2.2.9, “Developer Asks about Bringing MVS Source Files under CMVC Control” on page 32, the MVS developer created CMVC files to bring the MVS source files under CMVC control, referencing the *prod_00001* defect. To do this, the developer accepted the defect, created a track associated with both the *prod_00001* defect and the *MVS_Release_0* release, and then created the CMVC files. The sections that follow describe these actions.

5.8.1 Accepting the Defect

Most CMVC actions affecting defects and features are taken by the owner of the component managing the defect or feature. Defect or feature ownership can be reassigned to another CMVC user ID by the owner of the component, although the component continues to manage the defect or feature. The defect or feature owner can do two things with a new defect: accept it or return it to the originator. A defect that duplicates another open defect, for example, might be returned. If so, the originator would cancel it.

Our MVS developer accepted the project’s first defect. Because the *MVS* component process was *maintenance*, which does not include the *design-*

size-review subprocess, this action changed the defect's state from *open* to *working*.

To identify any defects or features needing your acceptance, highlight the component in the CMVC - Component Tree window, move the graphical cursor so that it is no longer over a component figure, and press the right mouse button to display the Popup Menu. Select **Show**, which displays a cascading menu. Select **Defects** from that menu to display another cascading menu. Select **Open...** from this last menu and release the right mouse button. The CMVC - Defects window is displayed showing all of the defects in the *open* state.

When our MVS developer followed these steps, there was only one open defect, *prod_00001*. Figure 74 on page 131 shows the Popup Menu and the cascading menus necessary to query the open defects of the *MVS* component.

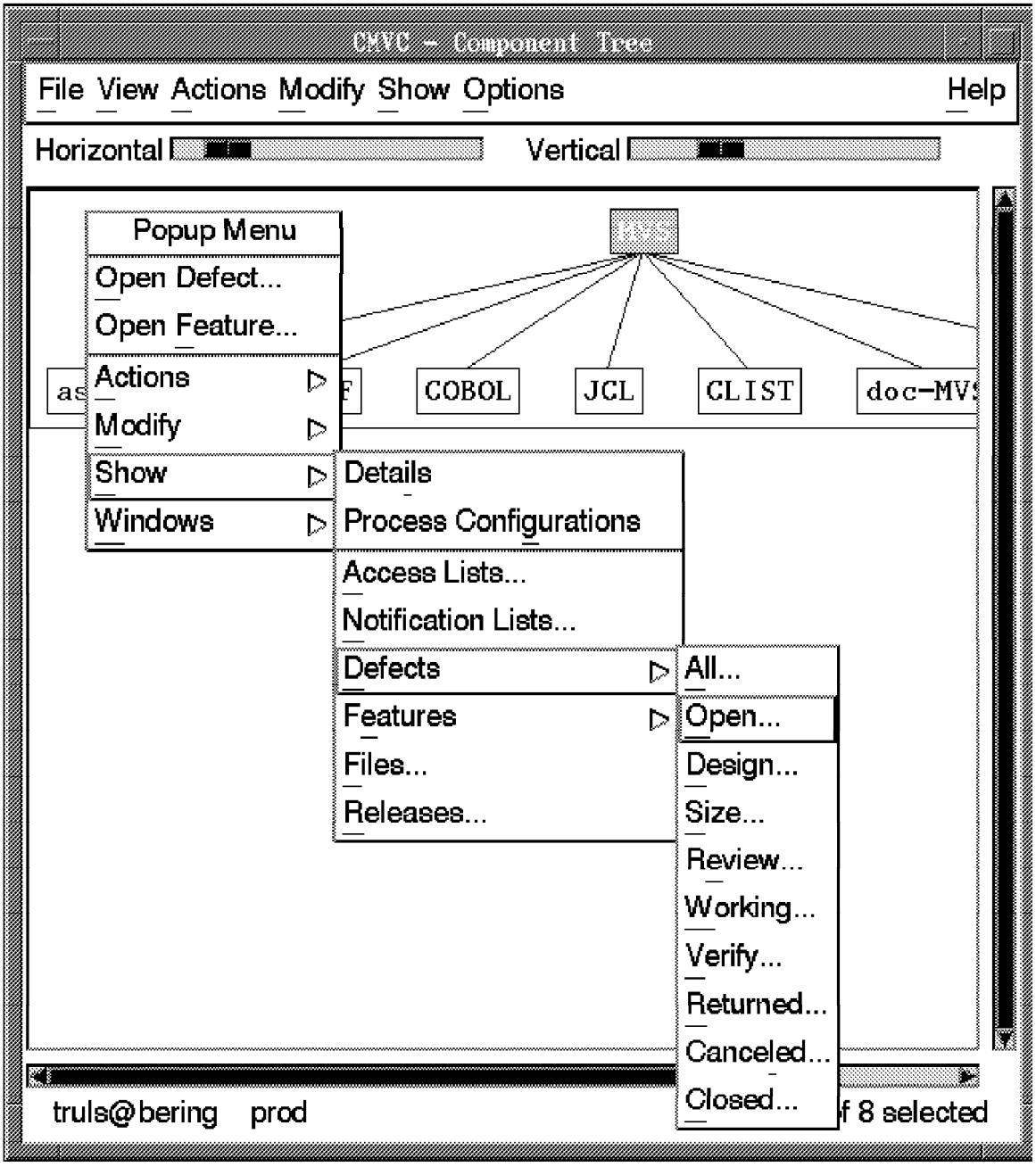

Figure 74. How to Display Open Defects from the CMVC Client GUI

To accept a defect from the CMVC - Defects window, highlight the defect record and select **Accept...** from the Actions pull-down. In the Accept Defects dialog box, enter the remarks. To choose a reason for accepting for the *Answer* field, select **Choices...**, and the list of available values is displayed. To accept the defect, select **OK** in the Accept Defects dialog box.

On our project, we had created a new choice for the list of reasons for accepting. This choice indicated that we were initializing a new release with the defect. Our MVS developer, therefore, selected *newbaseline* in the Choices List dialog box, when it was presented. Figure 75 on page 132 shows how the MVS developer accepted the *prod_0001* defect, using the CMVC client GUI.

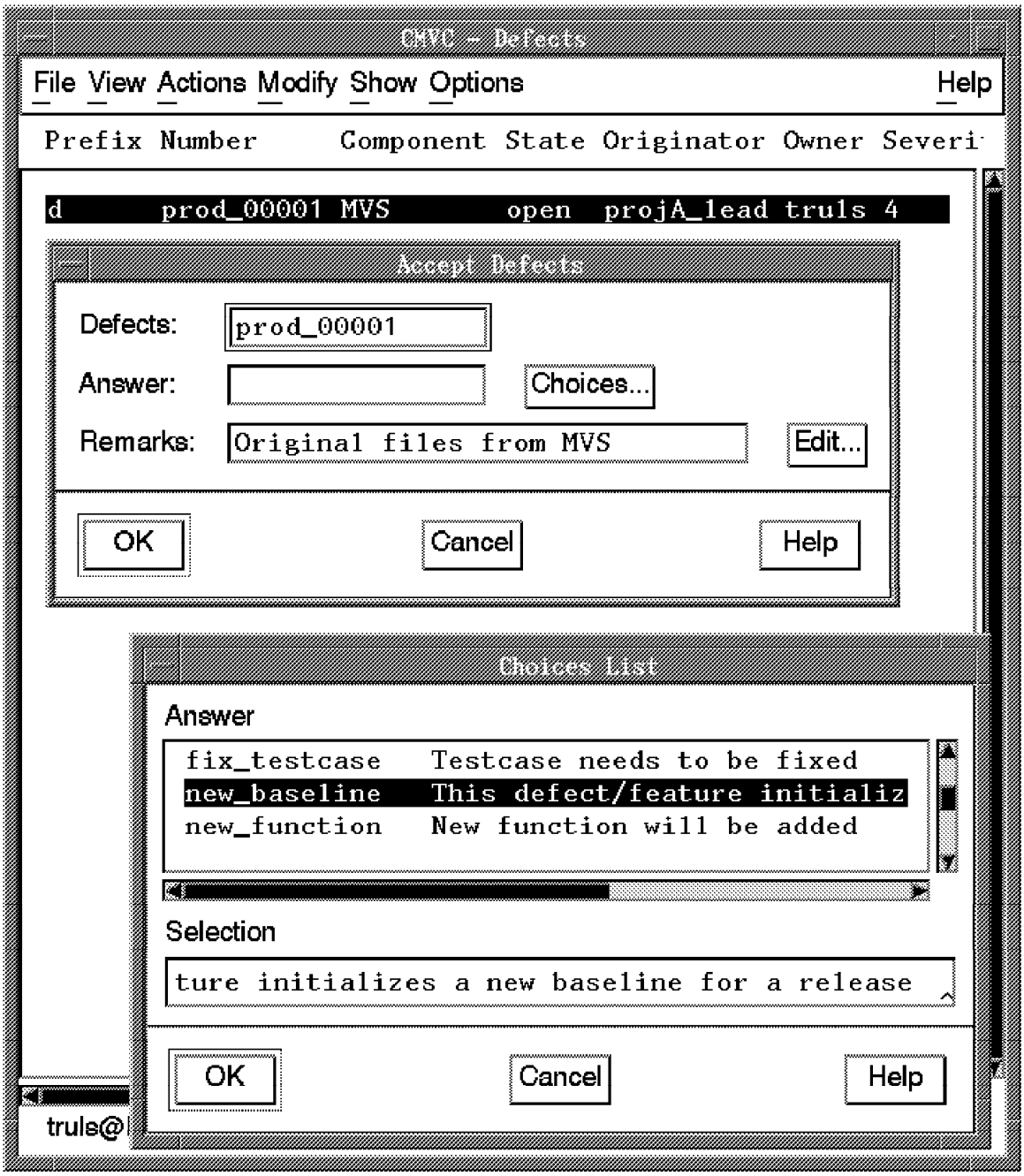

Figure 75. Accepting a Defect from the CMVC Client GUI

When a defect moves from *open* to *working*, CMVC automatically creates a verification record for the accepted defect, if the component process includes the *verify* subprocess. The *verify* subprocess ensures that the defect originator has the opportunity to rule on the acceptability of the resolution of the defect. The state of the verification record, initially, is *notReady*. The defect originator cannot verify that the defect has been satisfactorily resolved and close the defect while the verification record is in this state. Figure 76 on page 133 shows the relationships among component, defect, and verification records and their states at this point in time.

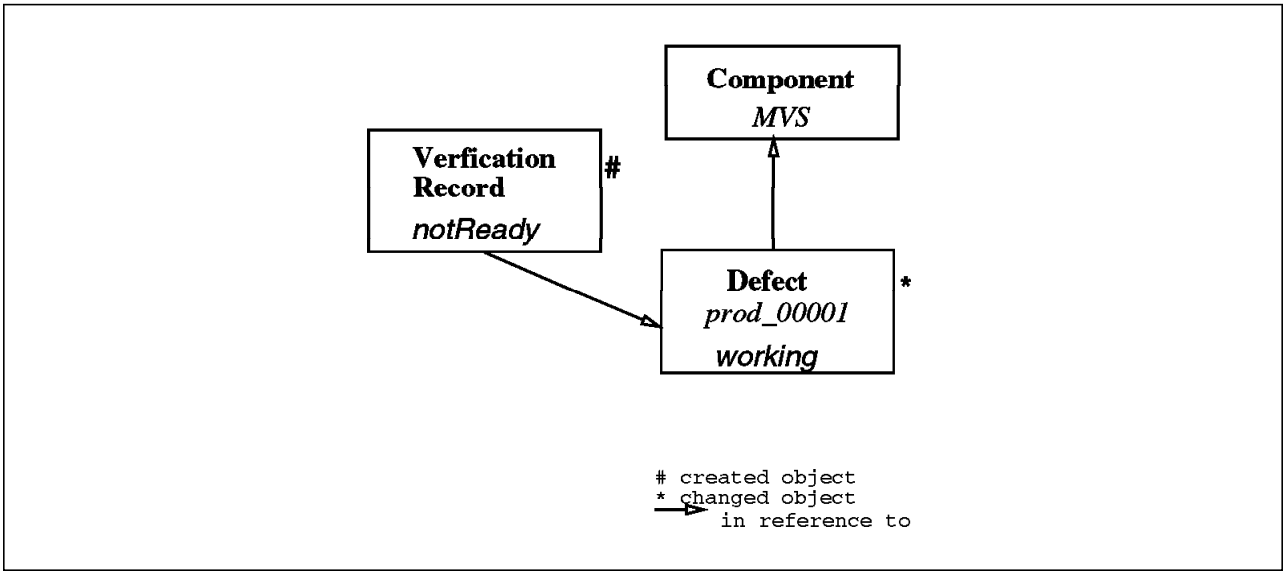

Figure 76. CMVC Object Relationships after Defect Acceptance

You can also use the CMVC command-line interface to accept a defect. We do not show an example of the **Defect** command, because it is not as convenient to use as the GUI window. One difficulty with this command is that it cannot prompt you for the choices lists. It is also awkward to enter remarks because you do not have access to an editor for lengthy remarks.

5.8.2 Creating a Track for the Defect and the Release

Changing the defect state to working enables the component owner to create a track in reference to the defect, if the release process associated with the defect includes the *track* subprocess.

In our project, the MVS developer created a track that associated the *prod_00001* defect with the *MVS_release_0* release. This track enables the source files to be created under authority of the defect and integrated in the release so they can be tested in a build.

To create a track, display the CMVC - Releases window by highlighting the **MVS** component in the CMVC - Component Tree window. Move the graphical cursor so it is not over a component figure, and press the right mouse button to display the CMVC Popup Menu. Select **Show**, and then on the cascading menu, select **Releases...**. Figure 77 on page 134 shows this CMVC Popup Menu and its cascading menu, with **Releases...** already selected.

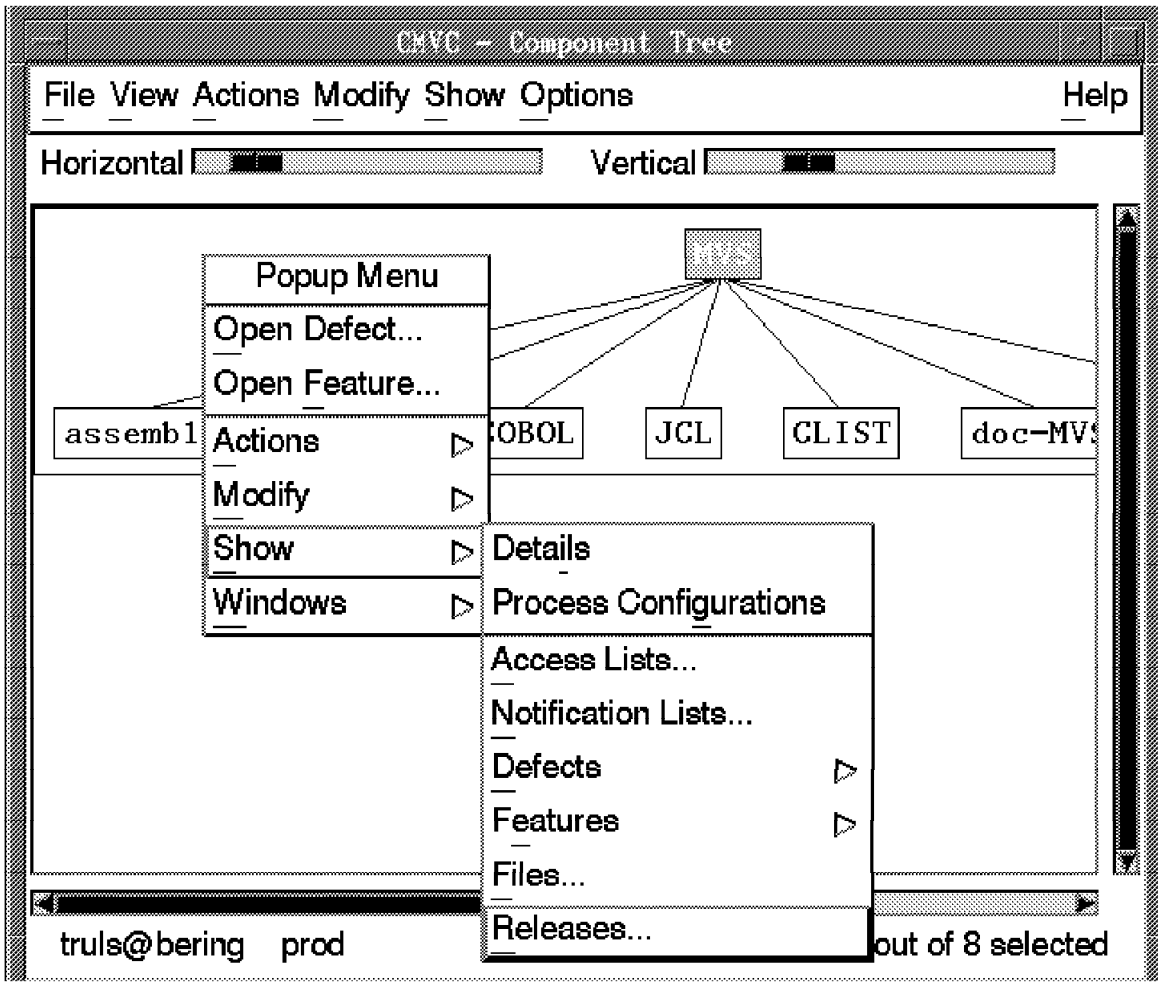

Figure 77. How to Display Releases for a Specific Component from the CMVC Client G

This selection displays the CMVC - Releases window showing all of the releases managed by the selected component. Highlight the release(s) for which you want to make a track. Next, highlight the defect in the CMVC - Defects window that was previously opened to accept this defect. Then select **Create Tracks...** from the Actions pull-down of the CMVC - Defects window. The Create Tracks dialog box comes up. Select **Import** to fill in the Releases field with the release names highlighted in the other window. The Target field can be used to express a date or level at which this track is targeted to be integrated. Typically, there is no need to enter the User field, because it will default to the defect owner, which at this initial stage in your project is acceptable. Select **OK**.

In our project, the MVS developer joined the *MVS_Release_0* release highlighted in the releases window with the *prod_0001* defect highlighted in the defect window to create the project's first track. The Target field had little value for our project, but we filled it in with an arbitrary value.

Figure 78 on page 135 shows how the MVS developer created a track for the *prod_00001* defect and the *MVS_release_0* release, using the CMVC client GUI.

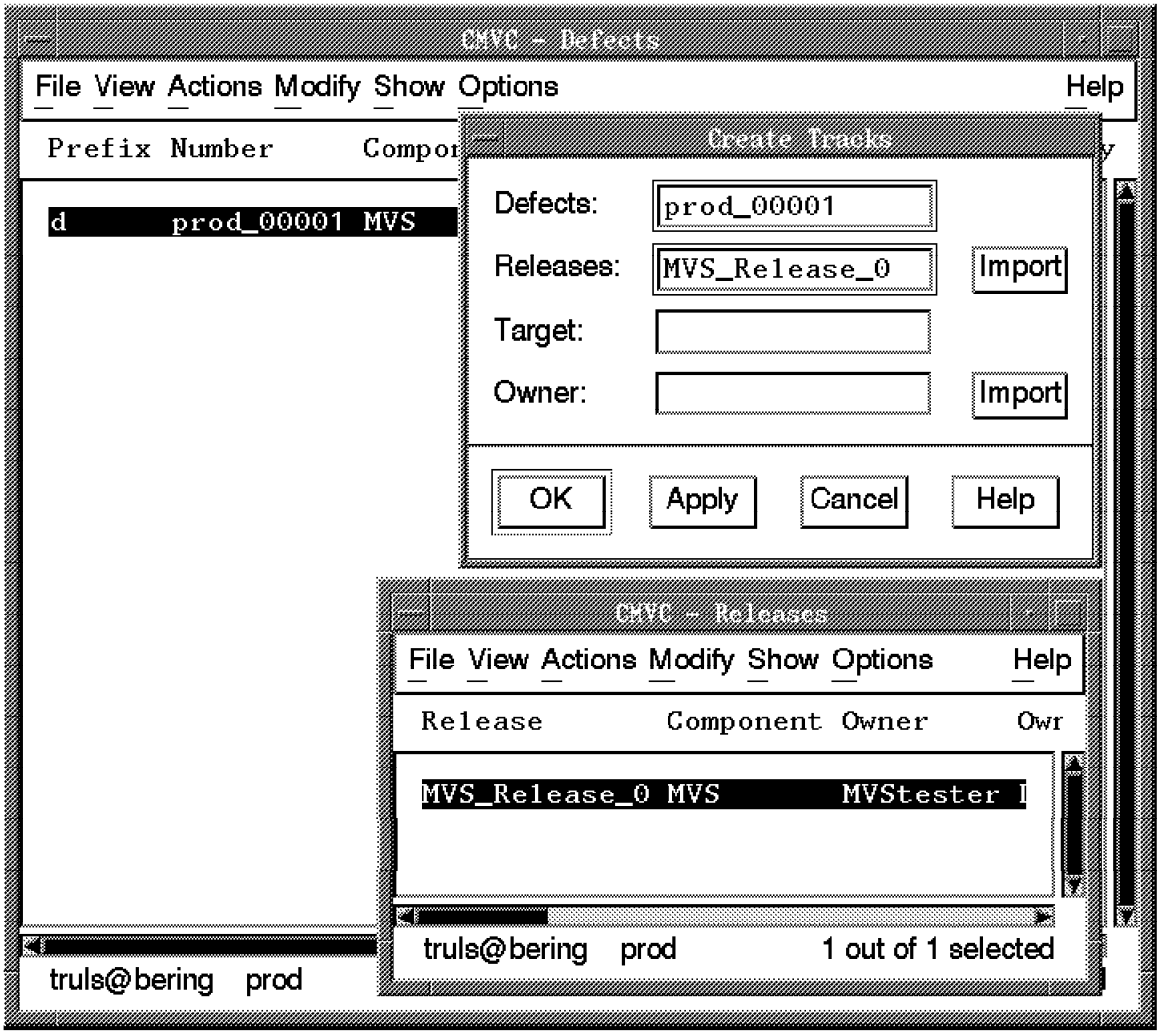

Figure 78. Creating a Track from the CMVC Client GUI

The state of the newly created track depends on which subprocesses are included in the release process. If the approval subprocess is included, for example, the state is now *approve*. If the fix subprocess is included, but the *approval* subprocess is not included, the the state is *fix*. If neither is included, the state is *integrate*.

Creating the track moves the state of the defect to the “working” state. If the release process includes the *test* subprocess, CMVC also automatically creates a test record indicating that, after the track is integrated with the release, it must be tested in a specific test environment. The test record is in the *notReady* state initially.

Our *MVS_Release_0* release process was *maintenance*, so the track went to the fix state, and a test record was created indicating that the *MVStester* CMVC user ID was responsible for testing the release in the *MVS_legacy* environment. Thus the newly established baseline would be tested on MVS to ensure that no errors occurred in the migration of source code from MVS to AIX.

You can also use the CMVC command-line interface to create and manipulate tracks. Figure 79 on page 136 shows the CMVC **Track** command. Specify the

release by setting the *CMVC_RELEASE* environment variable prior to execution of the CMVC **Track** command.

```
Track -create -defect prod_00001
```

Figure 79. Creating a Track with the Track Command

Figure 80 shows the relationships among the various CMVC objects involved in problem tracking at this point in time.

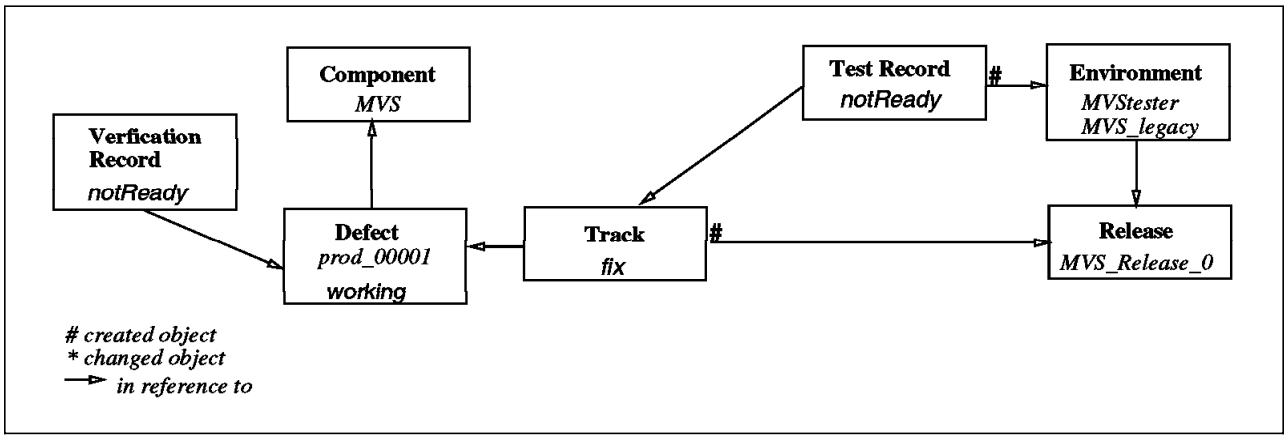

Figure 80. CMVC Object Relationships after Track Creation

The CMVC - Tracks window has a Corequisite pull-down menu. You can group tracks by creating corequisite links between them. This link will force you to add the grouped tracks as level members in the same levels.

Although we did not use this feature in our project, it is a valuable feature because sometimes a defect or feature will involve many changes to portions of your application source code files, which are associated with multiple distantly related components. You want multiple defects or features associated with the separate components, recognizing that different team members are responsible for the code changes. But, you also want to test the changes together at one time.

5.8.3 Creating the Files

After the track is created, you want to bring your initial source code files under CMVC control by creating CMVC files. Before you can do this, place the files in a directory path that includes the relative path name you want CMVC to associate with these files. (If you are using SDE WorkBench/6000 at this time, you also want to create context mappings for these directories. See E.2, “Significance of Context Mappings” on page 208 for more about context mappings.)

In our project, the MVS developer downloaded the files from the mainframe to the host, placing them in a directory path that terminated in `source/cobol`.

To create CMVC files, start in the CMVC - Component Tree window and highlight the component that will manage the new CMVC files. Display the Windows pop-up menu and select **Files** to display the CMVC - Files window. To specify the directory in which these files are presently stored, select **Set Directory...** from the Option pull-down. Enter an absolute path name in the Directory field and

select **Create...** from the Actions pull-down. The Create Files dialog box is displayed.

Display the CMVC - Defects window and highlight the defect associated with the track that will integrate these files into the release. Display the CMVC - Releases window, and highlight the release associated with the track. In this case, you can only identify a single release. Now, enter values in the fields of the Create Files dialog box by selecting **Import**. You could enter these fields by keying them in, if you do not care to import them. List all of the files located in the source directory by selecting **Choices...**. Toggle the Text or Binary push button to indicate the type of data in the files. Select **OK**. CMVC checks the existence of a track for the specified defect and release. If the checking is negative, a error message is displayed.

In our project, the MVS developer placed the MVS source files in `/home/aixcase2/source/cobol`. The relative path name associated with our CMVC files was `source/cobol`. Figure 81 on page 138 shows how the MVS developer brought the COBOL files under CMVC control, using the CMVC client GUI.

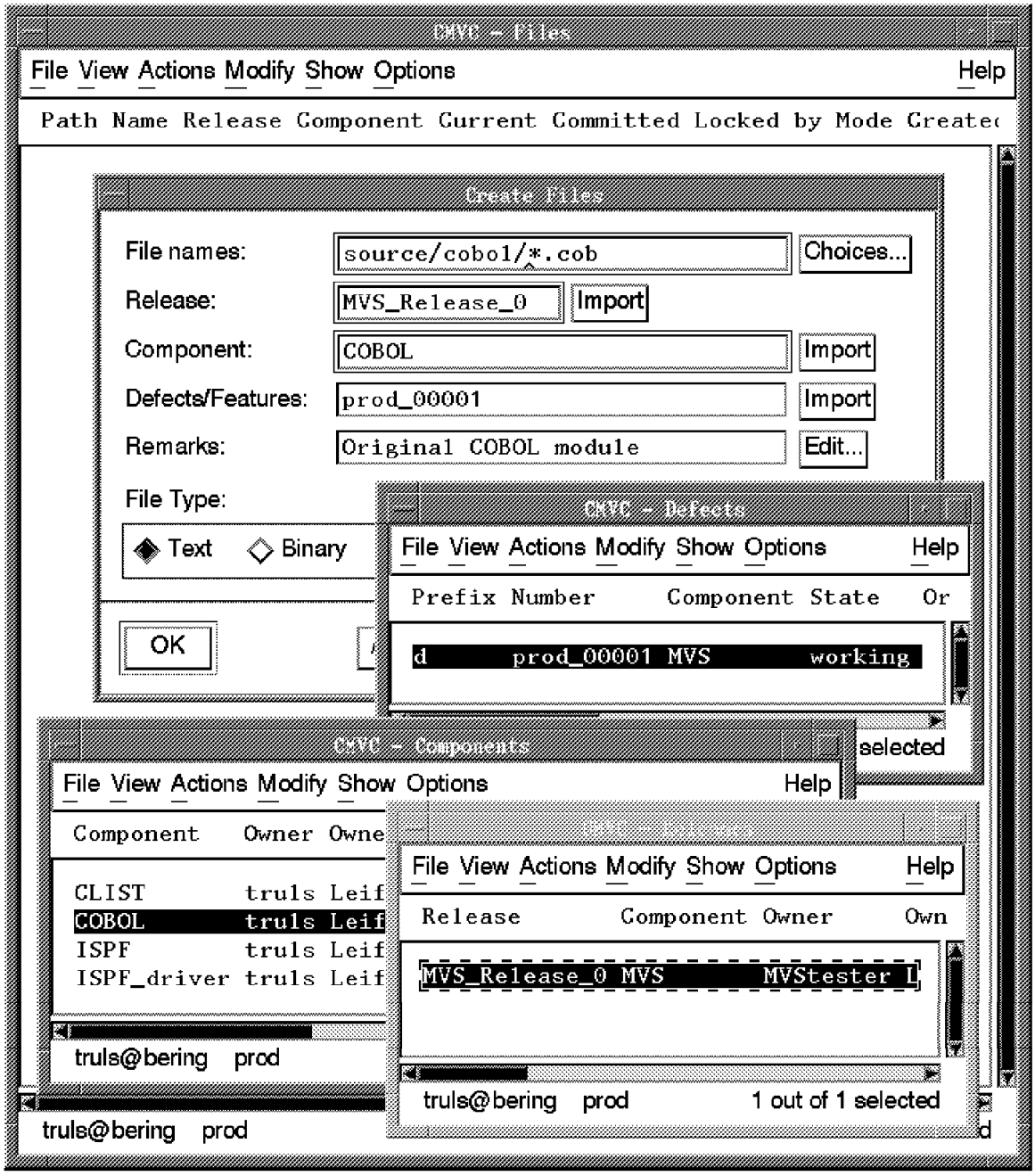

Figure 81. Creating Files in a CMVC Family

You can also use the CMVC command-line interface to create files with the CMVC. Figure 22 on page 33shows the **File** command. The CMVC command-line interface is more convenient if you need to create a lot of files at once. But, if you are developing new code, you will probably use the CMVC client GUI more often.

If you are using SDE WorkBench/6000, you may notice, in the Development Manager window, that the UNIX file permissions change to read-only after each file is placed under CMVC control. To check on the success of your file creation, display the CMVC - Releases window, apply a query to list this release, and highlight the record for this release. Then select **Files...** from the Show

pull-down to display the CMVC - Files window, which shows all of the files created.

Figure 82 shows the file list of our first release, *MVS_Release_0*. This screen capture shows only four columns of data. To the right of the column showing the "current version" of each file, is another column showing the "committed version." Our figure does not show that column, because at this point in time it is empty. The current version of a file is the highest version checked in to CMVC, but until that version is successfully integrated into a release or level, it is not committed. During integration testing, errors may be detected and another version checked in again. The files are to be committed when the level that in which they are integrated is committed.

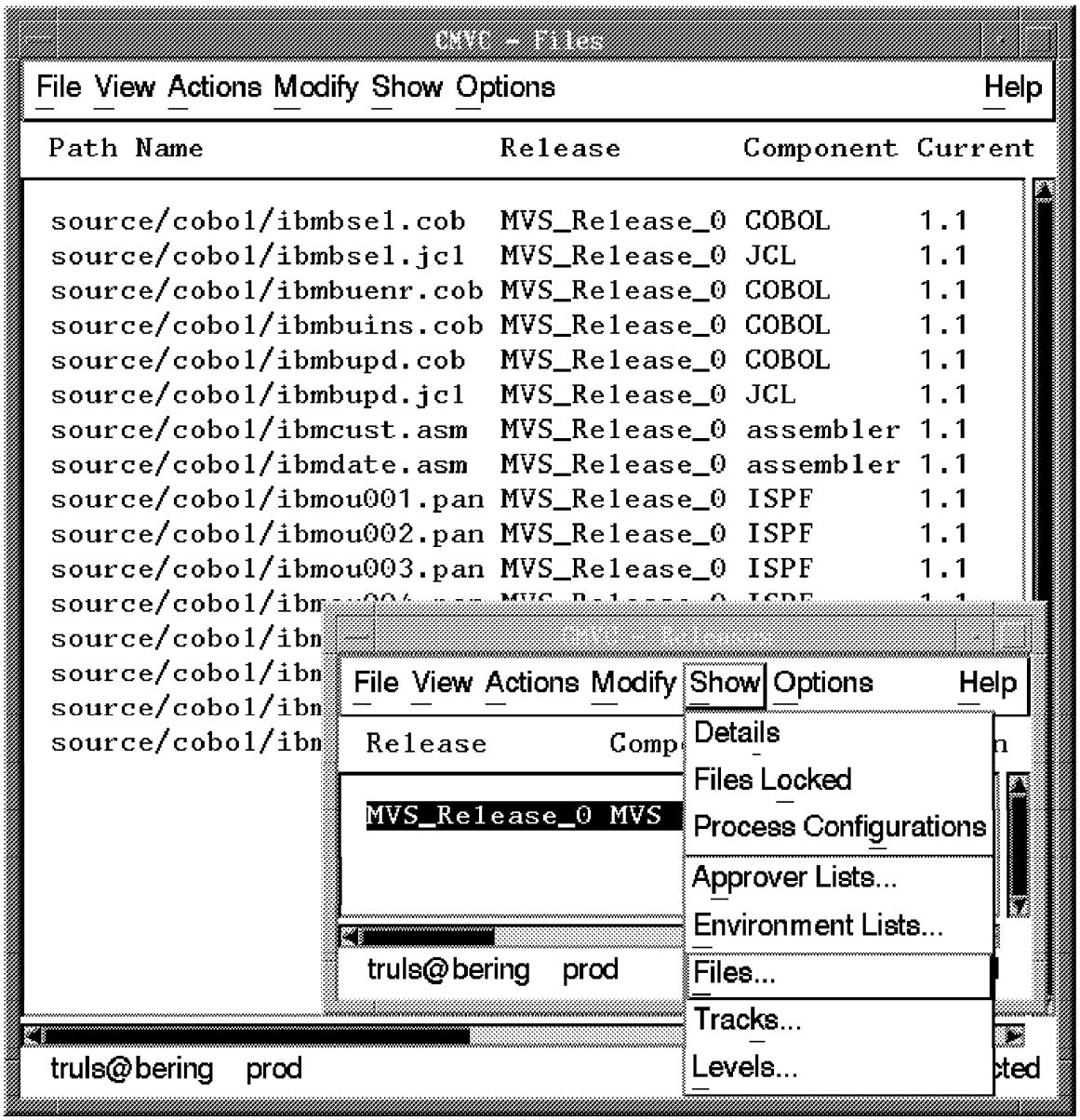

Figure 82. List of Created Files for a Release

Because our project created files in reference to the components *COBOL*, *JCL*, *assembler*, *ISPF*, and *CLIST* using a process that includes the *fix* subprocess, CMVC automatically created a fix record for each of those components. You can

also create fix records yourself before fixing the defect, to force changes on files managed by certain components. In this case, the initial state of the fix records is *notReady*. Figure 83 on page 140 shows the relationships among the various CMVC objects involved in the problem tracking at this point in time.

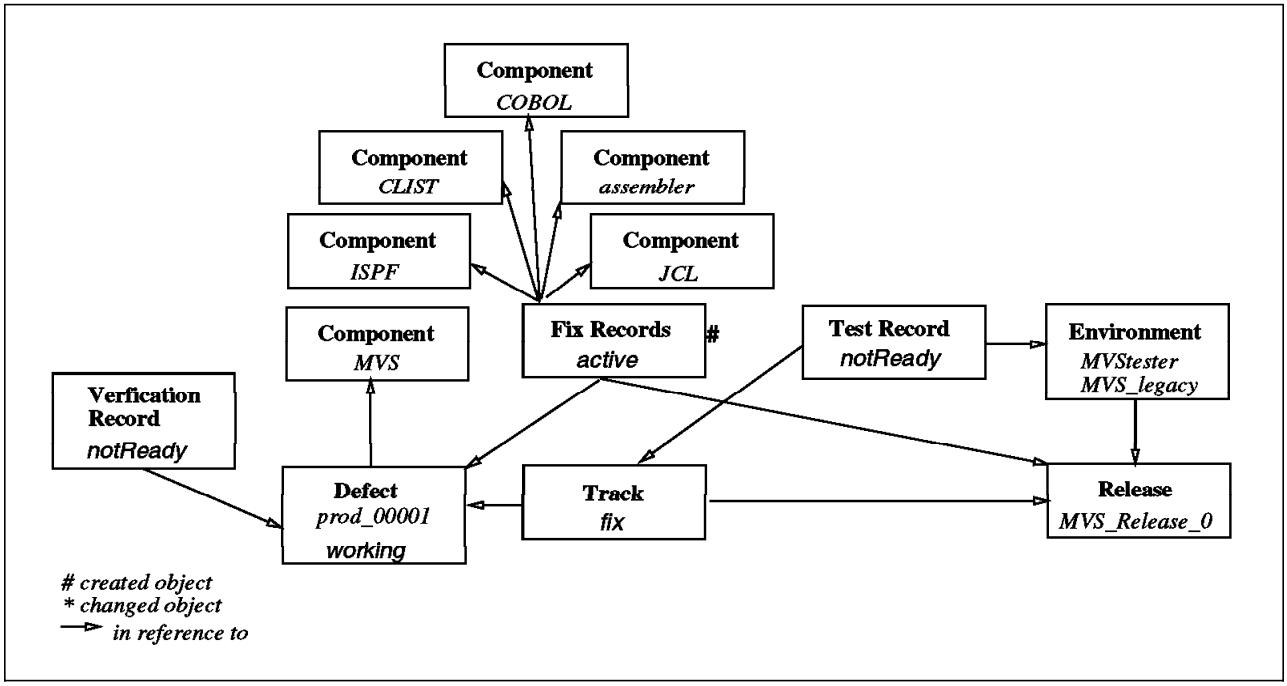

Figure 83. CMVC Object Relationships after File Creation

5.8.4 Integrating the Track by Completing Fix Records

To inform CMVC and the rest of the development team that you have fixed the *prod_00001* defect in the *MVS_Release_0* release, you have to move the track state from *fix* to *integrate* by completing all of the associated fix records.

To move the track to the integrate state, you must display a list of the fix records associated with this track. Highlight the track in the CMVC - Tracks window and then Select **Fix Records...** from the Show pull-down to display the CMVC - Fix Records window showing the right fix records.

To complete the fix records highlight all of them. Press the left mouse button while the mouse is pointing to the first record, and without releasing the left mouse button, bring the mouse down the list, highlighting each subsequent fix record until the last one is highlighted, then release the left mouse button. Select **Complete...** from the Action pull-down and select **Yes** in the Complete Fix Records dialog box to confirm the fix completion.

Figure 84 on page 141 shows the CMVC - Fix Records window with all of the fix records associated with the *prod_00001* defect

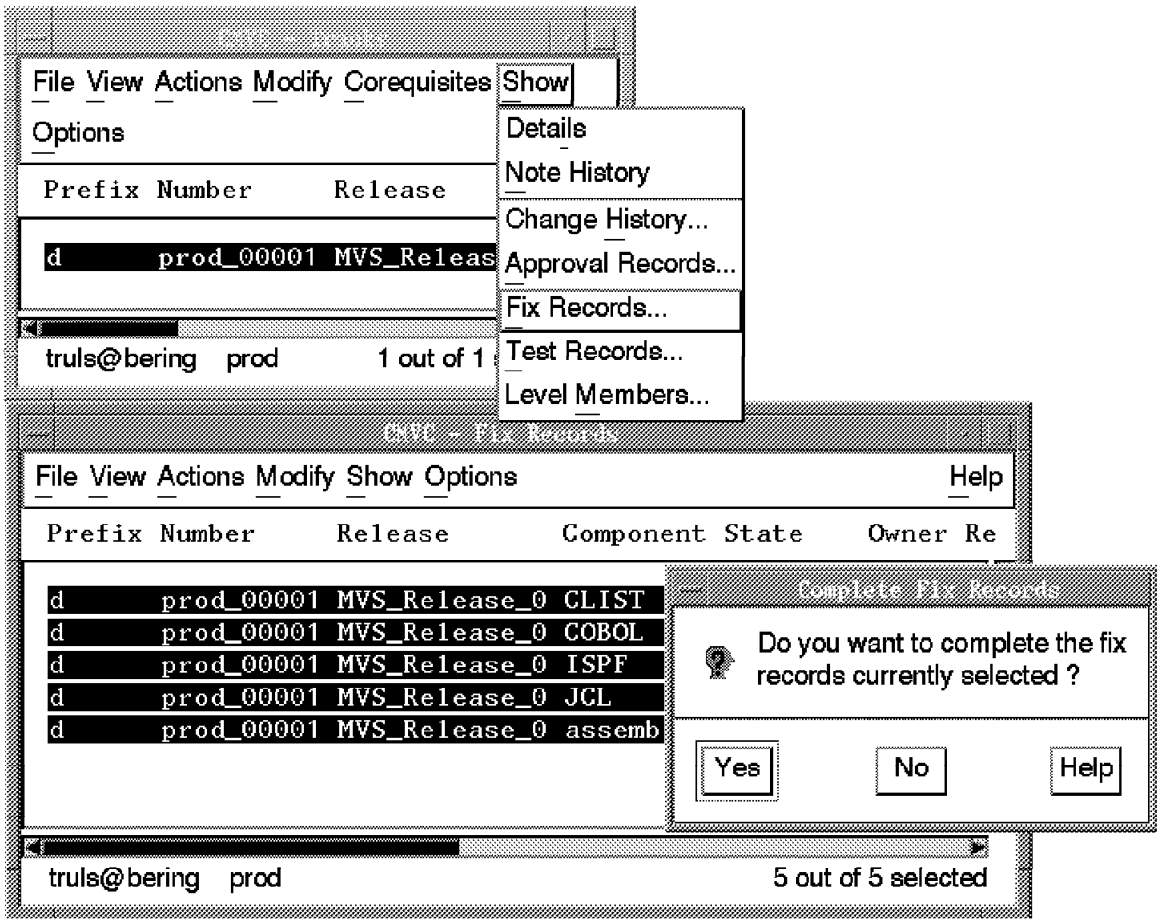

Figure 84. Completing Fix Records from the CMVC Client GUI

The track connects the file, defect, and release. The fix record relates the defect, release, and component. Figure 85 on page 142 shows the relationships among the various CMVC objects at this point in time. The track has moved to the *integrate* state, and the fix record has moved to the *complete* state. A track with the *integrate* state can no longer be used for creating or changing a file. However, the track state could be moved back from *integrate* to *fix* and then could be reused. Refer to Appendix D, “Hints and Tips for Using CMVC” on page 193 for a description of the procedure to reuse an integrated track.

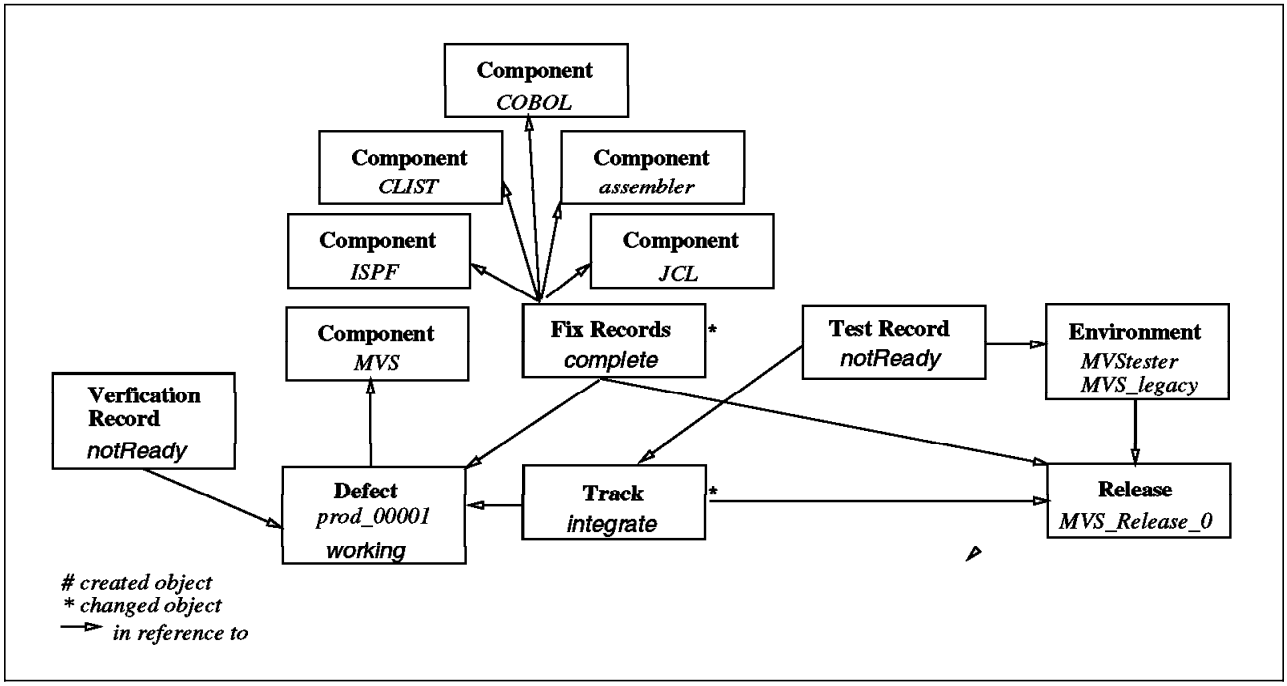

Figure 85. CMVC Object Relationships after Defect Fixing through Fix Records

You can list the files created in reference to the track by selecting the track in the CMVC - Tracks window. Then select **Change History...** from the Show pull-down to display the resulting query in the CMVC - Change History window. Figure 86 on page 143 shows all of the CMVC files created for our project under authority of the *prod_0001* defect and integrated into the *MVS_Release_0* release by the CMVC FileCreate action. This display also shows which version of each file was integrated, and the relative path name associated with the file name.

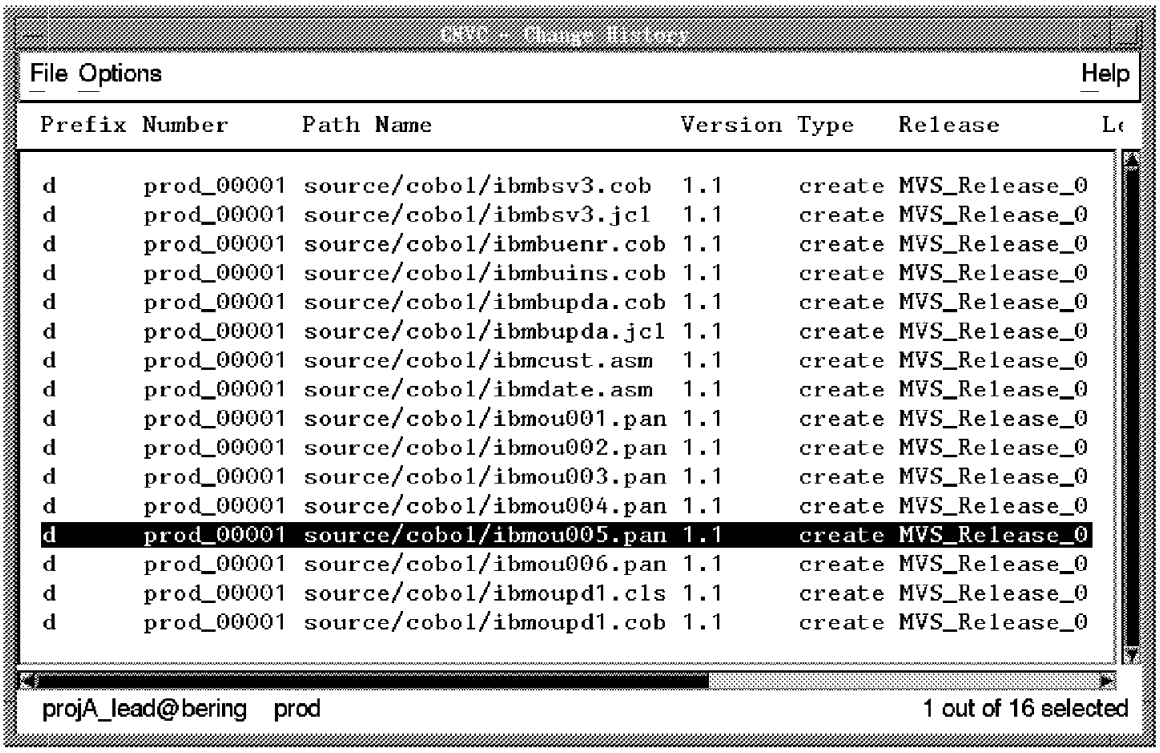

Figure 86. Track Change History

5.9 Builder Asks about Building Application on MVS

In 2.2.10, “Builder Asks about Building Application on MVS” on page 34 our MVS builder received advice on how to establish a baseline by creating the *0* level. The MVS builder integrated the files created under authority of the track associated with the *prod_00001* defect and the *MVS_Release_0* release by making that track a level member of this level. This was necessary because the release process included the level subprocess.

5.9.1 Creating a Level for the Release

To create the level, first open the CMVC - Releases window to highlight the release for which you intend to create this level. Next, open the CMVC - Levels window and select **Create...** from the Actions pull-down. When the Levels dialog box is displayed, enter 0 in the Levels field and enter the Release field by selecting **Import**. To choose a value for the Type field, select **Choices...** to display the list of the valid level types and select one. Complete the action by selecting **OK** in the Create Levels dialog box. The level is created but it has no level members yet, which means no tracks are associated with the level yet. It is in the *working* state; you can still delete it.

Figure 87 on page 144 shows how the MVS builder created the *0* level in the *MVS_Release_0* release, using the CMVC client GUI. Since this represented the current production release of the MVS application, the MVS builder chose the *production* level type value.

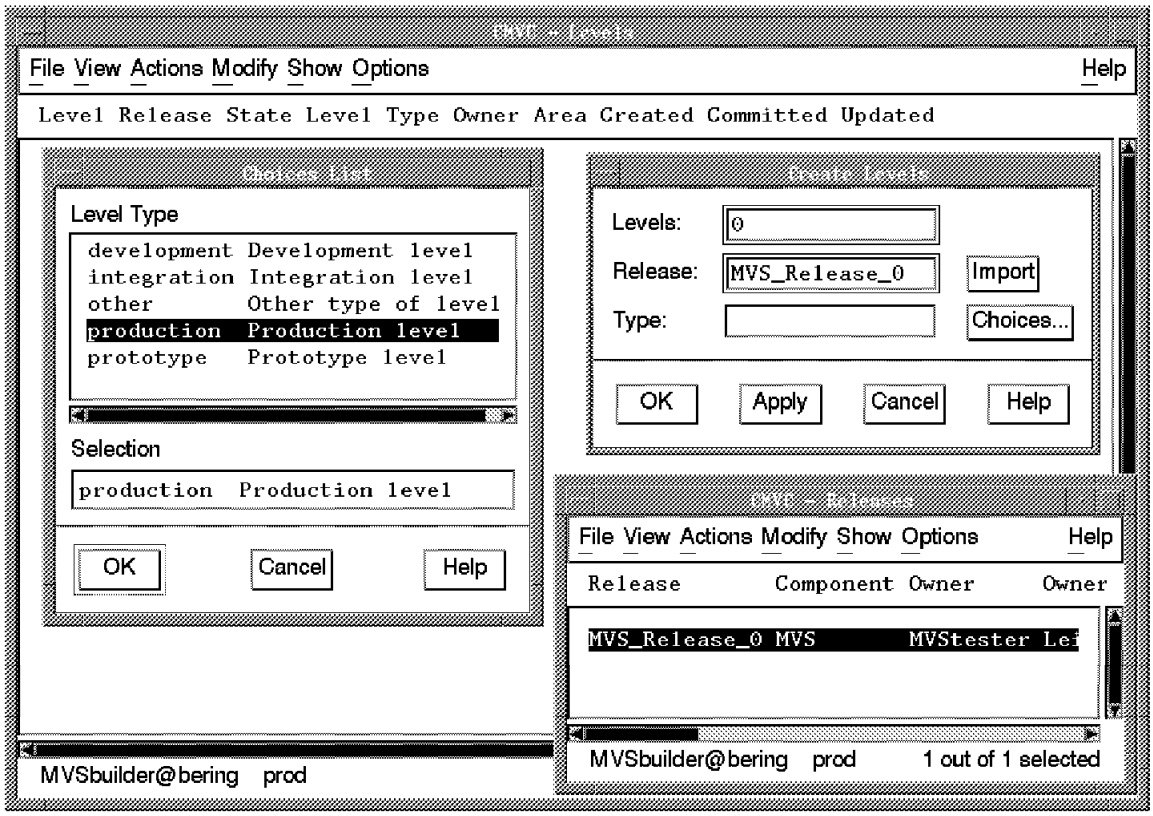

Figure 87. Creating a Level from the CMVC Client GUI

You can also use the CMVC command-line to create a level. We do not illustrate the **Level** command for this case, because, like the **File** command, it is less convenient to use than the GUI window. For example, it cannot prompt you to choose an appropriate *type* value.

Figure 88 on page 145 shows the relationships among the various CMVC objects after creating the level.

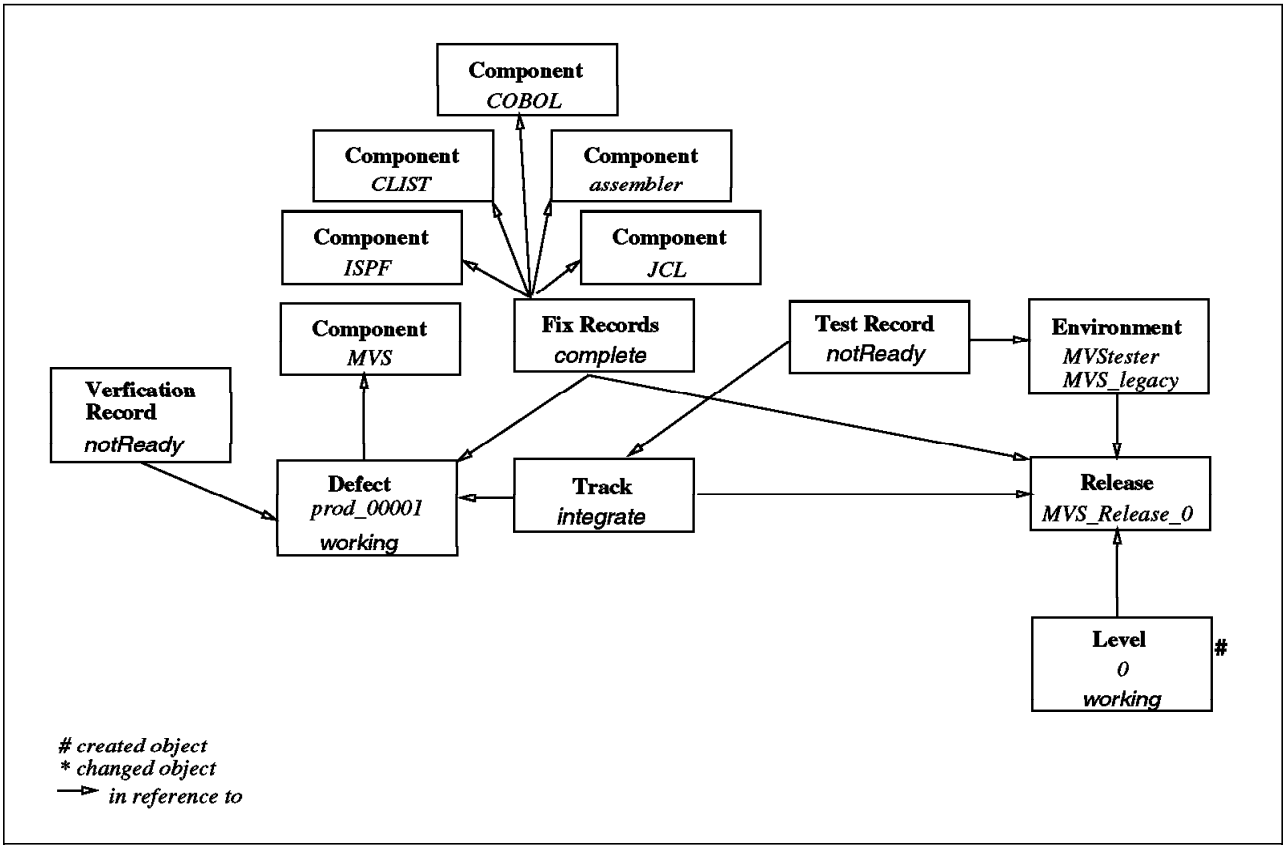

Figure 88. CMVC Object Relationships after Level Creation

5.9.2 Creating a Level Member

To create a level member, you identify one or more tracks associated with the same release that defines the level. This action links all of the files changed in reference to the defects associated with the tracks.

To create a level member, open the CMVC - Tracks window and highlight the track associated with the defect and release that you want to integrate and test. Display the CMVC - Levels window and highlight the level of which you are making a member. Now, select **Add Level Members...** from the Actions pull-down. When the Add Level Member dialog box opens, select **Import** to fill in the Tracks and Level fields. Complete the action by selecting **OK**.

Our MVS builder created a level member referencing the track associated with the *prod_0001* defect and the *0* level of the *MVS_Release_* release. Figure 89 on page 146 shows how the MVS builder added a track representing the *prod_00001* defect as a level member in the *MVS_Release_0* release, using the CMVC client GUI.

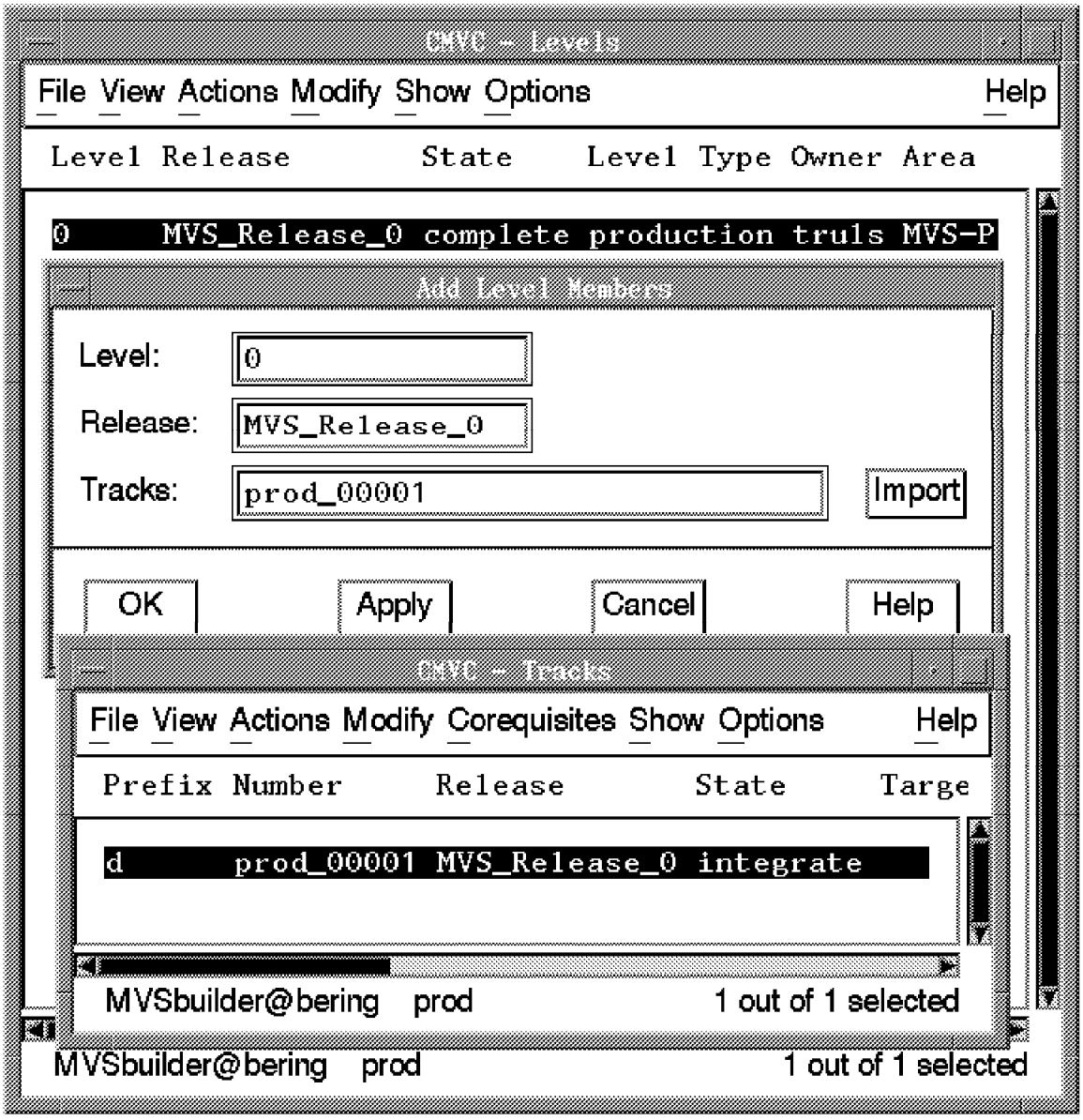

Figure 89. Creating a Level Member from the CMVC Client GUI

You can also use the CMVC command-line interface to create level members. Figure 90 shows the **LevelMember -create** command to add a level member.

```
LevelMember -create -level 0 -defect prod_00001 -release MVS_Release_0
```

Figure 90. Creating a Level Member with the LevelMember Command

Only tracks in the *integrate* state should be defined as level members, but you can add tracks in the *fix* state, which indicates that a change is pending. Adding a track in the *fix* state results in the message shown in Figure 91 on page 147.

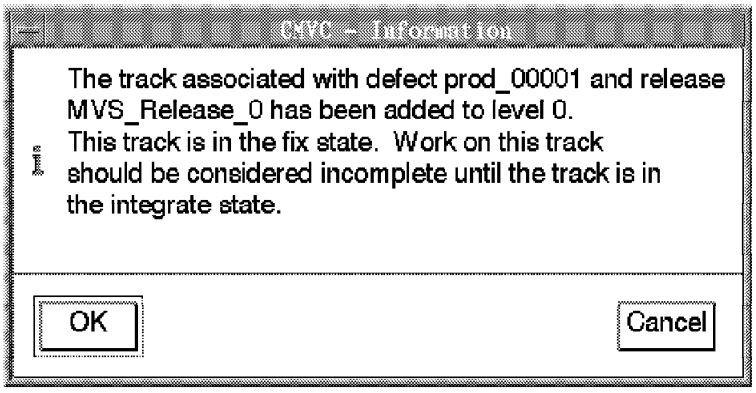

Figure 91. Fix Track as Level Member Information

Once we had created the level member to incorporate our track, all CMVC objects related to the migration of the MVS application from MVS to AIC were created. From then on, the actions we performed could only change the state of these objects, not create new objects. Figure 92 shows the relationships among the various CMVC objects now. Notice the new state of the level is *integrate*. A level with this state can only be partially extracted by the *developer* who has CMVC authority to perform unit tests. (Partially means only the files changed in reference to defects or features defined as level members of that level.) Refer to 5.9.5, “Extracting the Level or the Release” on page 151 for more about full and delta or partial CMVC ReleaseExtract actions.

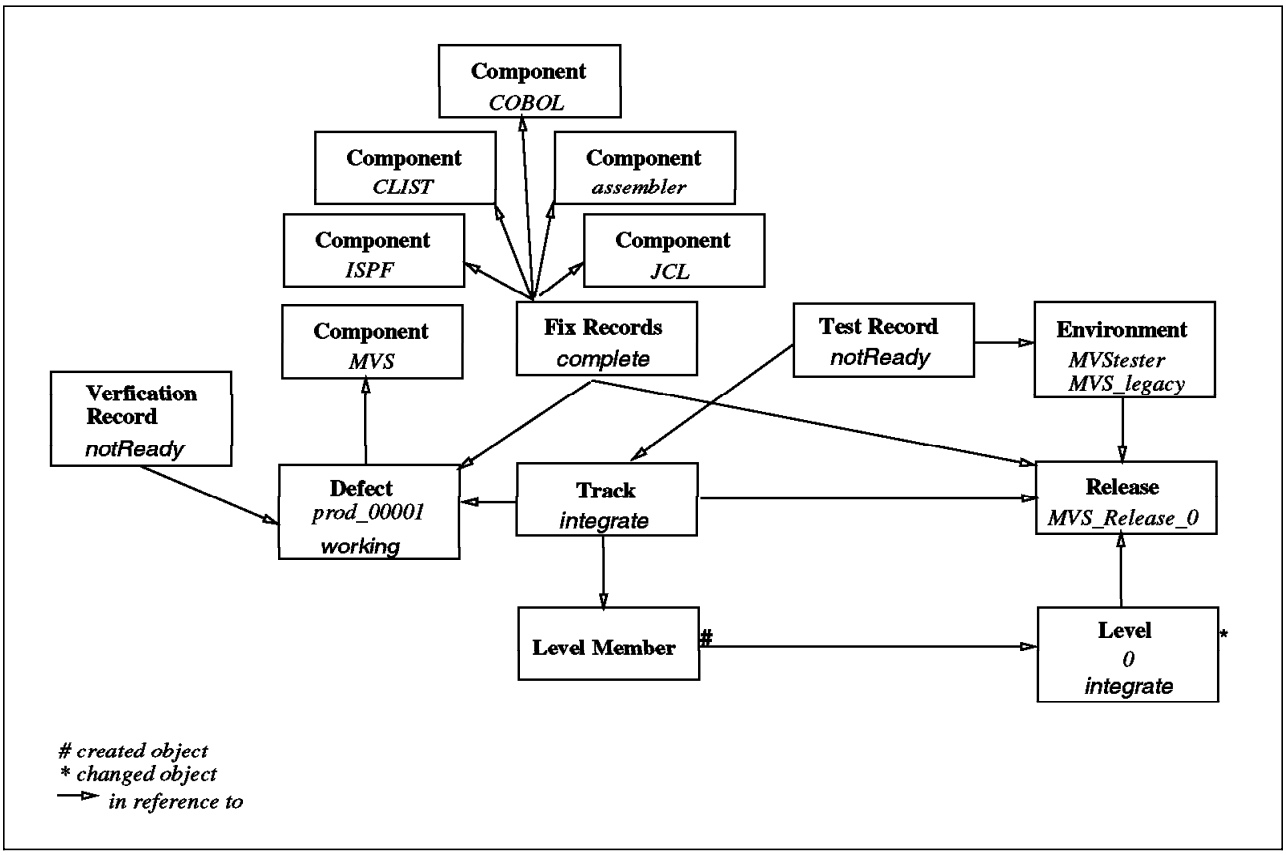

Figure 92. CMVC Object Relationships after Level Member Creation

If you want to change a file in reference to the track integrated in the *0* level, you can still do it but it is not very obvious. Refer to Appendix D, “Hints and Tips for Using CMVC” on page 193for a description of the procedure.

5.9.3 Committing the Level

Be careful before committing a level! Remember the following facts:

- The level state cannot be moved back from the *commit* state to the *integrate* state.
- You cannot add or remove a level member.
- A committed level cannot be deleted.
- The tracks defined as level members can no longer be used to make file changes. You have to create new tracks.
- The current version of files becomes the committed version.

To commit the level, highlight that level in the CMVC - Levels window and then select **Commit...** from the Actions pull-down. When the Commit Levels dialog box is displayed, select **OK**. After level commitment, the current version of files becomes the committed version. Figure 93 shows how the MVS builder committed the *0* level of the *MVS_Release_0* release, using the CMVC client GUI.

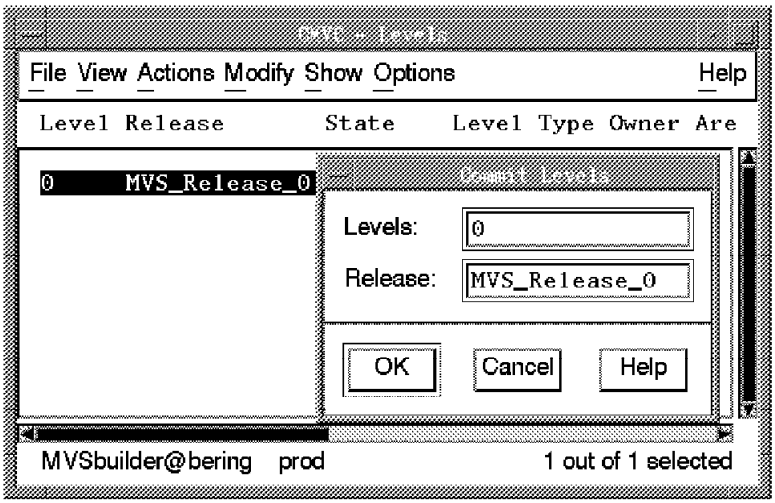

Figure 93. Committing a Level from the CMVC Client GUI

You can also use the CMVC command-line interface to commit a level.
Figure 94 shows the **Level -commit** command. Specify the release by setting the *CMVC_RELEASE* environment variable prior to execution of the CMVC **Level -commit** command.

```
Level -commit 0
```

Figure 94. Committing a Level with the Level Command

Figure 95 on page 149 shows the relationships among the various CMVC objects at this point in time. You can see that both the track and level states have moved from *integrate* to *commit*.

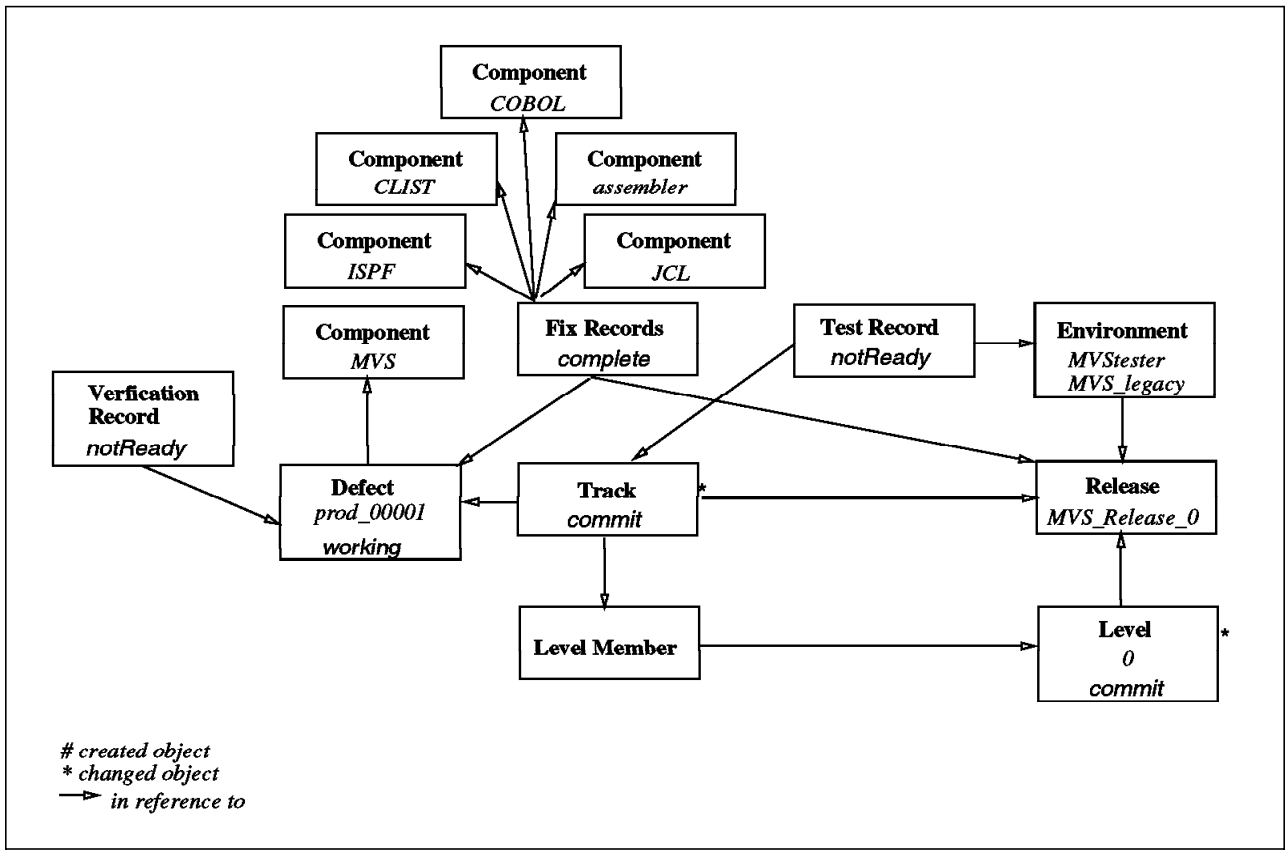

Figure 95. CMVC Object Status after Level Commitment

When a level is committed, CMVC creates a level map file in the `maps` directory, located in the family home directory. The name of the level map file is composed of the release and level name. The name of our map file is `MVS_Release_0/0`. This map file contains a list of the names of the changed files, their versions, and the type of changes, such as *create*, *delta*, or *rename*. This file could be used as a definitive listing of the configuration of your application at a production release. Figure 96 on page 150 shows the level map file created for the *0* level of the *MVS_Release_0* release. You can see four different fields:

- The file name
- The file version identifier in the database table *files*
- The file identifier in the database table *files*
- The change type: *create*, *link*, *rename*, *noDelta*, *delete*, and *delta*.

```
source/cobol/ibmdate.asm 56 57 create
source/cobol/ibmcust.asm 59 60 create
source/cobol/ibmou004.pan 62 63 create
source/cobol/ibmou003.pan 71 72 create
source/cobol/ibmou002.pan 74 75 create
source/cobol/ibmou001.pan 77 78 create
source/cobol/ibmou006.pan 80 81 create
source/cobol/ibmou005.pan 83 84 create
source/cobol/ibmbupda.jcl 86 87 create
source/cobol/ibmbsv3.jcl 89 90 create
source/cobol/ibmbsv3.cob 92 93 create
source/cobol/ibmbupda.cob 95 96 create
source/cobol/ibmbuins.cob 98 99 create
source/cobol/ibmbuenr.cob 101 102 create
source/cobol/ibmoupd1.cob 107 108 create
source/cobol/ibmoupd1.cls 113 114 create
```

Figure 96. /production/maps/MVS_Release_0/0 Level Map File

5.9.4 Completing the Level

What is the difference between a committed and complete level? These are identical, if the process selected for the release does not include the *test* subprocess. If the *test* subprocess is defined for this release, the test records move to the *ready* state only after completion of the committed level.

To complete a level, display the CMVC - Levels window, highlight the **0** level, and then select **Complete...** from the Actions pull-down. When the Complete Levels dialog box comes up, select **OK**. Figure 97 shows how the MVS builder committed the *0* level of the *MVS_Release_0* release, using the CMVC client GUI.

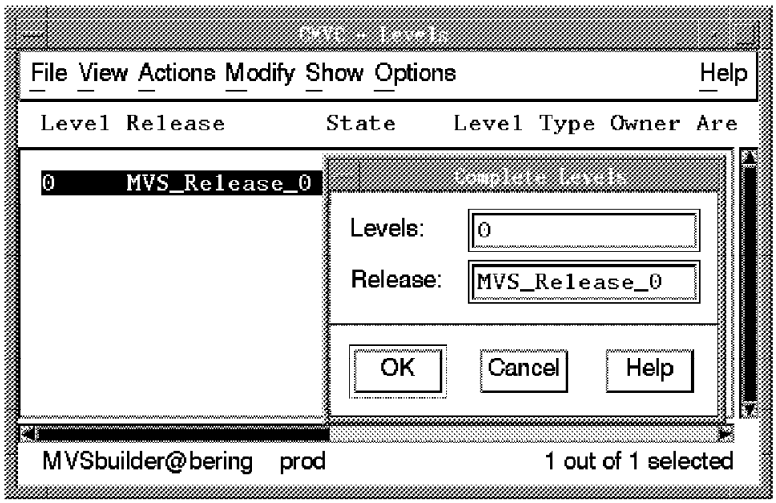

Figure 97. Completing a Level from the CMVC Client GUI

You can also use the CMVC command-line interface to complete a level. Figure 98 on page 151 shows the **Level -complete** command. Specify the release by setting the *CMVC_RELEASE* environment variable prior to execution of the CMVC **Level -complete** command.

```
Level -complete 0
```

Figure 98. Completing a Level with the Level Command

In Figure 99, you can see that the track state has moved from *commit* to *test*, the level state from *commit* to *complete*, and the test record is now in the *ready* state.

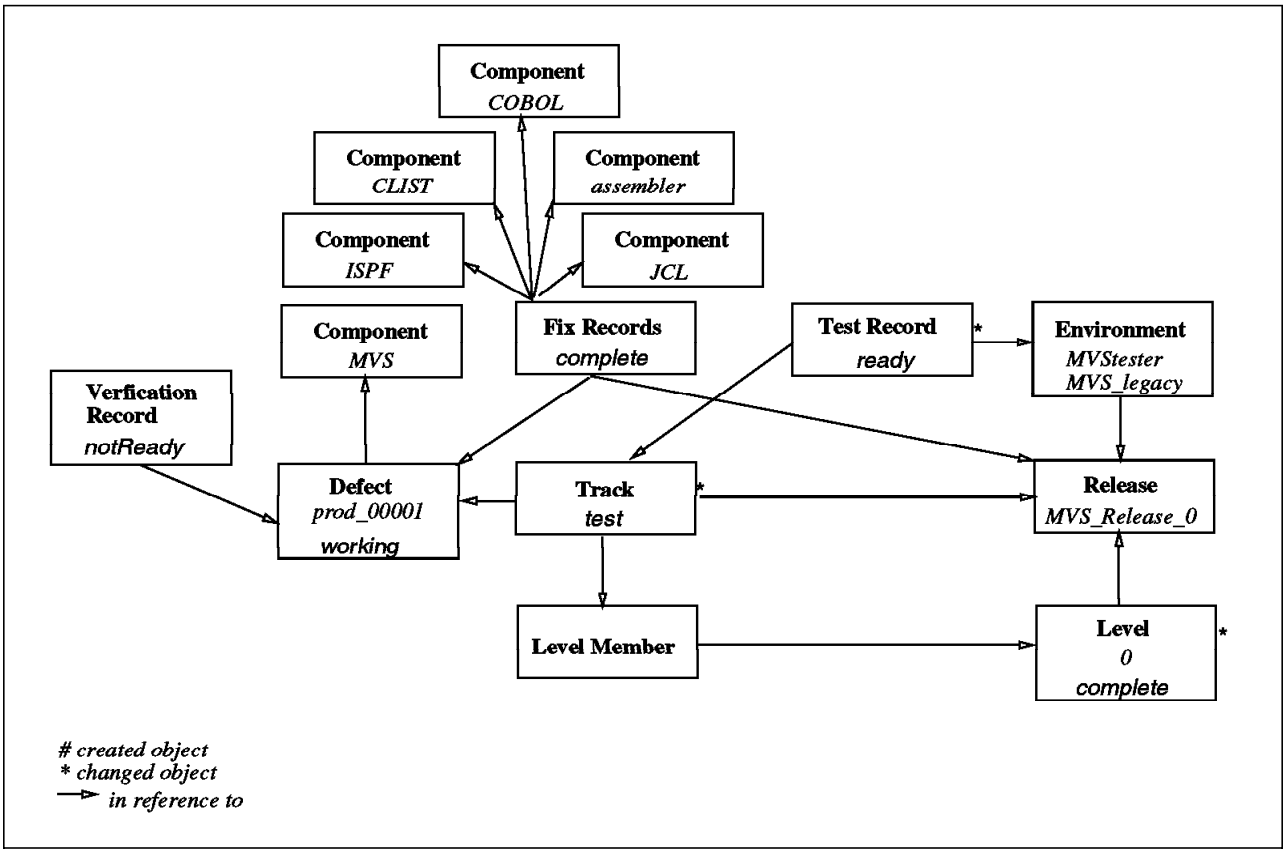

Figure 99. CMVC Object Status after Level Completion

5.9.5 Extracting the Level or the Release

The level extraction depends on the progress of the problem tracking (level state), but the release extraction can be executed at any time. When extracting a release, you can choose which versions of all the files should be extracted by selecting **Current version** or **Committed version** in the Extract Releases dialog box. You can also specify a date after which you want to extract the updated files. You can perform a delta or full extraction of a committed or complete level, but you can only partially extract an integrated level (delta extraction).

Full extraction means that all changed files belonging to the level, as well as the other files belonging to the release, are extracted. An integrated level can be assimilated by an existing production release of your application. A committed or complete level is a complete, new production release of your application. When a level or release is extracted, CMVC builds the file system according to the path names of the extracted files in the target directory.

To extract the level, display the CMVC - Levels window, highlight the *0* level, and then select **Extract...** from the Actions pull-down. When the Extract Levels dialog box is displayed, enter the fields according to your build target environment, and then select **OK**. The extraction target directory should be NFS-exported on the specified host. Select **Expand keywords** to transform the SCCS keywords contained in your source files into CMVC information such as file name, version, component, release, or level when the extract occurs. Refer to Appendix F, "Source File and Program Identification with CMVC Keywords" on page 211 for more information about the CMVC keywords. Figure 100 shows how the MVS builder did a full level extract of the *MVS_Release_0* release, using the CMVC client GUI. The files in which SCCS keywords were expanded were extracted to the product file tree, which is exported from its physical host, *bering*, to the developer's workstation host, *bengal*.

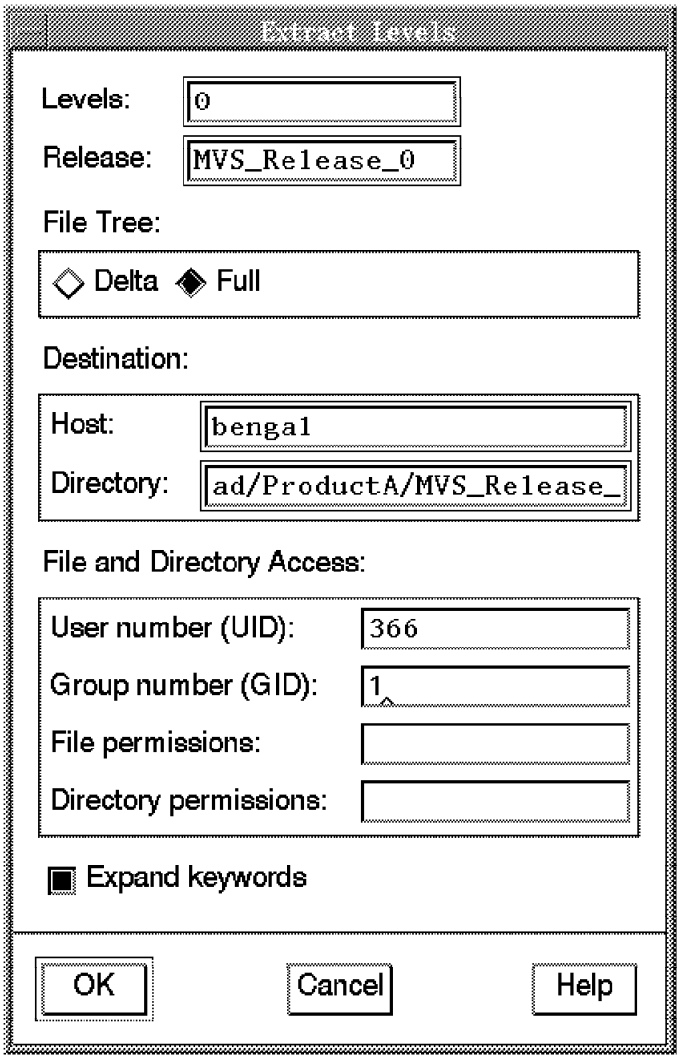

Figure 100. Level Extraction from the CMVC Client GUI

To extract a release, display the CMVC - Releases window, highlight the release, and then select **Extract...** from the Actions pull-down. When the Extract Releases dialog box is displayed, enter the fields according to your build target environment, and then select **OK**. The extraction target directory should be NFS-exported on the specified host. Figure 101 on page 153 shows how the MVS builder extracted the current version of all files in the *MVS_Release_0*

release, using the CMVC client GUI. The files were extracted, with SCCS keywords expanded, to the product file tree, which is exported from its physical host, *bering*, to the developer's workstation host, *bengal*.

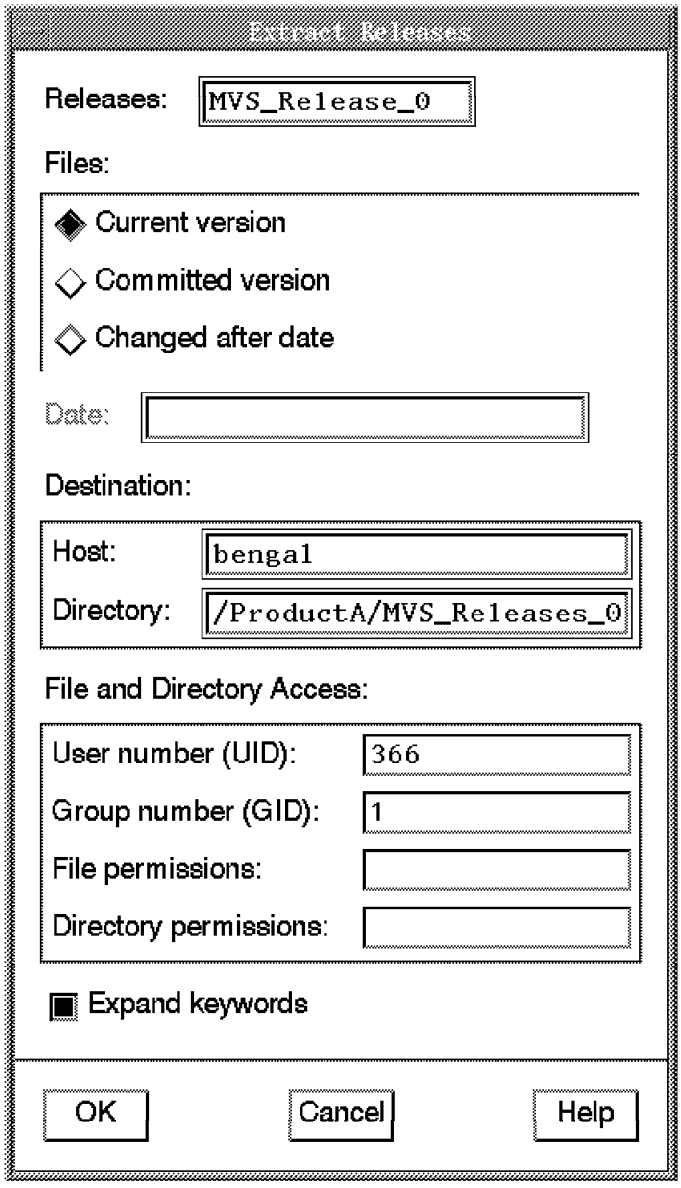

Figure 101. Release Extraction from the CMVC Client GUI

You can also use the CMVC command-line interface to extract releases and levels. Figure 102 on page 154 shows the CMVC **Release** and **Level** commands. These commands are not as convenient to use, but they can be inserted in a shell script to automate the production release builds of your application by team members who are not as skilled with CMVC as your build engineers. In the examples shown, the syntax of the CMVC **Release** command extracts the current version of the *MVS_Release_0* release, and the CMVC **Level** command extracts the full *0* level of that release.

```
Release -extract MVS_Release_0 -node bengal \
-root '/ad/ProductA/MVS_Release_0' -uid 366 -gui 1

Level -extract 0 -release MVS_Release_0 -node bengal \
-full -root '/ad/ProductA/MVS_Release_0' -uid 366 -gui 1
```

Figure 102. Extraction of Level and Release with CMVC Command

In our project, when the release or level extract successfully completed we uploaded the files from the /ad/ProductA/MVS_Release_0/source/cobol directory located on the AIX host *bengal* to the MVS mainframe on which we built our legacy version of the application.

5.10 Tester Tests the Application

In section 2.2.11, "Testing the Application" on page 35, the MVS tester was responsible for testing the *MVS_Release_0* release of the *productA* product on the MVS mainframe. Therefore, this team member had to log in to the mainframe, test the application there, and accept or reject the test record associated with the *prod_00001* defect and the *MVS_Release_0* release.

To accept a test record, display the CMVC - Test Records window, press both the Ctrl and O keys to display the Open List dialog box to enter a query that results in the relevant test record being displayed. Select the test record, and then select **Accept...** from the Actions pull-down. When the Accept Test Records dialog box is displayed, select **Yes**.

Figure 103 shows how the MVS tester accepted the test record for the *MVS_legacy* environment, which was associated with the *prod_00001* defect and the *MVS_Release_0* release, using the CMVC client GUI.

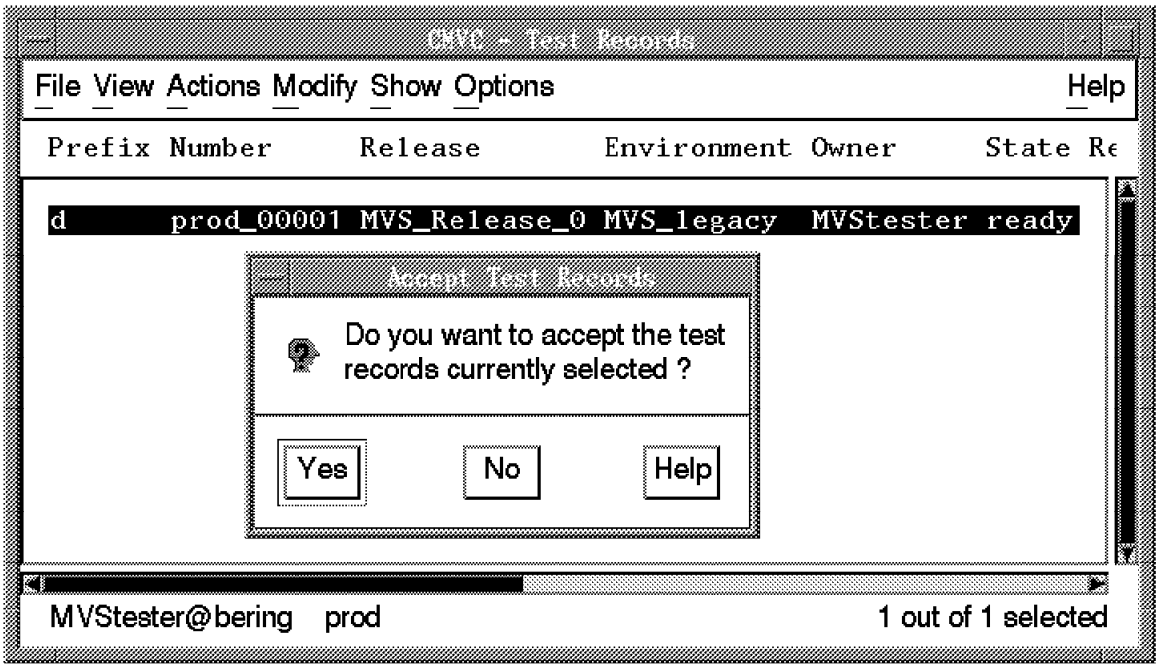

Figure 103. Accepting a Test Record from the CMVC Client GUI

You can also use the CMVC command-line interface to accept a test record. Figure 104 on page 155 shows the **Test** command. Specify the release by setting the *CMVC_RELEASE* environment variable prior to execution of the CMVC **Test -accept** command.

```
Test -accept -defect prod_00001 -environment MVS_legacy
```

Figure 104. Accepting a Test Record with the Test Command

Figure 105 shows the relationships among the various CMVC objects at this point in time. You can see that:

- The track state has moved from *test* to *complete*.
- The test record is now in *accept* state.
- The verification record state is now *ready*.
- The defect has changed from *working* to *verify*.

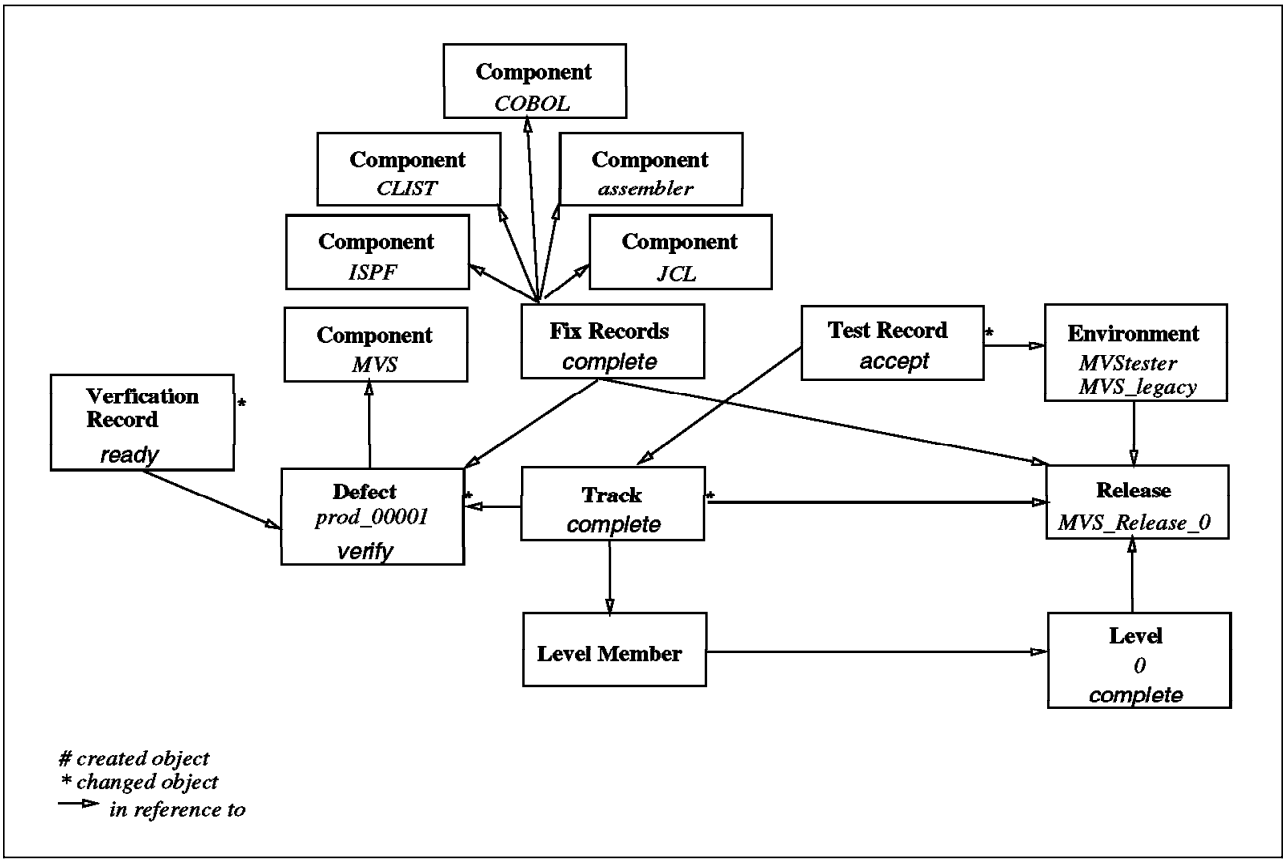

Figure 105. CMVC Object Status after Test Record Acceptance

5.11 Project Manager Ends the Migration

In the section 2.2.12, “Concluding the Migration” on page 36, the project manager, who was the defect originator, verified that the migration had been successful, by logging in to the mainframe and exercising the application. The project manager was then willing to close the *prod_0001* defect. Because the *MVS* component process includes the *verify* subprocess, the project manager did this by accepting a verification record.

To verify a defect or feature, display the CMVC - Verification Records window. Press both the Ctrl and O keys to display the Open List dialog box. Enter a query that results in the appropriate verification records. When the correct verification record is displayed in the CMVC - Verification Records window, highlight it, and then select **Accept** from the Actions pull-down. When the Accept Verification Records dialog box appears, select **Yes**. Figure 106 shows how the project manager verified that the *prod_00001* defect was resolved, using the CMVC client GUI.

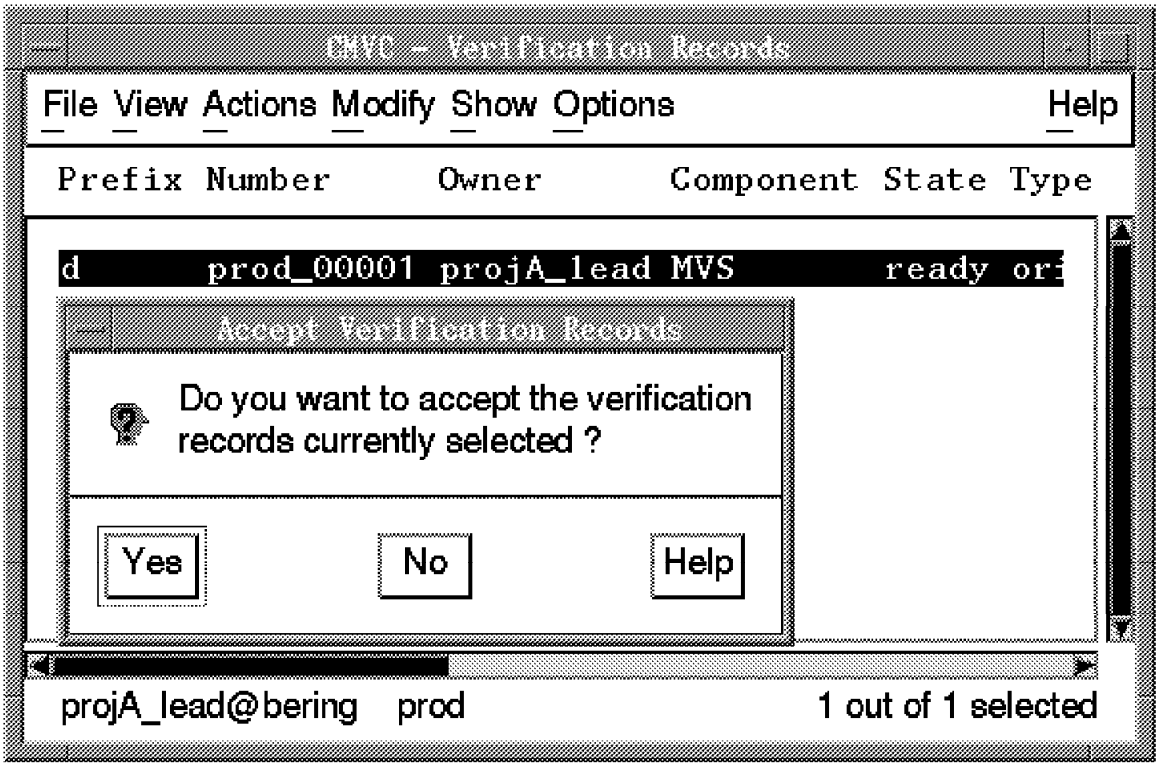

Figure 106. Verifying a Defect from the CMVC Client GUI

You can also use the CMVC command-line interface to accept a verification record. Figure 107 shows the **Verify** command.

```
Verify -accept -defect prod_00001
```

Figure 107. Accepting a Verification Record with the Verify Command

Figure 108 on page 157 shows the relationships among the various CMVC objects and the final states of each of them at the end of the problem tracking. You can see that the defect state has moved from *verify* to *closed* and the verification record is now in *accept* state.

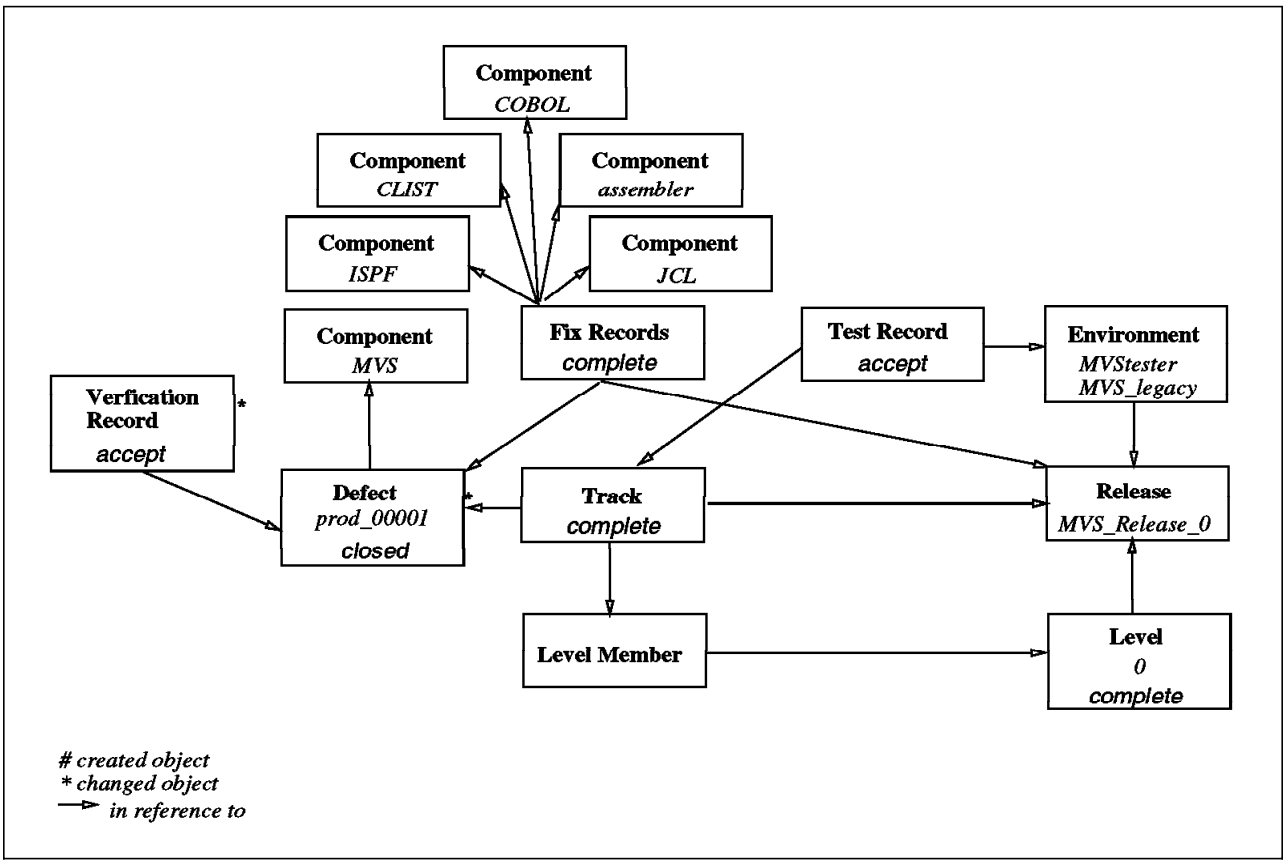

Figure 108. CMVC Object Status At the End of the Problem Tracking

Now, you have initialized the application baseline. You are ready to start collecting real defects and features and do controlled modifications to this baseline.

5.12 Project Manager Asks about Sharing Files with AIX Release

In section 2.2.12, “Concluding the Migration” on page 36, the manager asked about reusing the MVS source code in the AIX development.

You have two ways to reuse files or share files between releases. One way is to link the releases together by executing the CMVC *ReleaseLink* action. This is the best way, if you want to share all the files. The other way is to select the files you want to share, and then perform the CMVC *FileLink* action. To select the files, use the CMVC query capability from the Open File List dialog box and select all of the displayed files, and then execute the CMVC *FileLink* action. Or, if you know the file list, you can write a simple shell script using the **File** command.

Our project manager decided to link the *MVS_Release_0* release to the new *AIX_Release_1* release, because the team would reuse some of the files. This was not the right choice, because the team had to delete several irrelevant files from the new baseline (files related on the 3270 user interface). This problem is described and solved in D.2.2, “Deleting a File” on page 202.

Only the CMVC *ReleaseLink* action is described in this section. The project manager created the new release, *AIX_Release_1*, with the *maintenance* process,

and then opened the *prod_00002* defect. The AIX developer accepted it, so its state was *working*, and then created a track for it associated with the *AIX_Release_1* release, whose current state was *fix*.

To link one release to another, display the CMVC - Releases and press both the Ctrl and O keys to display the Open List dialog box. Enter the Release field, then press the Return key. When the correct release is displayed in the CMVC - Releases window, highlight it, and then select **Link...** from the Actions pull-down. When the Link Releases dialog box is displayed, enter the New release field. If you have the CMVC - Defects window already displayed, highlight the appropriate defect(s) or feature(s) and select **Import** on the CMVC - Releases window. If not, enter the Defects/Features field by keying the values in. Select **Committed version**, and then select **OK**. Notice that you have the choice between **Current version**, **Committed version**, and **Change after date** to identify which versions of the files should be linked. This choice enables you to create a new baseline reflecting the state of the files at any point in the past. Figure 109 shows how the AIX builder linked two releases to initialize the *AIX_Release_1* release, using the CVMC client GUI.

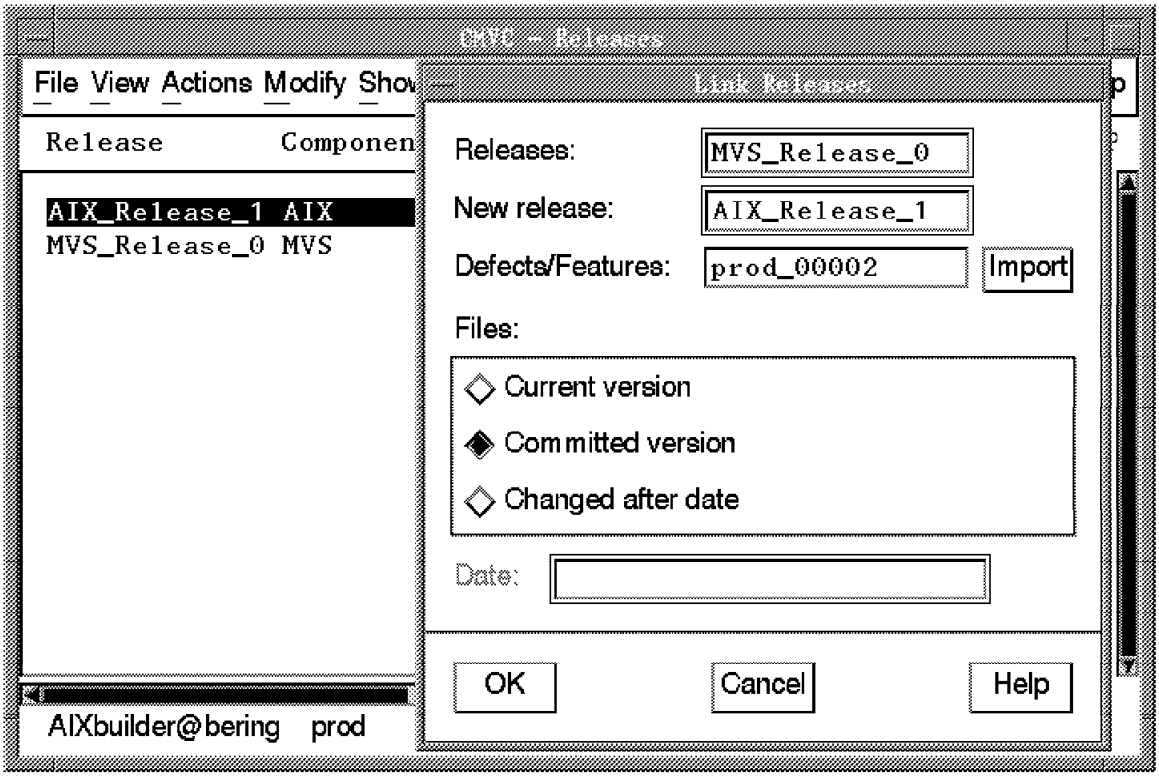

Figure 109. Linking Two Releases

You can link several releases to one release. To do that from the CMVC - Releases window, you should display the source releases, then select them by pressing the left mouse button without releasing it and dragging down the mouse pointer. After highlighting all source releases, select **Link...** from the Actions pull-down.

You can also use the CMVC command-line interface to link the releases. Figure 110 on page 159 shows two examples of the **Release** command. In the first example, the command links the *MVS_Release_0* release to the *MVS_Release_1* release, referencing the *prod_00001* defect. The second

example, shows the command linking three hypothetical releases to a fourth release, where problem tracking (defects and features) is not in use.

```
Release -link MVS_Release_0 to AIX_Release_1 \
-defect prod_00001 -committed

Release -link rel1 rel2 rel3 to targetRel
```

Figure 110. Two Examples of Linking Releases with the Release Command

Chapter 6. Installing CMVC and Supporting Databases

In this chapter we describe installation of the CMVC server, which requires previous installation of NetLS and ORACLE. Using the installation manuals for each product, we installed ORACLE first, NetLS second, and CMVC third.

6.1 ORACLE Installation, Initialization, and Shut Down

The *ORACLE for IBM RISC System/6000 Installation and User's Guide Version 6.0* provides a detailed checklist of considerations and actions to take in order to install ORACLE. The process involves running a shell script named **oracle.install** once, while logged in as *root*, and again while logged in as *oracle*.

6.1.1 ORACLE and Asynchronous I/O

One topic not covered in our ORACLE Version 6 documentation was the topic of asynchronous I/O. Asynchronous I/O is a feature of AIX 3.2 whose use was incorporated into later releases of ORACLE Version 6, and most releases of ORACLE Version 7. ORACLE advised us that the documentation for releases later than ORACLE Version 6.0.36.3.2 addresses this issue in Release Bulletins that accompany the distribution medium. It is also more formally discussed in *"*Key to Oracle on AIX 3.2*"* in *Oracle 7 for IBM on the RISC System/6000 Installation and Configuration Guide*. Asynchronous I/O is implemented as a pseudo device driver (kernel extension) in AIX 3.2. Use of asynchronous I/O by-products, such as database engines, improves performance with high user volumes.

Before installing ORACLE, you should verify that asynchronous I/O is configured and will also be configured on the next system reboot. If this is not the case, you will get unresolved external references and a load failure when the ORACLE program itself is loaded during the installation. Asynchronous I/O may already be configured on your host but not configured to survive a system boot. If this is the case, you will experience a trouble-free installation, but don't be surprised at the next system startup, when asynchronous I/O is no longer configured.

To configure asynchronous I/O, select **Configure Defined Asynchronous I/O** from the Asynchronous I/O menu in SMIT (fast path **smit aio**), or execute **mkdev -l aio0**. You must also explicitly set the field labeled "STATE to be configured at system restart," on the SMIT Change/Show Characteristics of Asynchronous I/O screen, to **available**, or execute the **chdev -l aio0** command.

6.1.2 ORACLE User ID, Group, and File System

For ORACLE we created the *oracle* UNIX login name, the */oracle* file system, and the *dba* and *oracle* UNIX login groups. We placed the *root* and *oracle* UNIX login names in the *dba* UNIX login group. The *oracle* UNIX login name and *dba* login group own the files in the */oracle* file system.

6.1.3 Starting ORACLE

To support initialization of ORACLE on a system startup, ORACLE advises you to modify the /etc/inittab file, by adding the following line:

```
oracle:2:wait:/bin/su oracle -c /oracle/bin/dbstart
```

When the ORACLE server is properly initialized there will be a set of background processes running for each database instance. The names of these processes include the unique SID for the database instance. Where *XXX* below is replaced by the SID, these processes are named:

- **ora_XXX_dbwr**
- **ora_XXX_lgwr**
- **ora_XXX_smon**
- **ora_XXX_pmon**.

6.1.4 Stopping ORACLE

ORACLE also advised against editing the /etc/shutdown file to call their **dbshut** command automatically whenever *root* brings the system down. We called Oracle to determine the reasoning behind this advice and were told that the **dbshut** command might fail to complete under certain, not too unusual, circumstances. One such circumstance might be that CMVC has not yet been shut down. If this happened, the entire system shutdown would likewise hang. Instead ORACLE recommends that you log in as the *oracle* user and execute the **dbshut** command interactively. This one command will shut down all ORACLE databases listed in the /etc/oratab file that have a "Y" in the third field. Following this procedure, if you encounter troubles bringing ORACLE down, you can resolve them before continuing to bring AIX down.

Another interesting fact, which we discovered the first time we shut down ORACLE, is that although the *root* UNIX login name is a member of the *dba* UNIX login group and therefore should be able to execute ORACLE commands, such as **dbshut**, it will not necessarily have all the environment variables set up as the *oracle* UNIX login name has. We later ran **dbshut** as *root*, and found that ORACLE refused to start up, indicating that it could not start because it was already running. This anomalous situation occurred because a path name for a file did not resolve properly, when **dbshut** ran in the *root* environment, and the file was not removed, as it should have been. The existence of this file, one defining the database's system global area (SGA), causes ORACLE to assume it is already running.

Since a system can go down in a less than orderly manner, you might still occasionally find that ORACLE refuses to start on a system startup. If this happens, ORACLE suggests you execute the command shown in Figure 111.

```
su - oracle -c "sqldba command=startup force open"
```

Figure 111. Forcing ORACLE to Start Up

This command starts up any database instance defined in the /etc/oratab file that has a "Y" in the third field. This command should not only remove the lingering SGA file but also ensure that ORACLE database recovery procedures are executed.

ORACLE will not shut down if CMVC daemons are currently executing. Therefore, you want to bring CMVC down first, then ORACLE, and finally NetLS.

6.1.5 ORACLE SID and CMVC Family Relationship

The CMVC documentation describes very little about the database products, on the assumption that they are best documented by their vendor. The ORACLE documentation, naturally says nothing at all about CMVC. This left it to us to try to map the conceptual elements of one paradigm to the other. When we decided to create our second CMVC family, it was unclear what the relationship was between a CMVC family and an ORACLE database. The CMVC documentation does indicate that to create your second and subsequent families, you should follow all the steps used to created the first family. When you run the **mkfamily** command, you must use a new family name.

CMVC technical advisors tell us that we should be able to use the same SID to create multiple CMVC families, but we did not know this at the time. So we ran the **oracle.install** script again and created a unique ORACLE database instance for each CMVC family. To minimize our confusion, we named the SIDs identically to the family, which they would represent (this being allowable because our family names are short enough). However, to be consistent with what appeared to be an ORACLE convention, we spelled the names in uppercase letters. This approach worked well for us, but there may be implications of which we are unaware in using a unique SID for each family.

6.2 NetLS Installation and Initialization

CMVC Version 2 is one of the first IBM AIX products to come out enabled for NetLS license control. In time many IBM LPPs and OPPs will make use of this licensing mechanism. Since this is a fairly new concept, we discuss this topic in the sections that follow.

6.2.1 License Serving Concepts

NetLS enables you to buy a license based on the maximum number of concurrent users you want to support at the server at any one time. You can install a NetLS license server on some host in the network and advise it of the number of concurrent licenses authorized for a given LPP on a specific host in the network. The LPP can be invoked from any host on the network. It requests a NetLS token (equivalent to one licensed user), which it can keep for a predefined period of time. When the time expires, the license server can distribute the token again to the next requester. If no tokens are available, the LPP can wait for one to become free or to return immediately; it depends on how the LPP is programmed to interact with the license server. The CMVC clients return a message indicating that a token was unavailable; they do not wait for a token. If a token is available the CMVC clients connect to the CMVC server. An LPP can be written to allow the client portion to execute without having a token unless or until it needs access to the server. You can also indicate that you need the token for a longer of time than the predefined minimum period for which the token is valid.

6.2.2 NetLS Password and CMVC Installation

You must obtain host identification data, by running the **ls_targetid** on the hosts that are designated NetLS license servers. On AIX, this data looks very much like the output of the **uname** command. You give this information to IBM, and you receive in return a vendor ID, product ID, and two passwords. The vendor password is used with the vendor ID to register the vendor in the NetLS database. The product password encodes information, such as the product, number of licenses, their type, and duration. These IDs and passwords are used to register the licences with the NetLS server software after the LPP is installed, but before you can use it. Additional licenses can be purchased and easily identified to NetLS by means of updated passwords.

Each password returned to you by IBM encodes the number of concurrent licenses you have purchased that can be issued from a particular host for a particular LPP. You may want to have more than one NetLS license server. If so, you will have to decide how many of the total number of licenses you have purchased will be installed on each NetLS license server, before contacting IBM to obtain your passwords.

NetLS requires the NCS daemons to be running. These daemons are **llbd**, the Local Location Broker, and **glbd**, the Global Location Broker. The NetLS daemon is **netlsd**. To ensure that NetLS-dependent clients and servers can operate on a system startup, entries are made for you in the /etc/inittab file by the installation procedures to start up the NetLS daemons and to execute the **/etc/rc.ncs** script. The **/etc/rc.ncs** script starts up the **llbd** and **glbd** daemons. We found that if we asked for three instances of the NetLS server to run, three identical sets of entries were made in the /etc/inittab file. We suspect that one entry would be sufficient. This entry is shown in Figure 112.

```
rcncs:2:wait:sh /etc/rc.ncs > /dev/console 2>&1
netlsd:2:wait:sh /etc/rc.netls >/dev/console 2>&1
```

Figure 112. Entries Made to /etc/inittab File by NetLS Installation

NetLS, and the daemons associated with it, must be brought down when someone logged in as the *root* login name, deliberately shuts down the AIX system.

Refer to Appendix G, “Appendix: Setting Up NetLS” on page 213 for the procedure to install NetLS in a simple environment.

6.3 CMVC Installation and Initialization

The sections that follow describe actions taken while installing and initializing the CMVC server. For each CMVC family, we created a user named the same as the family itself. The families were named *dev* and *prod*. Likewise, we created a file system for each family, but since the file system named */dev* already existed, we named them */development* and */production*. All files in those file systems are owned by the family UNIX login name and the *system* UNIX login group. We identified another AIX login name with which to create the CMVC families. CMVC creates an initial CMVC user ID and identifies it with the AIX login name that executes the commands to create a CMVC family. This CMVC user is the default CMVC superuser for that family. We chose *lrconas* as the AIX login name for this purpose and named the CMVC user identically.

We also modified the /etc/inittab file to support initialization of CMVC on a system startup. To start up CMVC, we required one entry to start up the server daemons, and another to start up the notification daemon. These entries are shown in Figure 113 on page 165.

```
cmvc1:2:wait:/bin/su - dev -c "/usr/lpp/cmvc/bin/cmvcd dev 3"
cmvc2:2:wait:/bin/su - dev -c "/usr/lpp/cmvc/bin/notifyd"
cmvc3:2:wait:/bin/su - prod -c "/usr/lpp/cmvc/bin/cmvcd prod 3"
cmvc4:2:wait:/bin/su - prod -c "/usr/lpp/cmvc/bin/notifyd"
```

Figure 113. Entries Added to the /etc/inittab File.

Since we could not shut down ORACLE automatically, by means of the **/etc/shutdown** command (shell script command), we could not shut down CMVC that way either, for the latter must precede the former. To shut down CMVC, we had to issue the commands shown in Figure 114. Each command had to be issued while we were logged in as the appropriate CMVC family UNIX login name.

```
/usr/lpp/cmvc/samples/stopCMVC dev
/usr/lpp/cmvc/samples/stopCMVC prod
```

Figure 114. Commands to Shut Down CMVC for Our Families

Appendix A. Implementation of ISO 9001 Using CMVC

This appendix provides a brief introduction to ISO 9000 and describes how the IBM CMVC tool can be utilized to meet some key ISO 9001 elements.

The software engineering industry is the fastest growing industry in the last half of the century. Throughout the industry, software development organizations are struggling with the challenges of reducing costs, increasing productivity and improving quality. Towards these efforts, quality management of software is essential. One way to establish a quality management system is to provide guidance for software quality assurance. Such guidance is found in the International Standard Organization (ISO) 9000 series of quality standards: ISO 9001, ISO 9002, ISO 9003.

ISO 9000 compliance is of key importance to organizations if they are to survive the fierce competition of the 1990s and beyond. With the introduction of ISO 9000, the software engineering industry has experienced a shift towards implementing techniques and processes aimed at developing processes that are well defined and repeatable. Adoption of these proven techniques and processes allows an organization to improve the overall process of:

- Creating world class software
- Maintaining a dynamic, responsive, and innovative environment
- Attaining a high return on investment through the pursuit of total customer satisfaction.

Software tools can assist in improving an organization′s management system and meeting ISO 9000 compliance. The IBM Configuration Management Version Control tool is an effective tool that simplifies the organization and management of diverse tasks involved in software development so that you can improve your entire product development process and it can be used to comply to some key elements of ISO 9001.[3]

A.1 ISO 9000

To facilitate the standardization of the many aspects of quality, the International Organization for Standardization (ISO) has developed a set of international standards for quality systems which are known as the ISO 9000 Series of Quality Standards. These standards apply to all organizations producing a product or service and are being accepted world-wide. An indication of their acceptance and significance in the software engineering industry, in particular, is reflected in Hubner′s words, “To obtain an ISO 9000 certification has become a business necessity in Europe” [1]. It is only a matter of time before ISO 9000 certification becomes a business necessity in North America and the Orient.

Three key ISO 9000 standards are:

- ISO 9001, Quality systems - Model for quality assurance in design/development production, installation and servicing.

[3] IBM software development Labs in Toronto and Austin use IBM CMVC to manage and control the software development process and implement certain elements of the ISO 9001 standard.

- ISO 9002, Quality systems - Model for quality assurance in production and installation.
- ISO 9003, Quality systems - Model for quality assurance in final inspection and test.

ISO 9001 provides the most comprehensive requirements for a software quality system where a contractual agreement between two parties must demonstrate the supplier's ability to design and supply a product or service.

Recognizing the peculiarities of the software industry, ISO 9000-3, a guideline for the application of ISO 9001 to the development, distribution and maintenance of software, has been released. Configuration management is a key element in this ISO 9000-3 guideline for software.

A.2 CMVC and ISO 9001

Configuration management provides a mechanism for identifying, controlling and tracking the versions of each software item. In many cases, multiple versions of software items are in use and must be maintained and controlled. Configuration management itself is not an element of the ISO 9001 standard (only an element of ISO 9000-3 guidelines); however, the following elements of ISO 9001 depend on configuration management:

- Document Control
- Design Control
- Product Identification and Traceability
- Inspection and Test Status
- Control of Nonconforming Product
- Internal Quality Audits.

Only certain aspects of Document Control, Design Control, Control of Nonconforming Product and Internal Quality Audits are addressed by Configuration Management. Product Identification and Traceability and Inspection and Test Status are fully addressed by it.

The remaining sections describe these ISO 9001 elements and highlight how CMVC can support them.

A.2.1 Document Control

Document Control covers:

- The determination of those documents that should be subject to document control procedures
- The approval and issuance of document control procedures
- The change procedures including withdrawal and, as appropriate, release. [3]

The aspects of Document Control that can be successfully addressed by CMVC are:

- Documents must be accessible to a group of people with predetermined interest and authority.

- The changes to the content of a document under document control have to be reviewed by a prespecified group of reviewers and the final version of the document has to be approved by them.
- All the users of the document have to be notified of the changes to it.
- Once the new document is finalized and becomes the most recent working document, provisions must be taken to prevent users from using a back-level version of the same document.

Files and documents pertaining to a particular project reside in one more CMVC components (where each CMVC component is dedicated to a specific department and/or all documents related to a particular project).

An access list and a notification list is associated with each CMVC component. The type of access each user has to the documents stored in CMVC depends on the user's role in the development team. The type of notification each user has depends on the interest or need to be informed of changes to documents in the CMVC environment. Access authority and notification subscription is assigned to a user by the component owner or by someone who has the authority to grant other users access and notification to a specific component. When a document is updated, the owner of the managing component where the document resides and all users who have subscribed to being informed of document updates receive mail notifying them of the update.

When the CMVC approval process is activated, approval must be given for proposed changes before work can begin on the implementation of a change. Approvers specified for each release need to review the information recorded in the defect or feature and evaluate the proposed changes to the release in relation to other project considerations. A CMVC approval record is created for each approver. Each approver indicates his or her evaluation of the changes and can optionally append comments to the defect or feature to explain the rationale for his or her decision. File changes for that defect or feature in that release cannot be checked in to the CMVC server until all approvers accept the proposed changes. [4]

A.2.2 Version Control in ISO 9001

Certain aspects of Design Control, Product Identification and Traceability, Inspection and Test Status, and Control of Nonconforming Product deal with the issue of *version control*. Activities are versioning documents and source modules that compose a product; identifying the various versions of the documents and source modules and the reasons why changes were made from one version to the next; inspecting and testing the content of each version and recording the status of this outcome; and finally, controlling nonconforming products and identifying the version of documents and source modules that reflect the nonconformance.

Version control, by definition, is the storage of multiple versions of a single file along with information about each version. [4] CMVC provides for version control and enhances this basic function with an extra layer of traceability so that each version is cross-referenced to a reported defect or a suggested enhancement.

The following sections highlight the specific sections of each of the ISO 9001 elements that emphasize the need for version control mechanisms in an organization.

A.2.2.1 Design Control

The ISO 9001 element of Design Control states that “the supplier shall establish and maintain procedures to control and verify the design of the product in order to ensure that the specified requirements are met.” One of the items that define this element of Design Control is the Control of the Design Changes, which is designed as:

> “The supplier shall establish and maintain procedures for the identification, documentation and appropriate review and approval of all changes and modifications.” [2]

CMVC can be used to control and verify the design of a product in a number of ways. First, the design specification documents themselves can be stored in CMVC. Modifications to the content of the design specifications can be controlled from both an access and an update perspective. Development teams can make use of the problem-tracking feature of CMVC to track and control the changes to design specifications and to ensure that all changes have been well documented, justified, and reviewed for appropriateness and applicability. CMVC′s design, size and review process for reported defects and suggested features provides for the identification, documentation and appropriate review of all changes and modifications. CMVC′s tracking process allows development teams to cross reference changes to design documents or source modules to the reported defects and features. It also provides an additional layer of control to those teams seeking an approval checkpoint prior to versioning the documents and a review of the changes after making the modifications but prior to committing them.

When source modules for a product are managed by CMVC, development teams can ensure that the changes made to the product are consistent with its design by using the integrated problem tracking and change control feature of CMVC. Attributes of reported defects and features can be used to cross-reference design documents with the proposed source module changes to the product. CMVC′s ability to track required changes in all project deliverables ensures that appropriate updates are made to design documents, source modules, test cases and end-user documentation for each reported defect and suggested enhancement of the product.

When the time comes to release a product, development teams can benefit from CMVC′s ability to maintain multiple versions or variants of a product. Enhancements for the next release can be incorporated into design documents and source modules without disrupting the integrity of the products that have been distributed.

A.2.2.2 Product Identification and Traceability

The ISO 9001 element applicable to version control for Product Identification and Traceability states:

> “here appropriate the supplier shall establish and maintain procedures for identifying the product from applicable drawings, specifications or other documents, during all stages of production, delivery and installation. Where and to the extent that, traceability is a specified requirement, individual product or batches shall have a unique identification. This identification shall be recorded.”[2]

When CMVC is used as the configuration management and version control tool for software development activities, several levels of identification and traceability are available.

Each version of a document or source module is identified by a version number. The combination of this version number, as well as the document or source module name (CMVC file name), and the product name that the document or source module is associated with (CMVC release name), uniquely identify a file in CMVC.

As previously discussed, changes to documents and source modules can be cross-referenced to CMVC defects and features. Users can then query the history of changes and identify the content of each version of a document or source module and the reason why it has changed over time. Alternatively, users can query the details of the defects and features to determine the document or source modules that were changed as a result of fixing a reported problem or implementing a suggested feature.

CMVC records the time and date of the changes to documents or source modules as well as the user who makes each change. This information provides additional traceability and can be queried at any time.

A.2.2.3 Inspection and Test Status

The ISO 9001 element applicable to version control for Inspection and Test Status states:

> **"...**The identification of inspection and test status shall be maintained**,** as necessary**,** throughout production and installation of the product to ensure that only product that has passed the required inspections and tests is dispatched**,** used or installed**.** Records shall identify the inspection authority responsible for the release of conforming product**."**[2]

The CMVC problem tracking mechanism allows development teams to establish two types of testing procedures. When development teams use the tracking mechanism, they can define test environments and testers for each release of a product. As defects are fixed and features are implemented, CMVC activates test records for each of the test environments and testers indicating when the changes made to documents and source modules have been committed in the product. Testers are notified when their test records are ready to be marked. By marking a test record with an accept, reject, or abstain status, a tester relays information to the development team as to the status of the change. When test records are rejected, additional defects or features can be opened to track the nonconformances, and attributes in both the original and the new defects or features can be used to cross-reference problems of a similar nature.

Another type of testing occurs at the end of the CMVC defect or feature lifecycle. Once applicable documents or source modules have been changed and committed into the various products, the originator of the defect or feature has the opportunity to verify that the resolution of the problem or the implementation of the suggestion has been accomplished to his or her satisfaction. Originators record their satisfaction of the outcome on verification records.

The status of test and verification records, the owner of test and verification records, and the timestamp of when each record was last updated is maintained

in CMVC. The reporting mechanism allows users to query the status of these records at any time.

A.2.2.4 Control of Nonconforming Product

The ISO 9001 element applicable to version control for the Control of Nonconforming Product states:

> **"....**Control shall provide for identification, documentation, evaluation, segregation (when practical), disposition of nonconforming product and for notification to the functions concerned.**"** [2]

Nonconformances with respect to products managed by CMVC are identified by opening a CMVC defect or a CMVC feature. Once the nonconformances are identified, the development teams can evaluate the validity of the nonconformance, and can return the defect or feature as invalid, as a documented deviation, or as a nonconformance that has already been addressed by another defect or feature report. Alternatively, the development team can decide to accept responsibility for the nonconformance and schedule its resolution in the appropriate product releases. In either case, all users who have subscribed to defect and feature reports and state changes will be kept informed.

When nonconformances are received for products that have been released to customers, development teams can resolve the nonconformance and reissue a product update. Development teams can make use of the attributes associated with product levels to describe the status of the package. For instance, a product level may be shipped and then subsequently replaced with a newer version that includes fixes for nonconformances.

A.2.3 Internal Quality Audits

> **".....**The audits and follow-up actions shall be carried out in accordance with documented procedures. The results of the audits shall be documented and brought to the attention of the personnel having responsibility in the area audited. The management personnel responsible for the area shall take timely corrective action on the deficiencies found by the audit.**"** [2]

An example of how IBM addressed this element using CMVC, is the internal audits to check the level of compliance of the various departments in the IBM PRGS Toronto Lab. Internal audits were conducted and nonconformances were issued and monitored until a satisfactory corrective action plan was put in place and successfully implemented.

In a specific area, a component dedicated to ISO 9001 was created and an owner was assigned to it. Traditionally, the owner of this component is the ISO focal point for the area. Each department in the area had its own component where all the documents and processes were stored.

The internal audit group would open defects (Nonconformances) against the area component. The component owner would then route the nonconformances to the corresponding department component. The departments are responsible for creating their own corrective action plan. Each corrective action was appended to the nonconformance in CMVC, and all the interested parties were notified that remarks were appended to the specific nonconformance.

The remarks added to each of the nonconformances that described the corrective action were retrieved via CMVC and then forwarded to the internal audit group for review. This group created the actual corrective action plan for each department and hence for the area as a whole.

A.3 Conclusion

To make software quality a reality and to comply to the ISO 9001 standard, CMVC can successfully be used for:

- Design Control
- Document Control
- Product Identification and Traceability
- Inspection and Test Status
- Control of Nonconforming Product
- Internal Quality Audits.

A.4 Brief Description of ISO 9000-3

This section describes the five elements listed in the section A.2, “CMVC and ISO 9001” on page 168, as well as the 9000-3 element on Configuration Management.

A.4.1 Configuration Management

Configuration Management should:

- Uniquely identify the versions of each software item
- Identify the versions of each software item that together constitute a specific version of a complete product
- Identify the build status of software products in development or the status of those software products delivered and installed
- Control simultaneous updating of a given software item by more than one person
- Provide coordination for the updating of multiple products in one or more locations as required
- Identify and track all actions and changes resulting from a change request, from initiation through to release.

A.4.1.1 Configuration Identification and Traceability

To establish and maintain procedures for identifying software items during all phases, starting from specification through development, replication and delivery, each individual software item should have a unique identification.

There should be provisions to uniquely identify the following items for each version of the software:

- The functional and technical specifications
- All development tools which affect the functional and technical specifications
- All interfaces to other software and/or hardware items
- All documents and computer files related to the software item.

The identification of a software item should be handled in such a way that the relationship between the item and the contract requirements can be demonstrated.

For released products, there should be procedures to facilitate traceability of the software item or product.

A.4.1.2 Change Control

Procedures should be in place to identify, review and authorize any changes to the software items under the control of configuration management. All changes to software items should be carried out in accordance with these procedures.

Before a change is accepted, its validity should be confirmed and the effects on other items should be identified and examined.

Methods to notify those concerned of the changes and to show the traceability between changes and modified parts of software items should be provided.

A.4.1.3 Configuration Status Report

The supplier should establish and maintain procedures to record, manage and report on the status of software items, of change requests and of the implementation of approved changes.

A.4.2 Design Control

The supplier should establish and maintain procedures to control and verify the design of the product in order to ensure that the specified requirements are met.

A.4.3 Document Control

Procedures should be established and maintained to control all documents that relate to the contents of this part of ISO 9000. They cover:

1. The determination of those documents that should be subject to the document control procedures.
2. The approval and issuance of document control procedures.

   All documents should be reviewed and approved by authorized personnel prior to issue. Procedures should exist to ensure that the pertinent issues of appropriate documents are available at appropriate locations where operations essential to the effective functioning of the quality system are performed. Obsolete documents should be promptly removed from appropriate points of issue or use.

   Where use is made of computer files, special attention should be paid to appropriate approval, access, distribution and archiving procedures.
3. The change procedures including withdrawal and, as appropriate, release.

A.5 References

1. Hubner, A. *ISO 9000 Implementation in Germany* , ***LOGON***, Volume 4, Number 4, September 1992 (IBM Internal Use)
2. *International Standard: ISO 9001*, Reference Number ISO 9001:1987(E)
3. *International Standard: ISO 9000-3*, Reference Number ISO 9000-3:1991(E)
4. *IBM CMVC Concepts*, SC09-1633-00, IBM Corporation, 1993.

Appendix B. Monitoring and Enhancing the Quality of Software with CMVC

Assessing the quality of a software product is a task that all software organizations eventually struggle with. This appendix briefly outlines software quality and the fundamentals of Software Reliability, and demonstrates how the IBM Configuration Management Version Control tool has been used, not only for managing the software development process, but also for providing data that can be used to predict and improve the quality of the software products under development.

B.1 Introduction

The software engineering industry is the fastest growing industry in the last half of this century. Throughout the industry, software-development organizations are struggling with challenges such as reducing costs, increasing productivity and improving quality. Without the appropriate focus on quality, the costs of software testing and maintenance will increase. The productivity of the whole software development team will be lower than it could have been if quality was *built* into, rather than *tested* into, the product.

While the manufacturing sector has achieved significant improvements in hardware reliability and has reduced manufacturing costs by using mathematical and statistical techniques, there is still no complete scientific and quantitative method of assessing software quality. Neither software testing nor proving can guarantee complete confidence in the correctness of a program. What is needed is a metric to reflect the level of program correctness, something that can be used to plan and control the additional resources needed to enhance software quality. Software Reliability is a good example of a quantifiable metric commonly used for assessing software technologies.

B.2 Software Quality

Intensive and well-planned testing will undoubtedly reveal a high percentage of the errors in a software product. Chances are, however, that some errors will go unnoticed.

Software companies have recently shifted their focus toward implementing methods for improving software quality. Despite all of the attention, there is still no unique definition of software quality. Software quality takes on different meanings depending on the individual perspective. For some, software quality is a product that meets specifications. For others, software quality is the absence of defects in the software product or the adherence to the ISO 9000 quality standards. Dunn [5] suggests that software quality is a set of attributes in the software product such as reliability, usability and usefulness, maintainability and salability.

Market Driven Quality (MDQ), with an emphasis on customer focus, six sigma $(6\sigma)^4$ focus on defect levels, and Malcolm Baldrige awards are common

elements among corporate strategies designed to achieve the objective of high quality products.

To help standardize the many aspects of quality, the International Organization for Standardization has published a set of quality assurance standards called ISO 9000. ISO 9000 consists of a 20-requirement quality system ranging from management responsibility to statistical techniques. An indication of its importance in the software engineering industry is reflected in Hubner′s declaration: ″To obtain an ISO 9000 certification has become a business necessity in Europe″ [2]. It is only a matter of time before ISO 9000 certification becomes a business necessity in North America and the Orient.

The IBM definition of software quality is an enriched version of Dunn′s definition. IBM uses the CUPRIMDS acronym to describe a set of attributes (capability, usability, performance, reliability, installability, maintainability, documentation and serviceability) that conforms to its definition of software quality.

In IBM, the quality effort has been directed toward MDQ and 6σ defect levels. At the IBM Programming Systems (PRGS) Toronto Lab, the strategy is to use the 6σ defect count as a criterion for deciding whether a product should be released to the market. Software configuration management systems are being used to help monitor the progress towards this goal.

B.2.1 Process Maturity Levels

An organization can be classified into a particular level of process maturity depending on the characteristics that it displays during the development cycle of a product. There are five levels of process maturity [3]:

1. **Initial**: Until the process is under statistical control, orderly progress in process improvement is not possible. While there are many degrees of statistical control, the first step is to achieve rudimentary predictability of schedules and costs.

   A large number of organizations have no formalized procedures for cost estimates, project plans or change control. While some organizations have formal procedures for planning and tracking their work, most have no mechanism to ensure that the procedures are used. These organizations portray the initial level of process maturity.

   Having configuration management and change control mechanisms in place makes it easier and more affordable for an organization to achieve the second level (repeatable) of process maturity.

2. **Repeatable**: The organization has achieved a stable process with a repeatable level of statistical control. It does not have rigorous project management of commitment, costs, schedules and changes.

3. **Defined**: The organization has defined a process as a basis for consistent implementation and better understanding and everyone in the organization is following the process.

---

[4] Sigma (σ) is the mathematical notation for the standard deviation. In terms of software, 6σ translates into a defect count of approximately three to four defects per million lines of code or a quality level of 99.9999998%. The typical software product produced by the average software organization in the United States has three to four defects per thousand lines of code; a quality level of 99.98% or 4σ.

4. **Managed**: The organization has initiated comprehensive process measurements and analysis. The most significant quality improvements begin.
5. **Optimized**: The organization now has a foundation for continuing improvement and optimization of the process.

B.2.2 Software Reliability

The underlying concepts of quality control are closely related to a series of mathematical and statistical models. These models are used to monitor and predict the quality of software products. By carefully collecting data on failures as they occur and subsequently deducing the type of model that describes these failures, the project manager can estimate, with a reasonable degree of accuracy, the number of defects, or possible failures, remaining in the software product. Having access to this kind of information helps the decision maker decide whether a product is ready for release to the market.

Many mathematical and statistical models predict the quality of software; among them are: Markov Models, Nonhomogeneous Poisson Process Models (NHPP) [6], Static Models, and, Bayesian Analysis and Modelling. (A discussion of these models can be found in the section B.6, “Discussion on Certain Reliability Models” on page 180).

An important aspect of Software Reliability is acquiring actual and valid data. Identification of data sources and the selection of a data-collection mechanism is not a trivial task. Data is traditionally defined as ″the number of failures of the software product under test or usage as they occur over time″. Failure, in this case, is defined as any unexpected behavior of the software product. However, this definition excludes changes made because of suggestions aimed at improving functionality or any other type of enhancement to the software product. A failure is also classified according to a severity level. It may be a fatal defect causing the software application to fail (crash), or it may be a mere inconsistency in the way a particular aspect of the application ″looks″. Distinguishing the various categories in the data is important to the overall quality assessment of the software product.

B.3 CMVC and Quality Control

The IBM PRGS Toronto Lab uses CMVC to manage and control its software-development process and monitor the resolution progress of identified defects for several UNIX-based software products under development. The development teams in the Application Development Technology Center of the Lab are supported by a quality focus group whose role is to monitor the progress and defect statistics of the various development teams and identify the Software Reliability model that best fits the data.

CMVC helps the quality focus group do just that. It tracks reported problems and retains information about the life cycle of each in a relational SQL database residing on the CMVC server.

During the development life cycle of a product[5], testers, developers and general users report defects as they encounter them. Each defect is opened against a specific CMVC component, a component that represents the source of the problem. All interested parties are notified of the newly reported defect as soon as it enters the system. Interested parties are made aware of changes related to the product, and they can express their views on the reported problem. The owner of the component automatically becomes the owner of the newly created defect and is responsible for analyzing the validity of the reported problem. If the defect is valid, it is accepted for resolution and becomes part of the defect statistic. If it is not a valid defect, it is returned to its originator.

The quality focus group retrieved the status of defects and features from CMVC at regular intervals. Data regarding the number of defects in the open and closed states was collected on a regular basis and was input into a Lotus 1-2-3** spreadsheet. Graphs of the data were generated for weekly management meetings. The graphs formed the basis of discussion on each development project from both a quality and a schedule perspective.

An example of a graph depicting the information collected is shown in Figure 115. The information provided through the graph addresses the following:

1. The number of valid defects in the open state versus time (where time is measured in weeks).
2. The cumulative number of defects in the closed state versus time (where time is measured in weeks).

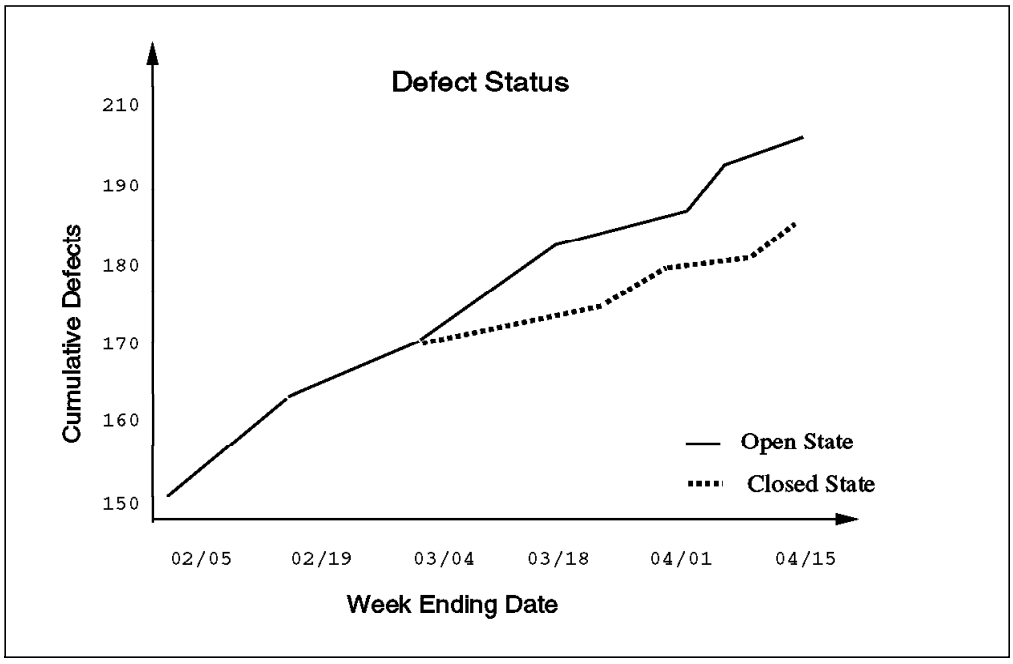

Figure 115. Defect Status over the Time

The difference between the first and the second curves, as they propagate over time, describes whether the arrival rate of valid defects is greater than or less than the rate at which the defects are being fixed. This information keeps managers informed about the current status of the projects, but does not allow

[5] The various phases of the development cycle are described in the section B.7, "Classification and Definition of Test Phases" on page 181.

them to predict the long run trend of the data (and hence the long run status of the projects).

For predictions, the same data was input into the mathematical models that are described in the section B.6, “Discussion on Certain Reliability Models” on page 180. The quality focus group could predict fairly accurately the behavior of the system over time. Therefore, they could determine when the products would have *n*-many latent defects as shown in Figure 116. If *n* was an acceptable figure, based on the 6σ criterion, then the time corresponding to *n* was the earliest time possible for releasing the product.

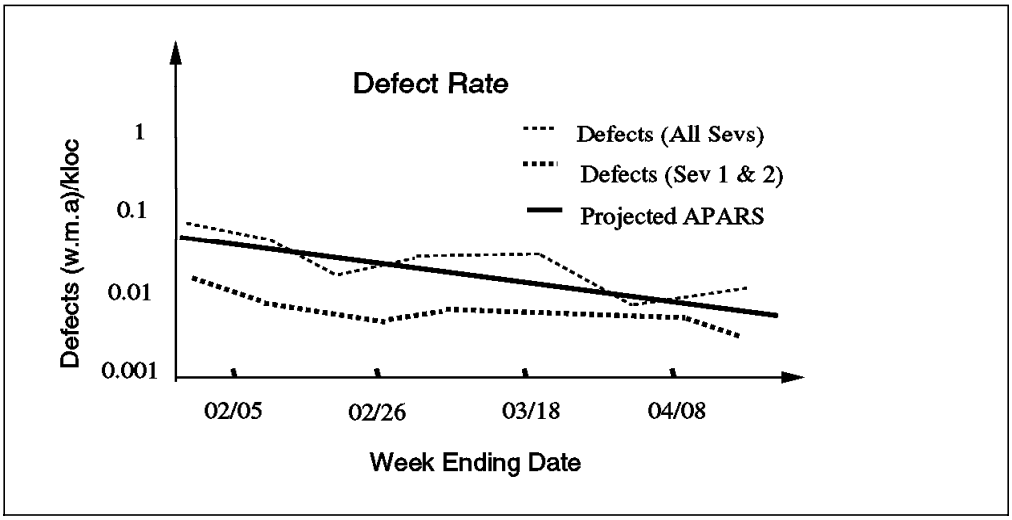

Figure 116. Predicting the Number of Latent Defects

As defects were opened during the development cycle, the quality focus group was able to identify, using CMVC′s reporting capability, the error prone modules and files. Eventually, a program was written to extract this information from CMVC during the evening, in a process transparent to the developers. The identification of high defect-density modules allowed the project teams to decide whether to redesign or partition the modules into more manageable and cohesive units.

Usability is another area of quality where CMVC is instrumental. Design enhancements are reported from users and the usability team throughout the development cycle. Each design change is reported by opening a feature. The feature owner is responsible for analyzing the design change and estimating the amount of work needed to implement it. The decision whether a design change should be implemented is made based on the technical feasibility and the implementation time required. With CMVC, the feature owner can defer a feature to a future release or return it to the originator as not being technically feasible. In general, CMVC tracks all reported design changes and retains information about their life cycle in the SQL database residing on the CMVC server.

B.4 Conclusions

Product quality is of key importance to organizations if they are to survive the fierce competition of the 1990s and beyond. The software engineering industry has recently experienced a clear shift toward putting techniques and processes in place to improve the quality of software products.

To make that quality goal a reality, CMVC:

- Tracks reported problems (defects), retrieving the necessary information and subsequently using it for both status reports and predictive models.
- Controls the evolution of the product, as well as the changes to files.
- Facilitates the tracking of design changes (features) to enhance the usability and performance of a product.

B.5 References

1. *IBM AIX CMVC/6000 Concepts*., SC09-1433-00, IBM Corporation.
2. Hubner, A. *ISO 9000 Implementation in Germany*, LOGON, Volume 4, Number 4, September 1992 (IBM Internal Use)
3. Humphrey, W.S. *Managing the Software Process*. SEI Series in Software Engineering, Addison-Wesley, August 1990.
4. Yourdon, E. *Decline and Fall of the American Programmer*. Yourdon Press/Prentice Hall, 1992.
5. Dunn, R. ″SQA: A Management Respective.″ *American Programmer*, November 1990.
6. Goel, A.L. ″Software Reliability Models: Assumptions, Limitations, and Applicability″. *IEEE Transactions on Software Engineering*. Vol. SE-11 No.12, December 1985.
7. Xie, M. *Software Reliability Modelling*. World Scientific Publishing, 1991.
8. Feiler, P.H. ″Configuration Management Models in Commercial Environments.″ Technical report CMU/SEI-91-TR-7, Software Engineering Institute, Carnegie Mellon University, 1991.

B.6 Discussion on Certain Reliability Models

A **Markov process** is a stochastic process {X(t), t>0} whose future development depends only on the present state of the process. In other words, the future behavior of the process does not depend on the past history of the process. The Jelinski-Moranda model is one of the earliest, based on Markov process, reliability models. The assumptions governing this type of model are:

- The number of initial software faults is an unknown but fixed constant.
- A detected fault is removed immediately, and no new fault is introduced.
- The times between failures are independent, exponentially distributed random quantities.
- All remaining software faults contribute the same amount to the software failure intensity. [7]

Nonhomogeneous Poisson Process Models (NHPP) have been used extensively and with success in studying hardware-reliability problems. They are

particularly useful when used to describe failure processes possessing certain characteristics, such as deterioration (or growth) trends. The application of NHPP models to software engineering is easily implemented. The cumulative number of software failures up to time t, N(t), can be described by a NHPP. Actually, N(t) follows a Poisson distribution with parameter m(t). The probability that N(t) is a given integer n is expressed by:

$$P\{N(t)=n\}=([m(t)]^n / n!)\ e^{(-m(t))},\ n = 0, 1, 2,...$$

where m(t) is the mean value function and describes the expected cumulative number of failures in [0,t), and (t) is the instantaneous failure rate. The underlying assumptions of the NHPP are:

- $N(0) = 0$
- $\{N(t), t \geq 0\}$ has independent increments
- $P\{N(t+h)-N(t)=1\} = \lambda(t) + o(h)$
- $P\{N(t+h)-N(t) \geq 2\} = o(h)$
- $o(h) \rightarrow 0$ as $h \rightarrow 0$.

Static Models are quite useful where time is not an important variable. A static model is used for estimating the number of software faults, assuming that faults are not removed immediately after detection. Xie [7] presented a collection of static models and described the underlying assumptions of each model.

Bayesian Analysis and Modeling. techniques are used to deal with certain problems that are common to Markov and NHPP models, for example, parameter estimation. Even the use of least square estimation, which is a straightforward process, does not give adequate results all the time [7]. A classic example of a Bayesian model and the most widely used in the existing literature is the Littlewood-Verall model. It assumes that times between failures are exponentially distributed with a parameter that is treated as a random variable, which is assumed to have a Gamma prior distribution. The choice of Gamma distribution is mainly due to its flexibility. It assumes that the successive times between failures are independent, exponentially distributed random variables with density function:

$$f(t_i|\lambda_i) = \lambda_i \exp\{-\lambda_i t_i\},\ i=1,2,....n;$$

where λ_i is an unknown parameter whose uncertainty is due to the randomness of the testing and the random location of the software faults.

B.7 Classification and Definition of Test Phases

The test phase of development consists of :

- unit test
- functional verification test
- product verification test (including stress test)
- customer environment test
- regression test

Each phase has its own objectives and strategies.

The purpose of unit test is to ensure that, after coding a particular segment, it works as designed. Also, after the integration of various code segments, unit testing is necessary to confirm that the resulting code still performs as planned. In our environment, unit testing is conducted by the same developers who write the code. Defects found during this phase are not usually reported. Instead, developers correct the defects prior to checking in the new modules.

The objective of a functional verification test is to ensure that a group of modules function as designed and that the design meets the specification. Any defects found during this phase are reported.

Product verification test focuses on installability, usability, stress standard compliance, configuration test, and performance of the software product.

The objective of a customer environment test is to ensure that the software product functions, as specified, in a customer-like environment composed of various software products and hardware configurations.

Regression test ensures that any changes made to the software product, as a result of defects found in any of the test phases, do not impact its functionality.

Appendix C. User Exit Samples and Suggestions

This appendix describes the two user exit programs used by our project. The first one inserts a module header into the C source files, and the second one generates the defect and feature numbers. It also gives some suggestions of user exit programs which could be associated with certain CMVC actions. Figure 117 shows how the /production/config/userExits file has been modified to enable the user exit programs.

```
## call insertHeader.ksh after CMVC checking
FileAdd     1 insertHeader.ksh

## call problemNumber.ksh after CMVC Defect and Feature opening
DefectOpen  2 problemNumber.ksh Defect
FeatureOpen 2 problemNumber.ksh Feature
```

Figure 117. Prod's config/userExits file

The user exit programs were integrated into CMVC by copying them into the /production/bin directory, so that CMVC could access and call the routine when necessary.

The last section of this chapter gives you some suggestions about how to use UE to adpat CMVC to your development methodology.

C.1 User Exit to Insert a Header and CMVC Keywords

Almost every development shop has a certain module header for source files that includes preliminary information, such as company copyright, file author, and function interface. Figure 118 on page 184 shows a shell script invoked when bringing a new file under CMVC control. This shell script inserts a module header at the top of file according to the type of the file. It also adds CMVC keywords enabling future identification of the executable file with the SCCS **what** command. CMVC keywords are expanded when a file, release, or level is extracted.

This user exit program illustrates a problem we had when we tried to get the UNIX login name from the parameter list. This parameter follows the Remark field. Each word of the remark text is considered as a parameter, which means that the number of parameters is not constant. When you want to get a parameter located after the Remark field, you have to calculate the position from the end of the parameter list as shown in Figure 118 on page 184 (**1**).

Note: If you do not fill in the Remark field, CMVC uses “Initial version” text.

```
#!/bin/ksh
#########################################################
#### FUNCTIONS
#########################################################
addHeader()
{
    ## test if the header should be added in this type of file
    case $1 in
    c | C | h) RC=0;;
            *) RC=1;;
    esac
}
Header_c_h()
{
    print "
(C) COPYRIGHT International Business Machines Corp. $(date +%Y)
All Rights Reserved
US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADF Schedule Contract with IBM Corp.
#-------------------------------------------------+
#                                                 |
# USAGE:                                          |
# WHERE:                                          |
#                                                 |
#                                                 |
#-------------------------------------------------+
"
}
Header_c()
{
    print '/*************************************************************'
    Header_c_h
    ## Get The real name of the person requesting the addFile
        Report -view users -where "login='$USERID'" -stanza \
        -family $FAMILY -become $ADMIN |
        grep name | sed  "s/name/Author's name/"
    print '*************************************************************/'
    ##%Z% expanded to @(#). String used by the UNIX command "what"
    ##%I% expanded to version identifier
    ##%W% expanded to file name, component name, release name,
    ##                level name if exists
    ##%D% expanded to date of the latest check-in
    ##%T% expanded to time of the latest check-in
    print 'static char CMVC_ID[]="%Z% %I% %W% %D% %T%"'
}
Header_C()
{
    Header_c
}
Header_h()
{
    print '/*************************************************************'
    header_c_h
    print '*************************************************************/'
    print '#define CMVC_ID "%Z% %I% %W% %D% %T%"'
}
```

Figure 118 (Part 1 of 2). UE Shell Script to Insert a Header and CMVC Keywords

```
#########################################################
#### MAIN PROGRAM
#########################################################
FAMILY=prod
ADMIN=lrconas
set -A PAR $*
FILENAME=${PAR[0]}
TMPFILE=${PAR[1]}
## get the CMVC ID
#position of CMVC ID = number of parameters minus 2
let I=${#PAR[*]}-2                                          1
USERID=${PAR[$I]}
## Test if a least two cmvcd daemons are running
## I need two because I use a CMVC command in this UE program
## Otherwise ===> dead lock
if [[ $(ps -u $FAMILY|grep cmvcd|wc -l) -lt 4 ]]
then
    print "The header cannot be inserted"
    print "      Not enough family daemons are running"
    print "The CMVC default number has been used"
    exit 1
fi

## get the file suffix
FILETYPE=${FILENAME##*.}
## Test if the header should be added
if [[ $(addHeader $FILETYPE) -eq 0 ]]
then
    ## call the function associated with the file type
    Header_$FILETYPE > $TMPFILE.$$
    cat $TMPFILE >> $TMPFILE.$$
    mv $TMPFILE.$$ $TMPFILE
    rm -f $TMPFILE.$$
fi
exit 0
```

Figure 118 (Part 2 of 2). UE Shell Script to Insert a Header and CMVC Keywords

C.2 User Exit to Generate Defect or Feature Number

This section shows how a user exit in CMVC is used to customize the process of generating defect and feature numbers. In particular we show how the number that is assigned to a new defect or feature is generated by a user-written routine. The user exit from CMVC would call this user-written function to change the defect or feature number as generated by CMVC itself after the defect or feature was created.

Almost every development shop has a certain scheme to be used for assigning feature and defect numbers. This scheme can even be part of the development contract, so certain enhancements (features) have to be mapped to specific feature numbers. So a common requirement is to have the numbers being assigned to a new feature or defect generated automatically from the system and have the numbers generated according to a customizable algorithm. CMVC has defined the user exit, *getNumber*, to be called after a new defect or feature was added to CMVC. In our project, we used this user exit to modify the number

as it was generated by CMVC and have the new number calculated from a routine we had written ourselves. Although CMVC recommends not to call a CMVC database function (in this case to change a defect number in the user exit) from inside a user exit, this was no problem in our case as the database modification would not cause any deadlock or loop problems.

The routine used to generate the new defect number was rather simple, as we had anticipated. The routine returned a string for each invocation, that was composed of a prefix identical to the family name followed by a unique number as a suffix which was incremented for each call. Both prefix and suffix were separated by an underscore character. The routine used a file called DATAFILE to store the last unique number for the family. To ensure that the access to that DATAFILE is sequential, a lock file called LOCKFILE is used to check whether or not the file can be accessed or not. As the numbers had to be unique for each family, we stored the DATAFILE and LOCKFILE in the CMVC family directory.

Figure 119 on page 187 shows the code for the number generation routine. The routine checks whether the lock file currently exists (**1**), and whether the wait cycle is implemented (**4**) if it does. If it is not locked, the number is read from the DATAFILE (**2**), and incremented for each call. The routine then concatenates the family name and the generated number (**3**).

```
#include <stdio.h>
/* This program generates a number incremented by one for each call  */
/* It is invoked with two parameters:                                 */
/*        getNumber dataFileLocation prefix                           */
/*  1- a string (char *), directory where DATAFILE and LOCKFILE       */
/*     files will be created                                          */
/*  2- a string (char *), which is added in front of the number       */
/*                                                                    */
/* And print a number such as prefix_00001                            */
/* It returns a code 0 if the generation is successful                */
/* otherwise a code 1                                                 */

int num(char *familyHome, char *prefix, char **result)
{
/* This function generates a number incremented by one for each call */
/* It is invoked with three parameters:                               */
/*  1- a string (char *), directory where DATAFILE and LOCKFILE       */
/*     files will be created                                          */
/*  2- a string (char *), which is added in front of the number       */
/*  3- an address to the result string (char **)                      */
/* The number part of the result string is 5 characters.              */

/* For example:                                                       */

/* char *familyHome="/production";                                    */
/* char *prefix="prod";                                               */
/* char *next_number_as_string;                                       */
/* int return_code;                                                   */
/* return_code = num(familyHome, prefix, &next_number_as_string)      */
/* printf("Number as string is %s\n", next_number_as_string);         */
/* free(next_number_as_string);                                       */
/* return_code = num(familyHome,prefix, &next_number_as_string)       */
/* printf("Number as string is %s\n", next_number_as_string);         */
/* free(next_number_as_string);                                       */
/* would print:                                                       */
/* prod_00001                                                         */
/* prod_00002                                                         */
/* The function would use the /production/prod.DATAFILE               */
/* and /production/prod.LOCKFILE                                      */
/* files, so the caller must have write access to these directories   */

/* The caller must free the memory being allocated to the address     */
/* pointed to by the second call parameter.                           */

/* Return values are:                                                 */
/*  0: correct value generated                                        */
/*  1: an error occurred, and a dummy string is returned              */

/********************************************************************/
/* Variable Declaration                                             */
/********************************************************************/
  FILE *Lockfile, *Datafile;
  int maxtry=3, i, number, position, error=1, okay=0;
  char *Lockfilename, *Datafilename;
  char *command;
```

Figure 119 (Part 1 of 3). Routine to Generate a Unique Number per Invocation

```
/*********************************************************************/
/* Variable Initialization                                           */
/*********************************************************************/

  *result=malloc(strlen(prefix)+5+1); /* e.g. prefix_00001 */
  strcpy(*result, prefix);
  strcat(*result,"00000");

  Lockfilename = malloc(strlen(familyHome)+1+strlen(prefix)+strlen(".LOCKFILE")+1);
  strcpy(Lockfilename,familyHome);
  strcat(Lockfilename,"/");
  strcat(Lockfilename,prefix);
  strcat(Lockfilename,".LOCKFILE");

  Datafilename = malloc(strlen(familyHome)+1+strlen(prefix)+strlen(".DATAFILE")+1);
  strcpy(Datafilename,familyHome);
  strcat(Datafilename,"/");
  strcat(Datafilename,prefix);
  strcat(Datafilename,".DATAFILE");

  command = malloc(strlen(Lockfilename)+strlen("touch ")+1);
  strcpy(command, "touch ");
  strcat(command, Lockfilename);
/*********************************************************************/
/* Number Generation                                                 */
/*********************************************************************/

  for (i=1; i<maxtry; i++) {
    if (  (Lockfile = fopen(Lockfilename, "r") )==NULL) {          1
      /* no lockfile was found.  Create one and proceed */
      system(command);
      if ( (Datafile = fopen(Datafilename, "r+") )== NULL) {
        /* Could not open the data file for update.  Try to create it */
        if ( (Datafile = fopen(Datafilename, "w") )== NULL) {
          /* could not open the data file */
          strcpy(command, "rm ");
          strcat(command, Lockfilename);
          system(command);
          free(command);
          free(Lockfilename);
          free(Datafilename);
          return(error);    /* dummy number */
        } else
        {
          /* created a new DATAFILE */
          number = 0;
        }
      } else
      {
        /* could open an existing data file */
        fgetpos(Datafile, &position);
        fscanf(Datafile, "%d", &number);                          2
        fsetpos(Datafile, position);
      }
```

Figure 119 (Part 2 of 3). Routine to Generate a Unique Number per Invocation

```
      fprintf(Datafile, "%d", ++number);
      fclose(Datafile);
      strcpy(command, "rm ");
      strcat(command, Lockfilename);
      system(command);
      sprintf(*result+strlen(prefix), "%d", number);                    3
      free(Lockfilename);
      free(Datafilename);
      free(command);
      return(okay);
    } else {
      fclose(Lockfile);                                                 4
      /* lockfile exists.  Wait for one second and retry */
      sleep(1);
    }
  }
  /* maxtry is reached.  Return the dummy number */
  free(Lockfilename);
  free(Datafilename);
  free(command);
  return(error);
}
/*********************************************************************/
/* Main Program                                                      */
/*********************************************************************/
main (argc,argv)
int argc;
char *argv[];
{
char *numb;
int rc;
/* get number problem */
rc=num (argv[1],argv[2],&numb);
/* test if generation successful
if ( rc == 0 )
{
    printf("%s",numb);
    return 0;
}
else
    return 1;
}
```

Figure 119 (Part 3 of 3). Routine to Generate a Unique Number per Invocation

The routine shown in Figure 119 on page 187 is invoked from the user exit, problemNumber.ksh, shown in Figure 120 on page 190.

Here also we have a similar problem in determining the position of the defect/feature number (**1**).

The user exit program calls the number generation routine to calculate a new unique number (**2**) and then calls the CMVC Defect or Feature command to modify the number for the new defect (**3**).

```
#!/bin/ksh
FAMILY=prod
ADMIN=lrconas
PREFIX=$FAMILY
set -A PAR $*
PROBLEMTYPE=${PAR[0]}
##get the problem number
#position of defect/feature number = number of parameters minus 3
let I=${#PAR[*]}-3                                          1
PROBLEMNUMBER=${PAR[$I]}
## Test if a least two cmvcd daemons are running
## I need two because I use a CMVC command in this UE program
## Otherwise ===> dead lock
if [[ $(ps -u $FAMILY|grep cmvcd|wc -l) -lt 4 ]]
then
    print "The number of the $PROBLEMTYPE cannot be generated:"
    print "      Not enough family daemons are running"
    print "The CMVC default number has been used"
    exit 1
fi

##generate the problem number
FAMILYHOME=$(lsuser -c -a home $FAMILY)
NAME=$(getNumber ${FAMILYHOME##*:} $PREFIX)                 2
if  [[ $? -eq 0 ]]
then
    $PROBLEMTYPE -modify $PROBLEMNUMBER \                   3
                 -name $NAME \
                 -family $FAMILY \
                 -become $ADMIN
    exit 0
else
    ERROR=$?
    print "The number of the $PROBLEMTYPE cannot be generated:"
    print "      number generator returns $ERROR code"
    print "The CMVC default number has been used"
    exit 1
fi
```

Figure 120. CMVC User Exit to Modify the Defect Number

C.3 User Exits Suggestions

This section suggests some user exit programs for the most used CMVC actions. Many other user exits may be required by your software development process, company policies, and project practices.

Examples of user exit programs associated with extracting a level or a release include:

- Upload to a mainframe of a level after extraction with removing level files on the target UNIX directory.
- Update subroutine UNIX library if the release represents a file created with the UNIX **ar** command.
- Automate the build process.

Some ideas associated with the problem tracking process include:

- Start up a quality inspection process when the state of a defect or a feature moves to *review* state.
- Create a clone defect or feature if a test or verification record is rejected.
- Supply quality database and tools with defect and feature attributes to calculate some metrics.

These are some suggestions for user exit programs linked to both CMVC *FileCheckin* and *FileCreate* actions:

- Unlock a checked-out file if no differences are detected between the working version and the current version. This avoids creating a large number of file versions and thus saves disk space.
- Update the common working source directory with all new source files.
- First compile a source file, and, if the new compilation is successful, accept the check-in and incorporate the object file into a UNIX subroutine library (using the UNIX **ar** command). If compilation errors are detected, the check-in action is aborted.
- Run some tools to analyze if the programming rules have been respected. For example insure that no “goto” instructions are used or check number of nested IF statements.
- Compute the number of lines of code which have been developped.
- Compress the binary files to save disk space. In this case, you should uncompress those files when you check out or extract them.

You also can create new tables in the family database and update these tables through user exit programs. For example, you could calculate the number of defects in each state for each component by creating a table with a column for each state and a column for the component name as shown in Figure 121.

```
create table defectMetrics
(
open number,
design number,
review number,
size number,
working number,
verify number,
close number,
return number,
cancel number,
total number,
component char(63)
);
```

Figure 121. Creating a New Tables in CMVC Family Database

This table should be updated by user exit programs, written in SQL, associated with each defect action or periodically started by a shell script executed by the UNIX **cron** daemon. In this example, at any time, you can get the number of opened defect for a given component by using a simple SQL query:

```
select open from defectMetrics where component='MVS';
```

DANGER

Be careful when using user exits! Too many user exit programs slowdown CMVC response time!

Appendix D. Hints and Tips for Using CMVC

This appendix gives some procedures we used for maintenance activities involved in administering our CMVC family. It also describes solutions to solve some problems we had when we made some minor file modifications.

D.1 Maintaining CMVC family

We used the UNIX **cron** daemon to run the following activities at specified intervals on the CMVC server host:

- Cleaning up the audit log file
- Aging the defects/features
- Getting the name of the obsolete levels
- Backing up the family data.

To notify the **cron** daemon of the maintenance activities, we logged in with the *prod*UNIX login name. Then we issued the **crontab -e** command. This command creates the /usr/spool/cron/contabs/prod file, if does not exist, otherwise it places an editable copy of the existing one in your current directory. Then we edited it with the editor **vi**. We entered the lines shown in Figure 122. The first line starts the program age.sh at 11:30 every night of each work weekday. The second line starts the program cleanLog.sh at 11:30 every Sunday night. The third line starts the program levelClean.sh at 9:00 the first day of each month. The fourth line starts the program backup2Tape.sh at 11:45 every night of each work weekday. The last line starts the program backup2VM.sh at 11:45 every Sunday night.

```
30 23 * * 1-5 $HOME/bin/age.sh > /dev/null 2>&1
30 23 * * 0 $HOME/bin/cleanLog.sh > /dev/null 2>&1
00 9 1 * * $HOME/bin/levelClean.sh > /dev/null 2>&1
45 23 * * 1-5 $HOME/bin/backup2Tape.sh > /dev/null 2>&1
45 23 * * 0 $HOME/bin/backUp2VM.sh > /dev/null 2>&1
```

Figure 122. Creating crontab File for prod Family

A description of each shell script is given in the following sections.

D.1.1 Aging Defects and Features

The age.sh shell script, shown in Figure 123 on page 194, updates the age of defects and features with the state set to either *open*, *design*, *size*, *review*, or *working*.

The syntax of the CMVC **age** command depends on the database used by the server, which might be ORACLE, INFORMIX, SYBASE, or DB2/6000. The path name where **sqlplus** is located must be set in the PATH environment variable, because the CMVC server **age** command uses it.

The first statement (**1**) enables you to export the environment variables initialized in the .profile file of the *prod* account, which are required to run the commands used by this shell script. Using this statement you can use this shell script for any family without modifications.

```
#!/bin/ksh
. $HOME/.profile                                    1

age $CMVC_FAMILY $ORACLE_PASS > /dev/null 2>&1
#age $CMVC_FAMILY $DB2_PASS > /dev/null 2>&1     ## for DB2/6000
#age $CMVC_FAMILY $SYBASE_PASS > /dev/null 2>&1  ## for SYBASE
#age $CMVC_FAMILY > /dev/null 2>&1               ## for INFORMIX
```

Figure 123. Shell Script for Aging Defects and Features of the prod Family

D.1.2 Cleaning Family Audit Log and User Log

The cleanLog.sh shell script, shown in Figure 124, keeps the family audit log from growing too large. When the log becomes too large, this shell script sends a note to each CMVC user defined in the family *prod* asking them to remove their CMVC user log file. Keep in mind that for each action executed from the GUI, the corresponding command is logged in a file located in the directory from which the end user started the CMVC client.

```
#!/bin/ksh
. $HOME/.profile

cmvclog.cleanu

mail $(Report -view users -raw -whe " login like '%'" \
-fam $CMVC_FAMILY -beco $CMVC_SUPERUSER | cut -f4 -d'|') <<!
Don't forget to remove your CMVC log file
if you don't need it
Thanks
!
```

Figure 124. Shell Script Cleaning-up Log Files of the prod Family

D.1.3 Managing Obsolete Levels

CMVC enables the system administrator to reclaim the file system and database space by archiving obsolete or completed levels, or releases. Archived data can be restored if required but not in the same family. Before archiving a level or release, you should review the prerequisites described in *IBM CMVC Server Administration and Installation*.

D.1.3.1 Getting List of Potential Obsolete Levels

The levelClean.sh shell script, shown in Figure 125 on page 195, sends a note to the owners of obsolete committed levels, that are not in a production (2). and which are one or more months old and. This note asks them if the SCM administrator might archive the obsolete levels to free storage for the creation of new levels. The date for obsolescence is calculated as any level committed one or more months ago (1).

The **Report** command uses an SQL statement to determine whom the note should be sent to (3).

```
#!/bin/ksh
. $HOME/.profile

let MONTH=$(date +%m)-1                                          1
let YEAR=$(date +%y)
let DAY=$(date +%d)
## test if January
if [[ $MONTH -eq '0' ]]
then
    let YEAR=YEAR-1
    let MONTH=12
fi
## test if February
if [[ $MONTH -eq '2' && $DAY -gt '27' ]]
then
   let DAY=27
fi
DATE=$(printf %2.2d"/"%2.2d"/"%2.2d $YEAR $MONTH $DAY)

mail $(Report -view users -raw -whe " id in \                    2
(select userId from levelView where commitDate < '$DATE' \       3
and type not in ('production'))" \
-fam $CMVC_FAMILY -beco $CMVC_SUPERUSER | cut -f4 -d'|') <<!
You are owner of a committed or completed level one or more months old.
If your levels are obsolete,
send me a note listing the obsolete level names.
Thanks
!
```

Figure 125. Shell Script Asking for Obsolete Levels

Because the CMVC **Report** command is used in the `levelClean.sh` script shell, the CMVC client must be installed on the server host, and there must be a host list entry for the family superuser CMVC ID at that host.

D.1.3.2 Archiving Obsolete Levels

When you have the list of the levels that you can archive, log in with the *prod* UNIX name. Figure 126 illustrates the different commands you execute to start a level or release archive. First you stop the server daemons and restart them in maintenance mode (flag `-m`), and then you execute the **cmvcarchive** command. The number of daemons shown is arbitrary; there is no relationship between that number and the archive procedure. Your procedure must stop all of your CMVC daemons.

```
prod@bering:/production>stopCMVC prod
prod@bering:/production>cmvcd -m prod 3
prod@bering:/production>cmvcarchive
```

Figure 126. Starting a Level Archive

Figure 127 on page 196 illustrates the dialog during the archive of the *0* level of the *MVS_Release_0* release.

```
Select one of the following options:
    1.  Check archive prerequisites and estimate the storage required
        to perform an archive.
    2.  Perform an archive.
    3.  Exit.
Enter selection: 2

Select one of the following archive options:
    1.  Archive levels of a release up to, and including, a specified level.
    2.  Archive one or more releases.
    3.  Exit.
Enter selection: 1

Enter the level name: 0

Enter the release names (eg. rel_1.1 rel_1.2): MVS_Release_0

Do you want the archive objects removed from the CMVC family?
    1.  No. Do not remove the archived objects from the CMVC family.
    2.  Yes. Remove the archived objects from the CMVC family.
Enter selection: 2

You have selected to remove the archived objects from the CMVC family.
Please confirm your decision by selecting one of the following options:
    1.  No. Do not remove the archived objects from the CMVC family.
    2.  Yes. Remove the archived objects from the CMVC family.
Enter selection: 2

Do you want the archive objects stored:
    1.  In a local file system?

    2.  On media in an external device (eg. tape device)?
Enter selection: 2

Enter the full name of the raw device (eg. /dev/rmt1 or /dev/rfd0 ): /dev/rmt0

The cmvcarchive program requires a working directory when archiving
to a device.  Select one of the following options:

    1.  Use the CMVC family's home directory as the working directory.
    2.  User another local file system as the working directory.

Enter selection: 1
```

Figure 127 (Part 1 of 2). Archiving Level ,0, of MVS_Release_0

```
Connecting to the database.
Checking that all archive prerequisites have been satisfied.
  Working ......... complete.
Archive prerequisites have been verified.
Generating the database populate file.
  Processing CMVC Releases and their associated objects.
  Working .... complete.
  Processing CMVC Levels and their associated objects.
  Working .... complete.
  Processing CMVC Tracks and their associated objects.
  Working ......... complete.
  Processing CMVC Files.
  Working ......... complete.
  Processing CMVC Paths.
  Working ...... complete.
  Processing CMVC Versions.
  Working ...... complete.
  Processing CMVC Components.
  Working ........... complete.
  Processing CMVC Users and their associated objects.
  Working ......... complete.
Database populate file has been created successfully.
Generating the database export file for the notes and versions tables.

Database export file has been created successfully.
Determining which level maps are to be archived.
  Working .... complete.
Determining the CMVC family's configuration files that are to be archived.
  Working ......... complete.
Determining which version control files are to be archived.
  Working ...... complete.
Copying all CMVC archive data to /dev/rfd0 using the cpio program.
483 blocks were copied.

The cpio program is now complete and the cmvcarchive program
is ready to remove the archived objects from the CMVC environment.

Before continuing, inspect the results of the cpio program and
select one of the following operations:

    1.  Continue.
    2.  Quit.

Enter selection: 1
Removing the archived objects.
  Removing CMVC Files and Versions...this may take some time.
  Removing CMVC Levels and their associated objects.
  Removing CMVC Tracks and their associated objects.
  Removing CMVC Defects and their associated objects.
Committing the database transactions.
Disconnecting from the database.
Archive complete.
```

Figure 127 (Part 2 of 2). Archiving Level ,0, of MVS_Release_0

To inspect the results of the archive process, you can use the **cpio** command.

```
cpio -itv < /dev/rmt0
```

The output of the **cpio** command is shown in Figure 128 on page 198. You can see the level map file (1) and the arbitrary names of the SCCS files created by CMVC for the files belonging to the level (2 to 3). CMVC translates your "file" names into these SCCS file names automatically; CMVC end users never use these names.

```
100644 prod       35 Apr 12 13:59:33 1994 CMVCarchive.users
100755 prod    62284 Apr 12 13:59:36 1994 CMVCarchive.dbData
100755 prod    12288 Apr 12 13:59:41 1994 CMVCarchive.export
100644 prod      626 Jul 31 11:58:27 1993 maps/MVS_Release_0/0  1
100644 prod     5076 Jul 29 15:16:34 1993 config.ld
100644 prod    10082 Jul 15 17:20:29 1993 authority.ld
100644 prod     4904 Jul 15 17:20:29 1993 interest.ld
100644 prod      303 Jul 15 17:20:29 1993 cfgcomproc.ld
100644 prod      604 Jul 15 17:20:29 1993 cfgrelproc.ld
100444 prod      355 Jul 31 11:00:28 1993 vc/0/0/0/0/s.22       2
100444 prod    66452 Jul 31 10:58:52 1993 vc/0/0/0/0/s.20
100444 prod     6581 Jul 31 10:54:46 1993 vc/0/0/0/0/s.18
100444 prod    12481 Jul 31 10:54:17 1993 vc/0/0/0/0/s.17
100444 prod    21293 Jul 31 10:53:49 1993 vc/0/0/0/0/s.16
100444 prod    22524 Jul 31 10:53:03 1993 vc/0/0/0/0/s.15
100444 prod     1767 Jul 31 10:49:01 1993 vc/0/0/0/0/s.14
100444 prod     4197 Jul 31 10:48:31 1993 vc/0/0/0/0/s.13
100444 prod     2035 Jul 31 10:47:14 1993 vc/0/0/0/0/s.12
100444 prod     1153 Jul 31 10:46:35 1993 vc/0/0/0/0/s.11
100444 prod     1268 Jul 31 10:44:42 1993 vc/0/0/0/0/s.10
100444 prod     1490 Jul 31 10:44:09 1993 vc/0/0/0/0/s.09
100444 prod     1333 Jul 31 10:43:33 1993 vc/0/0/0/0/s.08
100444 prod     1957 Jul 31 10:41:17 1993 vc/0/0/0/0/s.05
100444 prod      804 Jul 31 10:25:51 1993 vc/0/0/0/0/s.04
100444 prod     1316 Jul 31 10:21:59 1993 vc/0/0/0/0/s.03       3
100644 prod       72 Apr 12 13:59:42 1994 CMVCarchive.VCcleanup
100644 prod     1819 Apr 12 14:00:01 1994 CMVCarchive.info
100755 prod      512 Apr 12 14:00:01 1994 CMVCarchive.CPIOlist
483 blocks were copied.
```

Figure 128. Created Files for Archive of the Level ,0, of MVS_Release_0

D.1.4 Backing Up a Family Data

The backup2Tape.sh shell script shown in Figure 129 on page 199 backs up *prod* family data onto a tape.

Before backing up data, you have to start the CMVC server in maintenance mode to keep consistency between data located in the */production* file system, and data stored in the database (1). The maintenance mode enables you allows to access data in read-only mode. The backup is performed using the UNIX **tar** command (2). Other UNIX commands, such as **cpio** or **rdump**, could be used o backup data onto a remote machines's device. Where the **tar** command will find the database files depends on the type of database CMVC is using Refer to *IBM CMVC Server Administration and Installation* for more detailed information. Once the backup is completed, the CMVC server is started in normal mode (3).

```
#!/bin/ksh
. $HOME/.profile

stopCMVC $CMVC_FAMILY                                                    1
cmvcd -m $CMVC_FAMILY 3
print "CMVC Family $CMVC_FAMILY Backup: " $(date) >> $HOME/backup.trk
cd /
tar -cvf /dev/rmt0 ./$HOME ./$ORACLE_HOME/dbs/*$ORACLE_SID.*             2
stopCMVC $CMVC_FAMILY                                                    3
cmvcd $CMVC_FAMILY 3
notifyd
```

Figure 129. Shell Script Backing up a Family to a Tape

The previous backup is a daily procedure archiving data onto a tape. The backup presented in Figure 130 is a weekly procedure that uploads family data onto a VM mainframe. The statements related to CMVC server daemons of this shell script are the same as the tape backup shell script. The backup is also performed by using the UNIX **tar** command (1) but the output of the command is compressed with the UNIX **compress** command before it is stored in the file system. The compressed files are uploaded to the mainframe by using **ftp** command (2). If the host name of the mainframe is *sjsvm28*, the login name on the mainframe is *archive*, and the password of that login name is *rtyw67sa*, add the following line in the .netrc file of the *prod* family:

```
machine sjsvm28 login archive password rtyw67sa
```

This is necessary for **ftp** to work in batch mode.

Once unloaded the compressed files are deleted on the file system (3).

```
#!/bin/ksh
. $HOME/.profile

stopCMVC $CMVC_FAMILY
cmvcd -m $CMVC_FAMILY 3
print "CMVC Family $CMVC_FAMILY Backup: " $(date) >> $HOME/backup.trk
cd /
tar -cvf - ./$HOME | compress > $HOME/cmvc$CMVC_FAMILY.tarzbin          1
tar -cvf - ./$ORACLE_HOME/dbs/*$ORACLE_SID.* | compress \
> $HOME/orac$CMVC_FAMILY.tarzbin
stopCMVC $CMVC_FAMILY
cmvcd $CMVC_FAMILY 3
notifyd
ftp sjsvm28 > /dev/null 2>&1 <<!                                         2
prompt
bin
mput $HOME/cmvc$CMVC_FAMILY.tarzbin
mput $HOME/orac$CMVC_FAMILY.tarzbin
!
rm $HOME/cmvc$CMVC_FAMILY.tarzbin $HOME/orac$CMVC_FAMILY.tarzbin         3
```

Figure 130. Shell Script Backing up a Family to a VM Mainframe

D.1.5 How to Terminate a CMVC Transaction

When you are developing user exit programs you may cause a deadlock condition at the server For example, if you have started only one CMVC daemon and you use a CMVC client command in your user exit program: a second CMVC command client will not be able to execute, and the first one will “hung.” Infact, each CMVC client command requires a CMVC daemon to access the CMVC server. While a user exit is executing, one CMVC daemon is in use already. Another example is when you extract a level or a release and the network goes down. The CMVC client command will not be able to recover and it will hung.

The CMVC server provides you with a real-time program called **monitor** that permits the CMVC family administrator to monitor the activity of the CMVC server daemons. This program could be used to tune the CMVC server. To invoke the CMVC activity monitor, you must be logged in with the *familyName* name on the CMVC server. Then you can issue the **monitor 2** command, where 2 is the time in seconds between successive screen refreshes. An example of the CMVC activity monitor screen with three daemons running, one of which is running the command, is shown in Figure 131. You can see that the 01 daemon runs the UNIX process 28823 for the CMVC action, ReleaseExtract.

```
3 of 3 cmvcd daemons running.  Shared mem size is 1088.
Press any key to quit.
Total hits = 8

01,28823,00003,04/11/94,08:39:25,ReleaseExtract,truls,aixcase2,bering,MVS_Release_0
02,26730,00003,

03,35488,00002,
```

Figure 131. CMVC Activity Monitor Screen Example

To terminate the UNIX process 28823 shown in Figure 131, you log in as *prod*, and issue the following command:

```
prod@bering:/production>kill 28823
```

D.2 File Modifications with Track and Level Subprocesses Turned On

When the *track* and *level* subprocesses are selected for a release, you can change the meta-data associated with files; however, it is not easy. easily as you want. The following sections gives you some procedures we discovered, and then applied to make minor changes on files.

D.2.1 Modifying File Base Name and Path Name

If you enter a file name incorrectly, or discover as a result of the development process that a better tree design is required, you might want to rename the file. When you try, you may get the error message shown in Figure 132 on page 201.

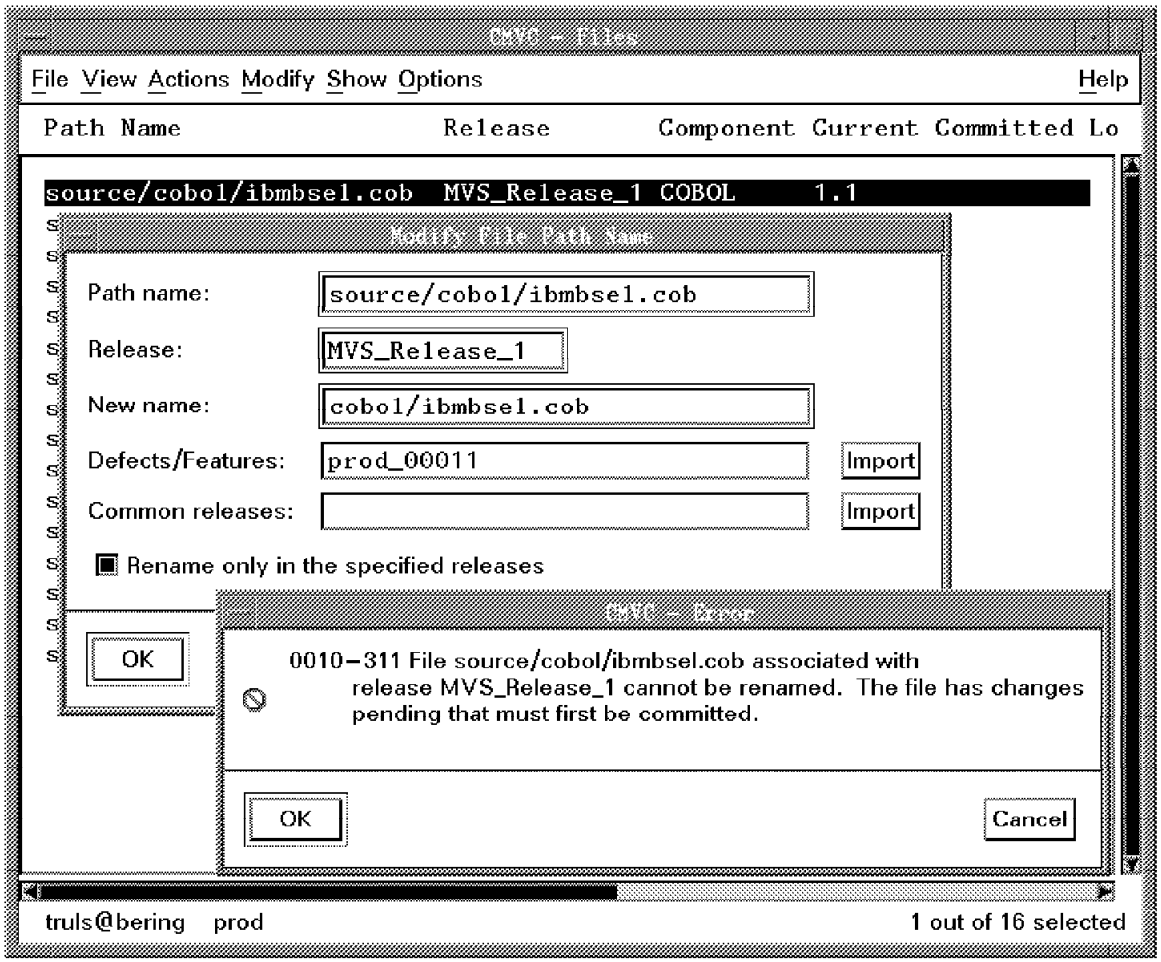

Figure 132. Error Message Issued when Renaming a File

That message could appeared if the release to which the file is attached, has a process that includes the *track* subprocess and the file has already been changed in reference to a track, which has not been committed yet. If you want to change the name to correct a mistake, and you do not want control this file name change, the easiest and fastest to do it is:

1. Wait for all the tracks and levels associated with the release to reach the *complete* state
2. Modify the current process release to *no_track*
3. Modify the name of the file
4. Modify again the process release from *no_track* to the original process.

If you want to keep track of this change, you have two solutions. The first solution consists of postponing the change, and then performing the following actions (preferably by a superuser, if you want to go faster):

1. Wait for all the tracks and levels associated with the release to reach the *complete* state
2. Modify the current process of the release to *track_level* if it is not already selected. Now, you have either *approval*, *fix*, or *test* subprocesses prevailing

3. Modify the current process of the component managing the file to *emergency_fix* if it is not already selected. In this case, you have neither the *DesignSizeReview*, nor *verify* subprocesses involved
4. Open a new defect on this component and accept it
5. Create a track for the defect and the release
6. Modify the name of the file in reference to the track
7. Create a new level for the release
8. Add the track as the level member of the new level
9. Commit and complete the level
10. Return the original process definitions of the component and the release to their original values.

The second solution could take longer longer than the first one depending on the component and release processes already in place:

1. Open a new defect
2. Perform actions related on the component process until the defect state has reached *working*
3. Create a track for the defect and the release
4. If the *approval* subprocess is selected, accept the approval records
5. Modify the name of the file in reference to the track
6. If the *fix* subprocess is used, complete the fix record
7. Create a new level for the release
8. Add the track as the level member of the level
9. Commit and complete the level
10. If the *test* subprocess is selected, accept the test records
11. If the *verify* subprocess is used, accept the verification record.

To rename a file that occurs in more than one release, apply either solution, but create one track and one level for each release for which you want that the name of the common file changed. Then enter the releases in the Common releases field in the Modify File Path Name dialog box, and select **Rename only in the specified releases**.

D.2.2 Deleting a File

When you try to delete a file, you have the same options described in the section D.2.1, “Modifying File Base Name and Path Name” on page 200. Whichever solution you apply, instead of selecting **Path Name...** from the Modify menu, you select **Delete...** from the Action menu.

Only the following pre-defined access authority groups can delete a file:

- *developer+*
- *writer+*
- *builder*
- *releaselead*
- *componentlead*

- *projectlead*.

If the CMVC Files window is displayed when you delete files, the deleted files continue to be listed in the CMVC Files window after a refresh, but the *Deleted* column is filled in for each file. To display the list of nondeleted files, execute a query from the Open File List dialog box with the SQL operator, **is null**, for the Delete date field.

A deleted file can be recreated, but you also have to follow the procedure described in the section D.2.1, “Modifying File Base Name and Path Name” on page 200. Instead of selecting **Path Name...** from the Modify menu, select **Recreate...** from the Action menu.

If you want to remove a file from CMVC permanently, you should destroy it after having deleted it. Destroyed files are no longer displayed in the CMVC File window. You can reuse the path name of a destroyed file for another file that you want to bring under CMVC control.

D.3 How to Reuse a Track in Integrate Status with Level Subprocess On

If you want to change a file belonging to a release for which a track associated with a defect involving that file is in the *integrate* state, you get an error message telling that the track is in *integrate* and can no longer be used. To reuse a track, you can apply the solution described in the following paragraphs.

The prerequisites to reusing a track are: the track state is *integrate* and the state of all levels of which the track is a member, is *integrate*.

After applying the following procedure, the track can be specified:

1. Remove the level members associated with the track. In the CMVC - Tracks windows, highlight the track, and then select **Level Members...** from the Show menu. In the CMVC - Level Members window, select all the level members, and then select **Remove...** from Actions menu
2. Move the track state from *integrate* to *fix*
3. Activate the fix record, if it exists, and attached it to the component managing the files you are going to change. In the CMVC - Fix Records window, select **Open List...** from the File menu. Fill:
   - The Defects/Features field with the defect or feature to which the track is linked
   - The Releases field with the name of the releases for which the track has been created
   - The Components field with the list of components managing the files to be changed.

   Select **OK** to display the list of the fix records. Highlight all the relevant fix records, then select **Activate...** from Actions menu, and select **OK** in the Activate Fix Records dialog box.

D.4 Common and Shared File

A CMVC file is identified by a file name, a release name, and a version identifier. Each file version can be attached to different releases and managed by different components. A file with multiple in multiple is called a shared file. A file with only one version shared by a set of releases is called common file. A file can be common to a set of releases and shared among other releases.

Figure 133 shows a list of common and shared files. You can see that a file can be managed by two different components if it is shared by two different releases. Each component manages a different line of development in the version history. From this this list, you can extract the following information:

- The file, *cobol/ibmbuenr.cob*, is a shared file of both the *MVS_Release_0* and *MVS_Release_1* releases because the committed versions are different. This means that the **Break common link** (flag **-force** for the command-line interface) was used *cobol/ibmbuenr.cob* of *MVS_Release_1* was checked in. Then the version *1.3* of *cobol/ibmbuenr.cob* of *MVS_Release_1* was linked to *cobol/ibmbuenr.cob* of *MVS_Release_0*. Originally *cobol/ibmbuenr.cob* was a common file of both *MVS_Release*_0 and *MVS_Release_1*.
- The file, *cobol/ibmbuins.cob*, is a common file of both *MVS_Release_0* and *MVS_Release_1*. This means that *cobol/ibmbuins.cob* of *MVS_Release_1* was checked out, and then *MVS_Release_0* was specified in the Common releases field (flag **-common** for the command-line interface). The track associated with *MVS_Release_0* was committed because the current and committed versions of *MVS_Release_0* are the same.
- The file, *cobol/ibmbupd.cob*, is a shared file of both *MVS_Release_0* and *MVS_Release_1*. The path name is common but the content of each instance is different.

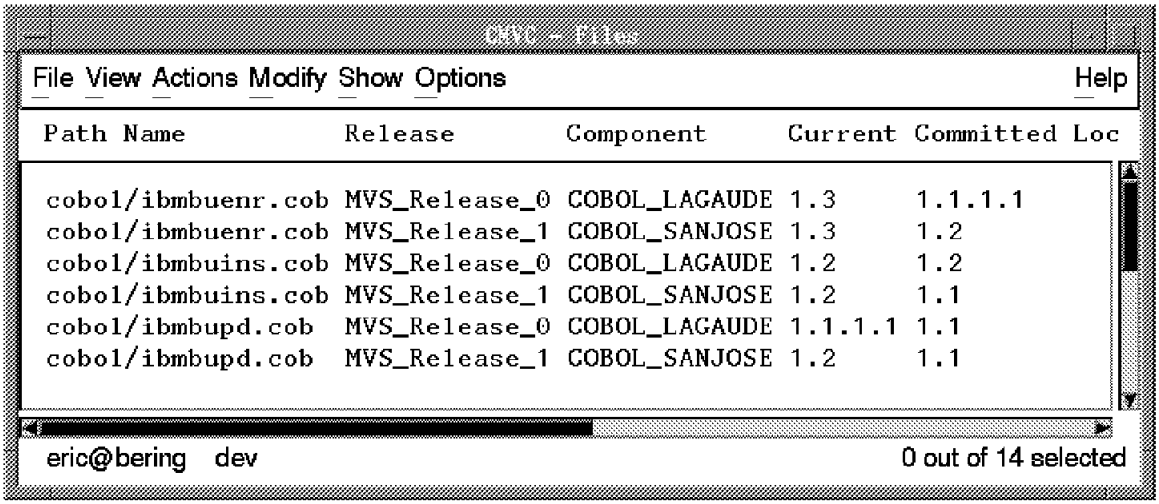

Figure 133. Example of Common and Shared Files

Figure 134 on page 205 shows the change history of the file *source/cobol/ibmbsel.cob* of the *MVS_Release_0* release. This file has been changed three times, and each change has been integrated in a level.

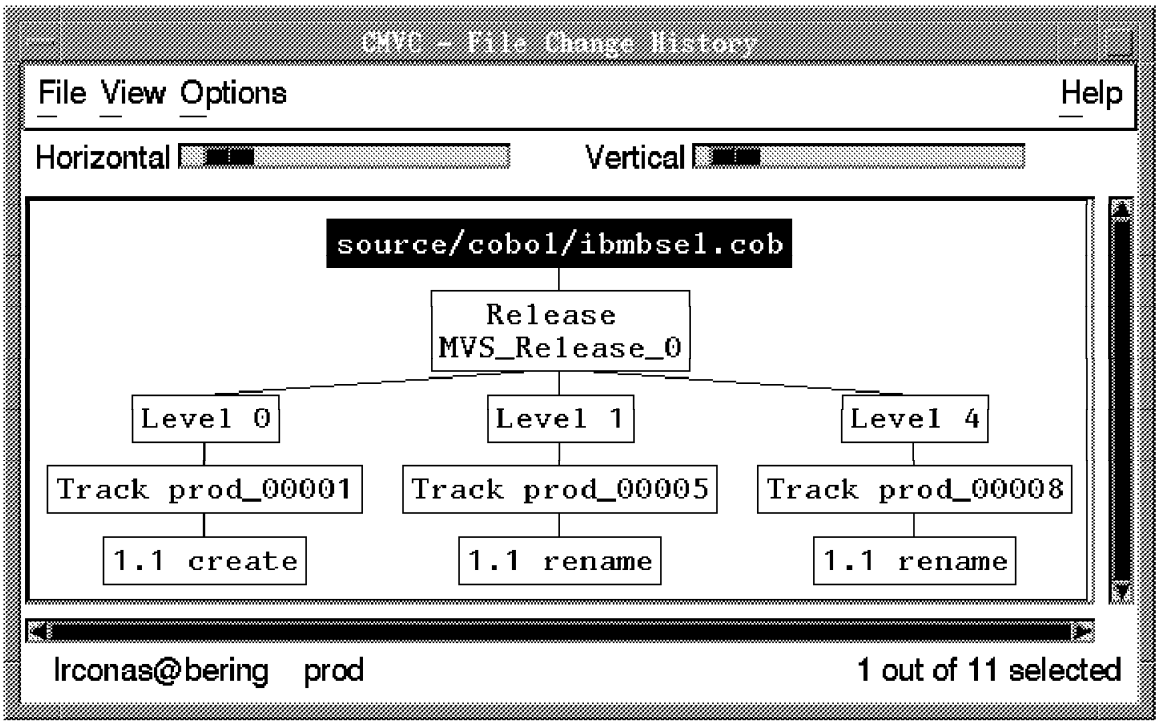

Figure 134. Example of File Change History

Appendix E. CMVC and SDE WorkBench/6000

This appendix addresses several topics that we feel deserve your special attention, although this appendix does not by any means address all there is to say about using CMVC from SDE WorkBench/6000. Become familiar with the pull-down menus provided in the Development Manager, especially the **Set Context Mapping...** selection on the CMVC pull-down. Study the question of execution host assignment for the CM class tool and make your users aware of the choices you have made for them in this area. If you think you are experiencing anomalous behavior, use the Broadcast Message Server (BMS) Monitor tool to observe what messages are being sent by Development Manager, or other integrated tools, to instances of the CM class tool, and to identify which responses are being returned by the CM class tool.

E.1 Development Manager Pull-Down Menus for CMVC

There are two pull-down menus related to CMVC in the Development Manager window's menu bar. The CMVC menu provides actions taken with respect to specific files; the Windows menu allows access to the CMVC windows. Figure 135 on page 208 shows the CMVC pull-down menu.

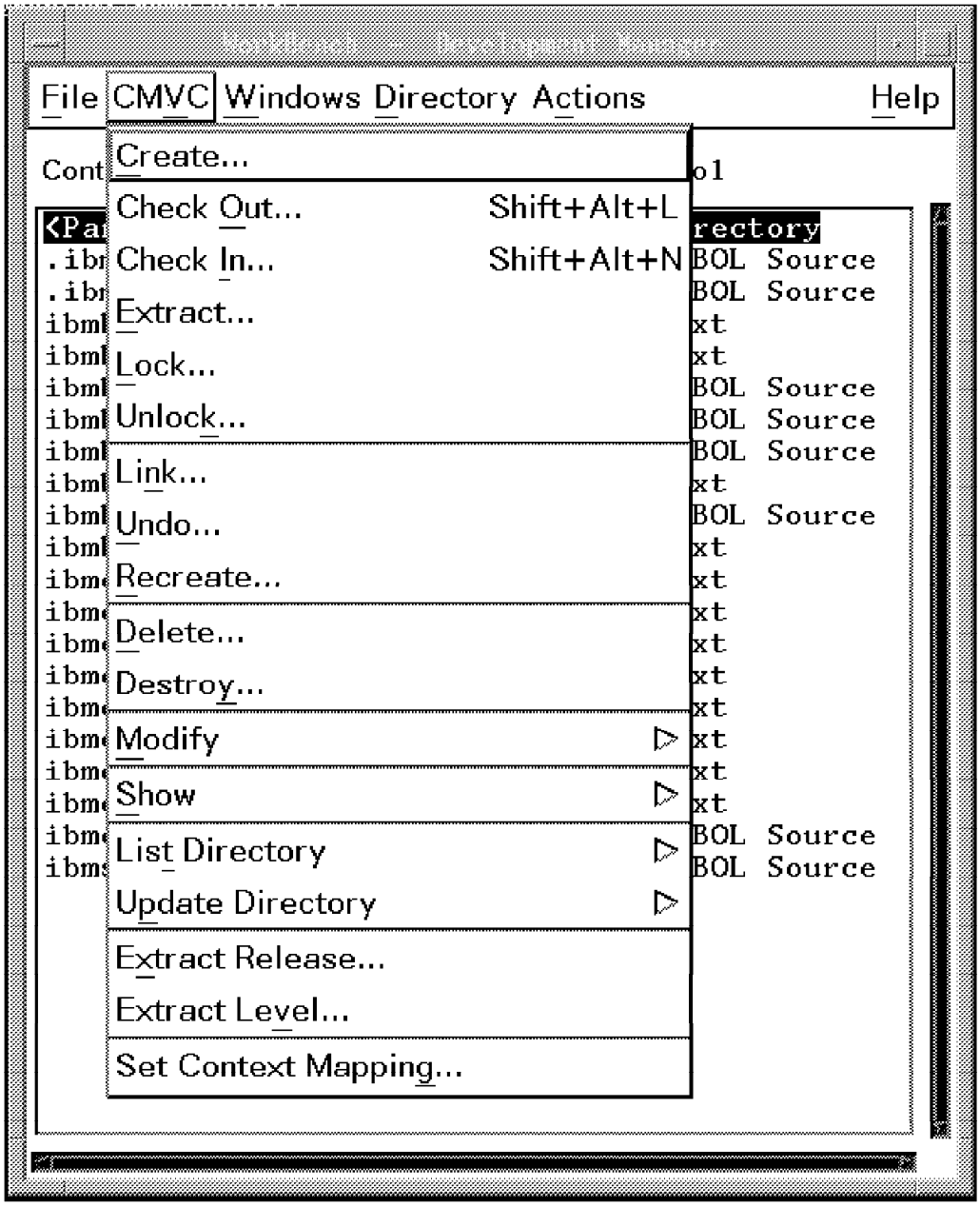

Figure 135. CMVC Pull-down Menu on Development Manager Menu Bar

These menus are described in great detail in *IBM CMVC User's Guide* . We recommend a careful study them.

E.2 Significance of Context Mappings

In SDE WorkBench/6000 "context mapping" associates the Development Manager data context directory with a particular CMVC component and release tuple. Several of these can be created; they are saved by CMVC in the `.cmvcrc` file in your home directory when you stop executing the CMVC GUI client. If a context mapping record has a path name, which overlaps the current Development Manager context directory, that matching part of the path name is stripped away from the path names of any selected files before they are passed

off as parameters to a CMVC command that is issued from the Development Manager CMVC pull-down menu, if that command involves the same component and release tuple associated with that context mapping. The purpose of this arrangement is so that end users can work in directory hierarchies that mimic the path names associated with files under CMVC control, and have the Development Manager prefill CMVC window fields (which require path and file names) with the correct path names. It is very important to master this concept and use context mappings, because CMVC can get confused if you add these path names by hand where they should be prefilled. Context mapping is described in more detail than we provide here in "The CM Class Messages" of "Using the Message-Integrated CMVC GUI" of *IBM CMVC User's Reference*

It is not recommended that users edit the *.cmvcrc* file. If you want to clear a single mapping default, clear the component and release fields from the Context Mapping dialog box when it is being prompted. However, if you want to delete some mappings, leaving others, or simply verify the mappings that you have defined, edit this file.

E.3 Implications of Host Scoping for CMVC

CMVC is not designed to be "network aware" in that it does not *know* it can concatenate the path name `/nfs/hostname` (using the data context's host name, parameter passed to it by Execution Manager) to the data context's path name parameter, to find a file on a remote host. The CMVC client merely compares the host parameter passed to it against, the host on which it is executing, and if they do not match, CMVC issues an error message.

We infer that CMVC designers made this design decision, because CMVC is a host scoped tool. But we do not think that being host scoped necessarily implies a tool is not network aware. Other tools, such as the Build Tool, which is network scoped, and the Development Manager, which is directory scoped, are network aware, and can locate remote data, assuming the proper file systems are exported and mounted. Host scoping simply means that only one instance of the tool can be invoked automatically by the Execution Manager per host.

If you have multiple CMVC clients installed on your network, and want to have CMVC access data on them, you must configure the CM class tool through your personal `.softinit` file or the system-wide `/usr/softbench/config/softinit` file to execute on whatever host is identified by the data context parameter passed to it. The value `%Host%` goes in the execution host field. When this is the case, then the Execution Manager starts up an instance of the CM tool on the remote host, if it recognizes a remote host in the data context being passed to CMVC. The CMVC client, executing remotely, binds to the local X server and display its results where they belong. CMVC starts up multiple instances of the CM tool under these circumstances, if different data hosts are indicated by subsequent user requests for CMVC actions.

If you have CMVC client only installed on the host where Execution Manager is running, then you must configure the CM class tool to execute only on the local host. To do this, place the value `%Local%` in the execution host field of the CM tool entry in the `.softinit` file or cause your system-wide `softinit` file to be updated this way. This forces Execution Manager to assume that only one instance of the CM tool should be started up, and to always start it on the same host it on which Execution Manager is executing.

When the execution host is set to %Local%, Execution Manager ignores the host portion of the data context. However The CMVC client does pay attention to the host portion. If you have asked it to deal with data on a remote host, it will balk. This can happen if you set the Development Manager to a path beginning with /nfs/hostname/....

If you really only have the CMVC client installed on the host on which your users execute Execution Manager, and you want to access remote files made available to your CMVC client machine through NFS, do not access them by setting the Development Manager context to a path name that begins with /nfs/hostname, or by explicitly naming remote data host. Development Manager is network aware, and infers from this path name that the data context is in fact on a remote host, and sets up the data context parameters accordingly. CMVC will then refuse your request (If CMVC were network aware, it could look to see if the data context were available at the default NFS path and access it there, but this is not the case as of CMVC Version 2, Release 1). Instead you should set up a link (whose path name does not include /nfs/hostname) to a directory mounted in this default NFS path, and set your Development Manager data context to this directory. Under these circumstances, the Development Manager is unaware that the data is remote and passes data context to CMVC as a local path name. Execution Manager then starts up a local CMVC instance.

The default execution host setting for CMVC is %Local%. This is briefly mentioned in "Broadcast Message Server" of *IBM CMVC User's Reference*. CMVC installation procedures appear to place a file, named softcmvcinit, in the /usr/softbench/config/softinitsrc/class-defaults file, but may not actually execute the /usr/softbench/etc/merge-init file, which merges all tool vendor contributions into /usr/softbench/config/softinit file, which is the system-wide configuration file. This is the recommended approach to tailoring your system-wide softinit settings. See "Customizing Tool Initiation" in *Installing IBM AIX SDE WorkBench/6000 and IBM AIX SDE Integrator/6000* for details on this file. You may be confused if you are not familiar with this process. Follow the directions given in "Installing the CMVC Clients" of *IBM CMVC Client Installation and Configuration*, which explicitly advise the system administrator to edit the /usr/softbench/config/softinit file directly.

E.4 CM Tool Messages

The SDE WorkBench/6000 integrated CMVC client may be activated through BMS messages by various tools that request files be checked in or out, releases be created or extracted, and context mapping be changed. It responds to the general tool messages sent by Execution Manager to start, stop. This message traffic can and should be observed if you, or your users, experience problems, or situations they do not understand, resulting from tool interaction. Refer "The Message Interface" in *Configuration Management Version Control User's Reference, Version 2 Release 1* if you need to determine which messages CMVC will respond to and what the parameters to the messages represents. Many problems may be caused by a misunderstanding on the part of the new SDE WorkBench/6000 user about the data and execution contexts involved with the tools being exercise in a distributed environment. Analysis of the message traffic readily identifies what data is being passed to what tool, and what precise action is being requested.

Appendix F. Source File and Program Identification with CMVC Keywords

This appendix introduces SCCS keywords and explains how CMVC interacts with them. Have you already tried the following AIX command?

```
what /usr/bin/chown
```

If you run this command, you should get a result similar to what is shown in Figure 136. This figure shows the list of source files compiled and linked together to produce the AIX **chown** command with the following information for each source file:

1. The CMVC identifier
2. The version identifier
3. The component managing the file: bos and cmdque
4. The AIX release to which the file is linked: bos
5. The level in which the version file has been integrated: 9219320b and 9238320
6. The date and time of the latest source file check-in.

```
11  1.20  com/cmd/que/qadm/chque.c, bos, bos320 4/30/91 08:28:55
61  1.44  com/cmd/que/libque/common.c, cmdque, bos320, 9219320b 3/9/92 14:12:44
66  1.36  com/cmd/que/libque/qdjdf.c, cmdque, bos320, 9238320 8/25/92 16:08:58
64  1.16  com/cmd/que/libque/jobnum.c, bos, bos320 6/3/91 12:03:52
70  1.10  com/cmd/que/libque/tcp.c, bos, bos320 6/3/91 12:02:10
20  1.21  com/cmd/que/qadm/qccom.c, bos,bos320 8/19/91 14:16:11
```

Figure 136. Result of the AIX what Command

If you would like to do the same for your project's executables then read the remainder of this appendix.

The following list describes the keywords we have used for our project. Refer to *IBM CMVC User's Reference* for a complete list of all SCCS keywords supported by CMVC.

Keywords	Meanings
%M%	Numeric name of the file that CMVC server defined
%D%	Date of the latest check-in. Format date is Year/Month/Day.
%H%	Date of the latest check-in. Format date is Month/Day/Year.
%T%	Time of the latest check-in. Format time is Hours:Minutes:Seconds.
%W%	File path name, component name, release name, level name. The level name appears only if file version is integrated in a level.
%Z%	The 4-characters string @(#) recognized by the **what** command as the beginning and the end of a "**what**" string in the binary file.

To insert CMVC keywords into a program source file, you should declare a variable with the string or character array type and initialize it with the keywords you have chosen. Figure 137 on page 212 shows the variable declarations

inserted in our C source file. The C keyword, *static*, allows us to use the same variable identifier, *CMVC_ID*, for any C source file (sort of local variable).

```
static char CMVC_ID[]="%Z% %M% %I% %W% %D% %T%"
```

Figure 137. CMVC Keywords into a C Source File

In our project, the CMVC keywords were automatically inserted in the C source file by a user exit program when the file was created in CMVC. That user exit program is described in Appendix C, "User Exit Samples and Suggestions" on page 183.

Whenever a source file is extracted from CMVC, these keywords are replaced by the current values they represent. However, when a source file is checked out the keyword expansion is suppressed, so the keywords will still be in the version that is checked in.

Appendix G. Appendix: Setting Up NetLS

This appendix gives a short and easy procedure for setting up NetLS.

G.1.1.1 NetLS Set Up Procedure

After NetLS has been installed on the selected NetLS license servers, it should be configurated on each server as follows:

1. Get the target ID. The target ID of the NetLS license server is mandatory to obtain a password. To get it issue the command:

```
/usr/lib/netls/bin/ls_targetid
```

   The result of that command should look like the following:

```
NetLS Target ID
---------------
274634

 LLA ID
---------
9712C91
```

   Note the NetLS Target ID, is *274634* in our example

   Note: Repeat the previous step for each NetLS server before going onto the next step

2. Obtain the passwords from IBM. Send to IBM, the NetLS Target IDs and the number of licences you want for each NetLS server. IBM returns a password for each server

3. Initialize the NetLS server. You should login as root before performing the following actions.

   a. Create a shell script to initialize the server by executing the command:

```
/usr/lib/netls/conf/netls_config
```

   b. Answer **N** to the question asking "Do you want to use existing database?"
   c. Select **2** ("using default cell")
   d. Answer **Y** to the question asking "Do you want to start llbd and glbd at system startup time?."
   e. Initialize the server with the shell script created previously, by executing the following command

```
/usr/lib/netls/conf/netls_first_time
```

   The /etc/inittab file is updated to start the NetLS daemons (**llbd**, **gldb**, and **netlsd**) when the server boots up.
   f. The **netlsd** does not come up so you have to start it manually by entering the command:

```
/usr/lib/netls/ark/bin/netlsd
```

4. Register the information obtained from IBM into NetLS. You should login as root before performing these actions.

   a. Enter the vendor password by executing the following command:

```
/usr/lib/netls/bin/ls_admin -a \
-v "vendorName" \
vendorID \
vendorPassword
```

   b. Enter the product password by executing the following command:

```
/usr/lib/netls/bin/ls_admin -a \
-p "vendorName" \
productName \
productPassword \
version
```

   Check carefully the passwords before pressing on the Enter key.

5. Check the NetLS set up

   a. Check the registration in NetLS for both the vendor and product by using the following command:

```
/usr/lib/netls/bin/ls_admin -s \
-p vendorName productName
```

   b. The result of that command should look like the following:

```
LS_ADMIN Version 2.0.1 (GR1.1.0) IBM-AIX
(c) Copyright 1991,1992,1993, Hewlett-Packard Company, All Rights Reserved
(c) Copyright 1991,1992,1993, Gradient Technologies Inc., All Rights Reserved

Server: ip:baffin.sanjose.ibm.com
    socket address family  ip
    socket address         9.113.44.201
    socket port            1056
    target type            IBM/AIX
    target id              131137
Vendor: CASE
    vendor id         5e1d4ffe2f74.02.09.15.11.06.00.00.00
Product: CMVC [2.1.0]
    id              1234
    Licenses:
expired:         5 Concurrent Access from 1994/02/08 to 1995/02/07, timestamp 760748567
          multi-use rules:  same user group node.
```

   c. Check if NetLS is operational with the following command issued from a machine on which the CMVC client is installed:

```
lpp/cmvc/bin/Report -testServer -family foo -become goo
```

   The NetLS setup is OK if the following message comes up:

```
0010-247 The host name, foo, for the CMVC server cannot be resolved.
         Verify that the CMVC server's host name and address are included
         in the CMVC client's '/etc/hosts' file or the data files of the
         name server.  If the host name and address appear in either
         of these files, the network or name server may be experiencing
         a temporary problem.
         If the problem persists, contact the family administrator.
```

If the setup is not OK, you get the following message:

```
0010-877 Cannot connect to NetLS server.

        To solve the problem, contact the CMVC family administrator.
```

6. Clean up the NetLS configuration on the NetLS server if you have problems. The following procedure enables you to perform this cleanup.

   DO NOT apply this procedure if NetLS is used by other products.

   a. Stop the NetLS daemons by issuing the following commands:

```
root@server\>stopsrc -s llbd
root@server\>stopsrc -s glbd
root@server\>stopsrc -s netlsd
```

   b. Remove the /tmp/llbdbase.dat file, if it exists

   c. Remove the /etc/ncs/glb.e, /etc/ncs/glb.p, /etc/ncs/glb_log, /etc/ncs/glb_obj.txt, and /etc/ncs/glb_site.txt files, if they exist

   d. Remove the /usr/lib/netls/conf/cur_db, /usr/lib/netls/conf/lic_db, /usr/lib/netls/conf/lic_db.bak, /usr/lib/netls/conf/cur_db.bak, and /usr/lib/netls/conf/log_file files, if they exist

   e. Repeat the setup procedure. If the NetLS configuration fails again, check the target ID, and then contact IBM.

Appendix H. Tailoring CMVC Windows for Different Types of Users

This appendix describe some CMVC client customization based on the roles of certain people, such as project manager, developer, and builder. These customizations use CMVC reporting capabilities and additional programs to:

- Calculate some statistics and metrics
- Display set of data with certain properties
- Automated and simplify the problem tracking process.

CMVC implements some processes to manage change control and problem tracking. Like all the processes, the CMVC processes are repeatable and they can be automated by using the CMVC command-line interface from shell programs.

CMVC stores project data in a relational database you can build queries to find a set of elements that have certain properties. The results of the queries can be used to calculate statistics and metrics.

H.1 Customization Example for Project Manager

A project manager is essentially interested by the project progress and statistics. For example, the project progress can be monitored through the states of defects, features, and levels, as well as the number of levels, files, and locked files (checked out files). Figure 138 on page 218 illustrates a hypothetical CMVC - Tasks window customized for a manager and Figure 140 on page 220 shows the `.cmvcrc` file associated with this window.

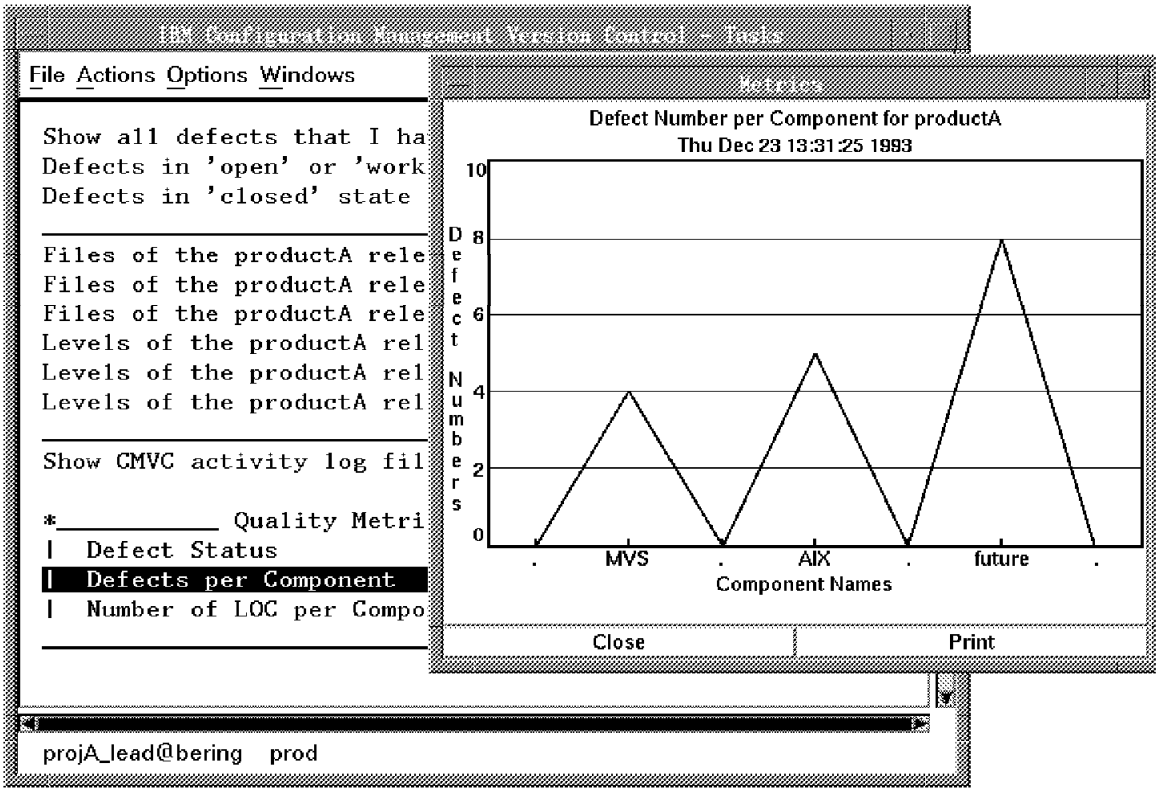

Figure 138. Customized CMVC - Tasks Window for Project Manager

The two first tasks listed in of the CMVC - Tasks window enables you to display the defects you have to act on. The third one gives you all defects in the *closed* state. The queries provide you with lists of files and levels for each release of your product. The *Show CMVC activity log file*, (**1**) in Figure 140 on page 220, shows you the log of all CMVC commands performed by any of your team members. Figure 23 on page 38 shows a part of the *prod* family log file. Based on the content of the log file, you can compute some statistics, such as:

- How many files created per month?
- How many defects opened per month?
- How many defects closed per month?
- How many defects you have opened per month?
- How many tests rejected on AIX_Release_1?
- How many unauthorized actions performed?

Figure 139 on page 219 gives a simple shell script used to compute the statistics of the above list.

```
#!/bin/ksh
set -A YEAR January February March April May June July August September October November December
set -A MONTHN 01 02 03 04 05 06 07 08 09 10 11 12
let I=0
for MONTH in ${YEAR[*]}
do
    RANG=monthN[$i]
    print "***************** $MONTH ******************"
    print -n "Number of Defects Opened= "
    grep "DefectOpen,SUCCESS,$RANG" $HOME/log |wc -l
    print -n "Number of Files Created= "
    grep "FileAdd,SUCCESS,$RANG" $HOME/log |wc -l
    print -n "Number of Defects Closed= "
    grep "DefectVerify,SUCCESS,$RANG" $HOME/log |wc -l
    print -n "Number of Defects I Opened= "
    grep "DefectOpen,SUCCESS,$RANG.*projA_lead," $HOME/log |wc -l
    print -n "Number of Tests Rejected on the AIX_Release_1= "
    grep "TestReject,SUCCESS,$RANG.*AIX_Release_1" $HOME/log |wc -l
    print -n "Number of Unauthorized Actions Performed="
    grep "Transaction.*UNAUTHORIZED,$RANG" $HOME/log |wc -l
    let I=I+1
done
```

Figure 139. Shell Script to Compute Some Statistics

In Figure 140 on page 220, statis program **2** calculates metrics for which the log file is not sufficient. This program use the CMVC **Report** command to extract data from CMVC according to certain input parameters, formats extracted data, and finally displays a graphical representation of the metrics, as shown in Figure 138 on page 218. Our project manager needs only to click twice on the specific line in the CMVC - Tasks window to obtain this information. This program was unique to our project, but you can develop a similar programs and trigger their execution easily from tailored CMVC - Task window entries.

```
*family:        prod@bering@1222
*userID:        projA_lead
*directory:     /home/aixcase4
*task1_description:    Show all defects that I have to verify
*task1_query:    userLogin in ('projA_lead') and state in ('ready')
*task1_window:    verification
*task2_description:    Defects in 'open' or 'working' state
*task2_query:    ownerLogin in ('projA_lead') and (state in ('open') or state in ('working'))
*task2_window:    defect
*task3_description:    Defects in 'closed' state
*task3_query:    state in ('closed')
*task3_window:    defect
*task4_description:    _____________________________________________
*task4_query:
*task4_window:    information
*task5_description:    Files of the productA release AIX_Release_1
*task5_query:    releaseName in ('AIX_Release_1')
*task5_window:    file
*task6_description:    Files of the productA release AIX_Release_1
*task6_query:    releaseName in ('AIX_Release_1')
*task6_window:    file
*task7_description:    Files of the productA release OO_Version_1
*task7_query:    releaseName in ('OO_Version_1')
*task7_window:    file
*task8_description:    Levels of the productA release AIX_Release_1
*task8_query:    releaseName in ('AIX_Release_1')
*task8_window:    level
*task9_description:    Levels of the productA release AIX_Release_1
*task9_query:    releaseName in ('AIX_Release_1')
*task9_window:    level
*task10_description:    Levels of the productA release OO_Version_1
*task10_query:    releaseName in ('OO_Version_1')
*task10_window:    level
*task11_description:    ______________________________________________
*task11_query:
*task11_window:    information
*task12_description:    Show CMVC activity log file
*task12_query:   rexec bering cat /production/audit/log > $HOME/log  1
*task12_window:    information
*task13_description:
*task13_query:
*task13_window:
*task14_description:    *___________ Quality Metrics _________*
*task14_query:
*task14_window:    information
*task15_description:    |  Defect Status                      |
*task15_query:    statis defect status  &                          2
*task15_window:    information
```

Figure 140 (Part 1 of 2). Part of Project Manager's .cmvcrc File

```
*task16_description:    |  Defects per Component              |
*task16_query:    statis defect component &
*task16_window:    information
*task17_description:    |  Number of LOC per Component       |
*task17_query:    statis file loc component &
*task17_window:    information
*task18_description:    ____________________________________
*task18_query:
*task18_window:    information
```

Figure 140 (Part 2 of 2). Part of Project Manager's .cmvcrc File

H.2 Customization Example for Developer

Most of time, as a developer, you are responsible for fixing defects or implementating a feature by changing or creating files. Figure 141 on page 222 displays the CMVC - Tasks window customized for a developer and Figure 142 on page 222 shows the `.cmvcrc` file generated when customizing it. As you can see, you can build queries by using SQL (**1**) if you know the CMVC tables and views. The tables and views are described in the *IBM CMVC User's Reference*. You can also get the description by viewing the files, `tables.db` and `views.db`, located in the `/usr/lpp/cmvc/install` directory on the server machine.

Note:

Those file names are provided by the CMVC server for ORACLE. For DB2/6000, the file names are, `tables.db2` and `views.db2`. For INFORMIX, the file names are, `tables.ifx` and `views.ifx`. For SYBASE, the file names, are `tables.syb` and `views.syb`.

In Figure 142 on page 222, CMVC calls the `createTrack.sh` shell script, which allows you to fix a defect without being familiar with the CMVC fix process (**2**). Figure 145 on page 224 shows this shell script.

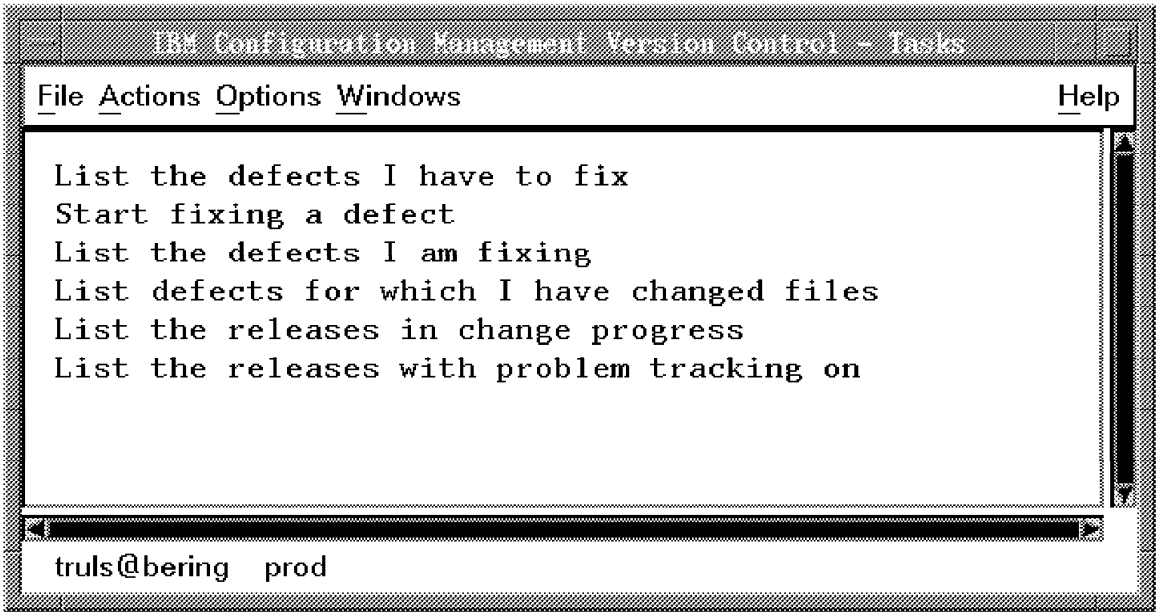

Figure 141. Customized CMVC - Tasks Window for a Developer

```
*family:        prod@bering@1222
*userID:        truls
*directory:     /home/aixcase2
*task1_description:    List the defects I have to fix
*task1_query:    state in ('working') and ownerLogin in ('ted') and id not in (select defectId from Tracks)
or id in (select defectId from Tracks  where state in ('fix'))        1
*task1_window:    defect
*task2_description:    Start fixing a defect
*task2_query:    ksh createTrack.sh truls > log 2> ./error&     2
*task2_window:    information
*task3_description:    List the defects I am fixing
*task3_query:    userLogin in ('truls') and state in ('fix')
*task3_window:    track
*task4_description:    List defects for which I have changed files
*task4_query:    state in ('active') and defectname in
(select name from defectview where ownerlogin='truls')
*task4_window:    fix
*task5_description:    List the releases in change progress
*task5_query:    name in (select releaseName from FixView where state in ('active'))
*task5_window:    release
*task6_description:    List the releases with problem tracking on
*task6_query:    relProcess not in ('no_track','prototype')
*task6_window:    release
```

Figure 142. Part of a Developer's .cmvcrc File

The createTrack.sh shell script, shown in Figure 145 on page 224, is called from the CMVC - Tasks window when you click twice on **Start Fixing a defect** or when you highlight this line and press the Return key.

In Figure 145 on page 224, you enter the defect number to be fixed from a dialog box (1), and then select **OK** to confirm your choice. If you do not know which defect you can select, select **Help** in the dialog box, and follow the instructions displayed in the help dialog box (this step is shown in Figure 143 on page 223). The program checks the existence, the state, and the ownership of the selected defect (2). The program then checks to see if the defect defect state is *working* using the **Report** on the view defectView. If the check fails, the program is aborted (3).

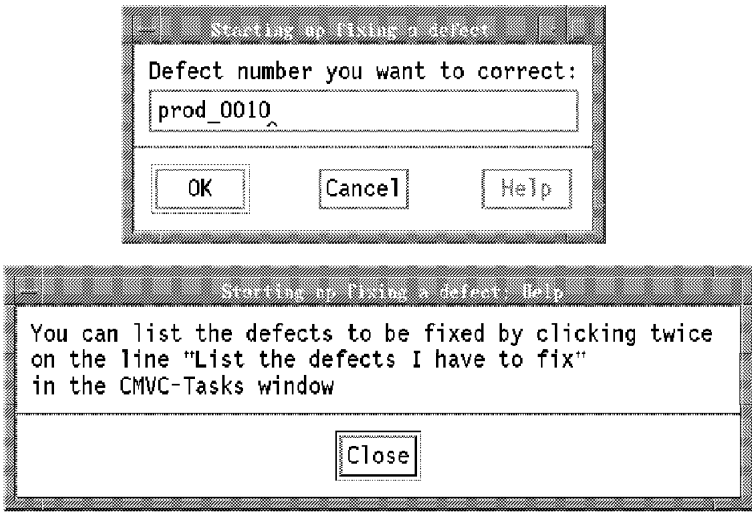

Figure 143. Getting the Number of Accepted Defects

In a second time (**4**) you enter one or more releases affected by the selected defect, from a dialog box, and then select **OK** to confirm the list. If you do not know which release, you can select **Help** in the dialog box and follow the instructions displayed in the dialog box. This step is shown in Figure 144. The program checks the process of each release of the list (**5**). The process must not be *prototype*, nor *no_track*, because these two processes do not include the *track* subprocess. This check uses the **Report** on the view releaseView. If the check fails, the program is aborted.

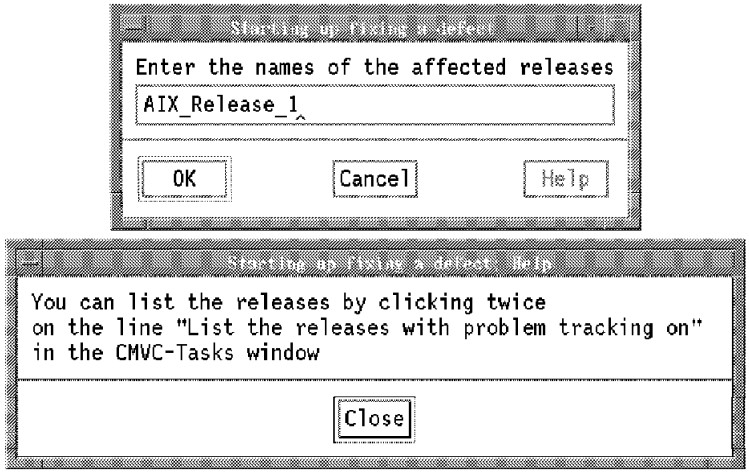

Figure 144. Getting the List of Affected Releases

Finally, the program creates a track for each release (**6**).

```
#!/bin/ksh
### Function definition
getPar()
{
    PAR=$(mgti -fn rom17 -prompt "$1" -helpMsg "$2" \
         -helpTitle "$3: Help" -title "$3")
    [[ -z "$PAR" ]] && exit 2
}
displayInfo()
{
    mfyi  -fn rom17 -helpMsg "$2" -helpTitle "$3: Help" -title "$3" "$1"
}
OWNER=$1

TITLE="Starting up fixing a defect"
PROMPT="Defect number you want to correct:"

HELP="You can list the defects to be fixed by clicking twice
on the line \"List the defects I have to fix\"
in the CMVC-Tasks window"

## get defect number
getPar "$PROMPT" "$HELP" "$TITLE"                                    1

## check defect status and ownership
info=$(Report -raw -view defectView  \                               2
       -where "name in ('$PAR') and  state in ('working') \
       and ownerLogin in ('$OWNER') ")

## test result of previous query
print $info | grep $PAR > /dev/null 2>&1
if [[ $? -ge 1 ]]
then
   INFO="The defect $PAR has not existed
         or not in working state"
   displayInfo "$INFO" "$HELP" "$TITLE"
   exit 1                                                            3
fi
DEFECT=$PAR
```

Figure 145 (Part 1 of 2). Shell Script to Create a Track: createTrack.sh

```
PROMPT="Enter the names of the affected releases"
HELP="You can list the releases by clicking twice
on the line \"List the releases with problem tracking on\"
in the CMVC-Tasks window"
getPar "$PROMPT" "$HELP" "$TITLE"                                    4
set -A RELEASES $PAR
CONTINU=FALSE
let I=0
## check the release process
for RELEASE in ${RELEASES[*]}                                        5
do
    ## query for checking if track subprocess is turned on
    info=$(Report -raw -view releaseView -where "name in ('$RELEASE') \
           and relProcess not in ('no_track','prototype') ")
    print $info | grep $RELEASE > /dev/null 2>&1
    if [[ $? -ge 1 ]]
    then
       INFO="The release $RELEASE has not existed
             or has not the track subprocess is turned on"
       displayInfo "$INFO" "$HELP" "$TITLE"
    else
       CONTINU=TRUE
       REL[$I]=$RELEASE
       let I=I+1
    fi
done
## create a track for each release
if [[ $CONTINU = "TRUE" ]]
then                                                                 6
    INFO=$(Track -create -defect $DEFECT -release ${REL[*]} -verbose)
    HELP=""
    displayInfo "$INFO" "$HELP" "$TITLE"
fi
```

Figure 145 (Part 2 of 2). Shell Script to Create a Track: createTrack.sh

H.3 Customization Example for Builder

Most of time, the build process is described by the `Makefile` file used by the AIX **make** command. This file details a series of commands performing different actions to build your application. The build process is not integrated in CMVC but if you want to automate it, create a `Makefile` file for each release. This allows you to manage your build process under CMVC control.

For the remote distributed build, if the target machines run TCP/IP and NFS, you can use remote execution and share the target file systems through NFS. In this case, make one or more `Makefile` files use remote execution of build commands. Figure 146 on page 226 displays a CMVC - Tasks window customized for our builder working on the release *AIX_Release_1*, and Figure 147 on page 226 shows the `.cmvcrc` file generated when customizing it.

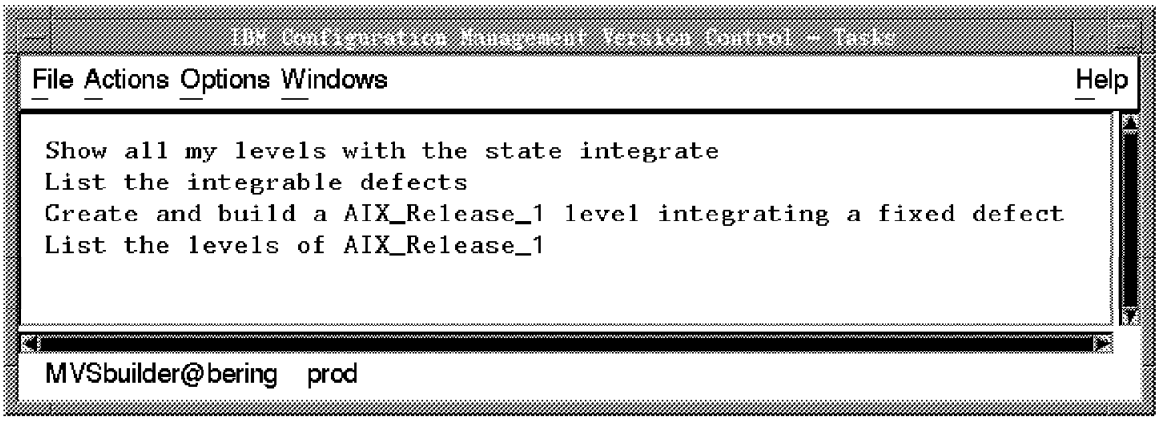

Figure 146. Customized CMVC - Tasks Window for Builder

```
*family:        prod@bering@1222
*userID:        MVSbuilder
*directory:     /home/aixcase4
*task1_description:    Show all my levels with the state integrate
*task1_query:     userLogin in 'MVSbuilder' and state in 'integrate'
*task1_window:    level
*task2_description:    List the integrable defects of AIX_Release_1
*task2_query:    releaseName in ('AIX_Release_1') and state in ('integrate')
*task2_window:    track
*task3_description:    Create and build a AIX_Release_1 level integrating a fixed defect
*task3_query:    ksh createLevelPar.sh AIX_Release_1 /ad/ProductA/AIX_Release_1 > log 2> error&
*task3_window:    information
*task4_description:    List the levels of AIX_Release_1
*task4_query:    name like '%' and releaseName in ('AIX_Release_1')
*task4_window:    level
```

Figure 147. Part of Builder's .cmvcrc File

The createLevelPar.sh shell script, shown in Figure 151 on page 229, is called from the CMVC - Tasks window when you click twice on **Create and build a AIX_Release_1 level integrating a fixed defect** or when you highlight this line and press the Return key.

In Figure 151 on page 229, first you enter the defect number to be integrated into a level from a dialog box (**1**), then select **OK** to confirm your choice. If you do not know which defects you can select, select **Help** in the dialog box and follow the instructions displayed in the dialog box. This step is shown in Figure 148 on page 227. The program checks the existence and the state of the track associated with the chosen defect and your default release, *AIX_Release_1* (**2**). The track state must be *integrate*. This check uses the **Report** on the view trackView. If the check fails, the program is aborted (**3**).

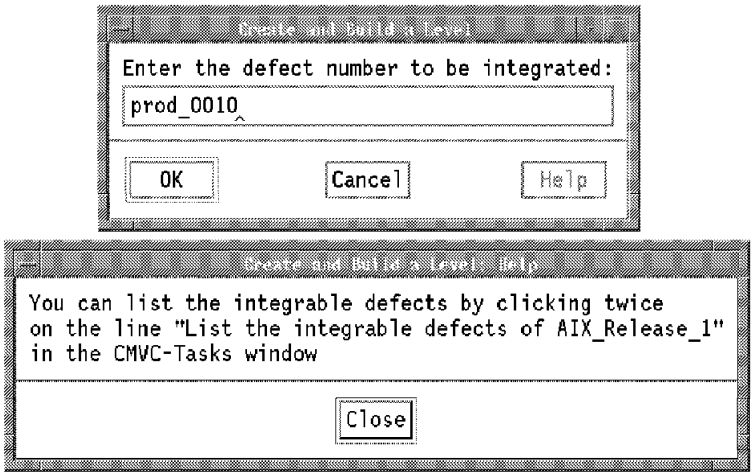

Figure 148. Getting Fixed Defect Number

In a second time, you enter the name of the level you want to create to integrate the fixed defect from a dialog box (**4**), and then select **OK** to confirm the level name. If you do not know which level name you can enter, select **Help** in the dialog box and follow the instructions displayed in the dialog box. This step is shown in Figure 149. The program checks the nonexistence of level name entered for the default release (**5**). This check uses the **Report** on the view `levelView`. If the check fails, the program is aborted (**6**).

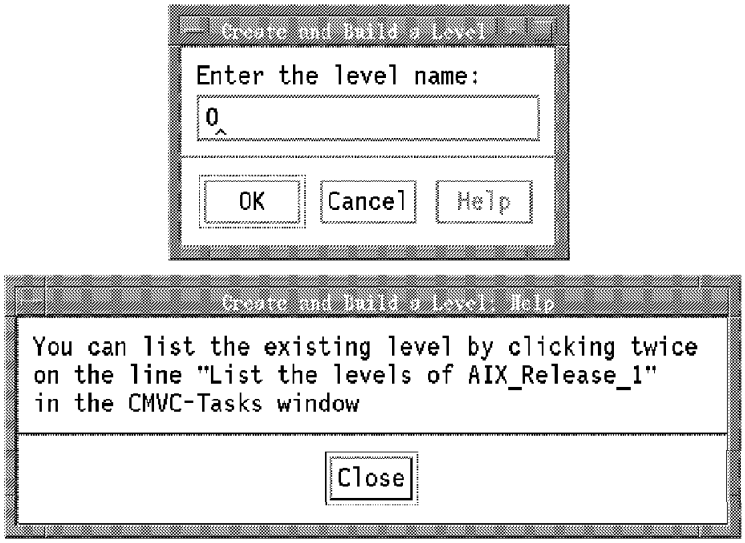

Figure 149. Getting the Level Name

Then, the program creates the level and adds the track as level member in through another shell script named `createLevel.sh` (**7**), whose content is shown in Figure 152 on page 231.

Then, the program extracts (delta extraction) the level by calling another shell script named `buildLevel.sh` (**8**), described in Figure 153 on page 231.

Finally, the program is waiting for the result of the level build (**9**). If you select **OK** in the dialog box shown in Figure 150 on page 228, it means the build has been successfully executed, the program automatically commits and completes the level with the completeLevel.sh shell script shown in Figure 154 on page 232 (**10**).

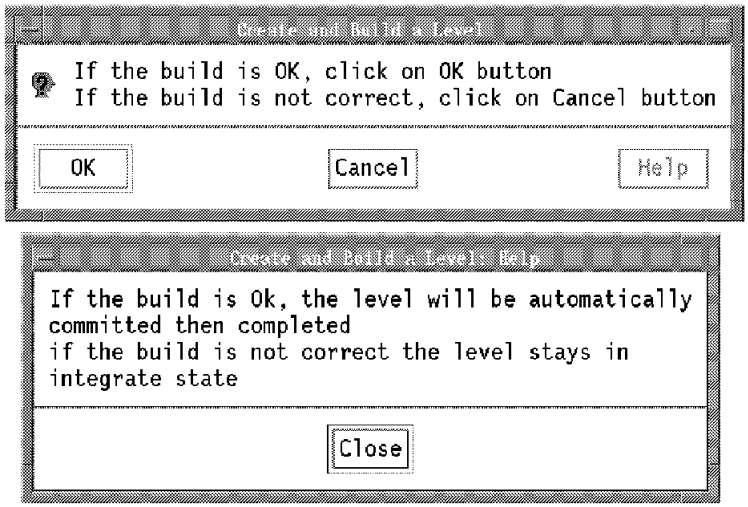

Figure 150. Accepting a built Level

```
#!/bin/ksh
getPar()
{
    PAR=$(mgti -fn rom17 -prompt "$1" -helpMsg "$2" \
         -helpTitle "$3: Help" -title "$3")
    [[ -z "$PAR" ]] && exit 2
}
displayInfo()
{
    mfyi  -fn rom17 -helpMsg "$2" -helpTitle "$3: Help" -title "$3" "$1"
}
confirm()
{
    myni  -fn rom17  -helpMsg "$2" -helpTitle "$3: Help"  -title "$3"  "$1"
}

RELEASE=$1
BUILDDIR=$2
MAKELOC=$BUILDDIR"/"$3
TITLE="Create and Build a Level"
PROMPT="Enter the defect number to be integrated:"
HELP="You can list the integrable defects by clicking twice
on the line \"List the integrable defects of AIX_Release_1\"
in the CMVC-Tasks window"
getPar "$PROMPT" "$HELP" "$TITLE"                                    1
## Check track existence and status
info=$(Report -raw -view trackView \                                 2
        -where "defectName in ('$PAR') \
       and state in ('integrate') and releaseName in ('$RELEASE')")
echo $info | grep $PAR > /dev/null 2>&1
if [[ $? -ge 1 ]]
then
   INFO="A track has not existed for the defect \"$PAR\"
   and for the release $RELEASE  or not in integrate state
   or the defect has not existed"
   displayInfo "$INFO" "$HELP" "$TITLE"
   exit 1                                                            3
fi
DEFECT=$PAR
```

Figure 151 (Part 1 of 2). Shell Script to Proceed a Level: createLevelPar.sh

```
TITLE="Create and Build a Level"
PROMPT="Enter the level name:"
HELP="You can list the existing level by clicking twice
on the line \"List all the levels of AIX_Release_1\"
in the CMVC-Tasks window"
getPar "$PROMPT" "$HELP" "$TITLE"                                    4
## Check level existence
info=$(Report -raw -view levelView -where "name in ('$PAR') \        5
       and releaseName in ('$RELEASE')")

echo $info | grep $PAR > /dev/null 2>&1
if [[ $? -ge 0 ]]
then
   INFO="The level is already existing
   for the release $RELEASE"
   displayInfo "$INFO" "$HELP" "$TITLE"
   exit 1                                                            6
fi
LEVEL=$PAR
aixterm -title "Creating a Level" \                                  7
-e ksh createLevel.sh $DEFECT $LEVEL $RELEASE

aixterm -title "Extracting and Building a Integrated Level" \        8
-e ksh buildLevel.sh $LEVEL $RELEASE $BUILDDIR $MAKELOC

INFO="If the build is OK, click on OK button
If the build is not correct, click on Cancel button"
HELP="If the build is Ok, the level will be automatically
committed then completed
if the build is not correct the level stays in integrate state"
confirm "$INFO" "$HELP" "$TITLE"
[[ $? = 2 ]] && exit 2                                               9

aixterm -title "Completing a Built Level" \                          10
-e ksh completeLevel.sh $LEVEL $RELEASE $BUILDDIR $MAKELOC &
```

Figure 151 (Part 2 of 2). Shell Script to Proceed a Level: createLevelPar.sh

The shell script illustrated in Figure 152 on page 231 is invoked from the createLevelPar.sh program, described above. This shell script calls the CMVC **Level** command to create a level with the default type of production (1), and the **levelMember** command to add the created level in the selected track (2).

```
#!/bin/ksh
DEFECT=$1
LEVEL=$2
RELEASE=$3

printf "Creating the level $LEVEL for the release $RELEASE\n"
printf "to integrate the defect $DEFECT\n"
Level -create $LEVEL -release $RELEASE -type production -verbose    1

printf "Integrating the defect $DEFECT into the level $LEVEL\n"
LevelMember -create -level $LEVEL -release $RELEASE \               2
            -defect $DEFECT -verbose

print "Press Enter to continue..."
read x
```

Figure 152. Shell Script to Create a Level with A Track: createLevel.sh

The shell script illustrated in Figure 153 is invoked if the build has been successfully executed, by createLevelPar.sh program, described above. This shell script uses the CMVC **Level** command to extract (1) (delta extraction because the **-full** is not used) the integrated level. The **make** program (2), automatically builds the extracted level in the build directory specified by the variable MAKELOC.

```
#!/bin/ksh
LEVEL=$1
RELEASE=$2
BUILDDIR=$3
MAKELOC=$4
printf "Extracting the level $LEVEL of the release $RELEASE\n"
Level -extract $LEVEL -full -release $RELEASE \                 1
       -node $(hostname) -root "$BUILDDIR" -uid $(id -u) -gid $(id -g) -verbose

printf "Building the $LEVEL of the release $RELEASE\n"
cd $MAKELOC
make                                                            2
if [[ $? = 0 ]]
then
   prog=$(grep "PROGRAM.*=" Makefile)
   ${prog:##* }
fi
cd -
print "Press Enter to continue..."
read x
```

Figure 153. Shell Script to Extract and Build a Level: buildLevel.sh

The shell script illustrated in Figure 154 on page 232 is invoked if the build has been successfully executed, by the createLevelPar.sh program, described above. This shell script uses the CMVC **Level** command to commit and complete the built level (1). The **make** program (2) automatically installs the result of the build in the target directory, specified by the variable, INSTALLDIR.

```
#!/bin/ksh
LEVEL=$1
RELEASE=$2
BUILDIR=$3
MAKELOC=$4

INSTALLDIR=$HOME/bin

printf "The build is ok, the level $LEVEL will be committed then completed\n"

Level -commit $LEVEL -release $RELEASE -verbose       1
Level -complete $LEVEL -release $RELEASE -verbose

# install the executable in $HOME/bin
cd $MAKELOC
make DEST=$INSTALLDIR install all                     2
cd -
print
print
print "Press Enter to continue..."
read x
```

Figure 154. Shell Script to Commit and Complete a Level: completeLevel.sh

Glossary

Glossary terms are defined as they are used in this manual. If you cannot find the term for which you are looking, refer to *IBM Dictionary of Computing*

absolute path name. A directory or a file expressed as a sequence of directories, followed by a file name beginning from the root directory.

access list. A CMVC object that controls access to development data. A list of user ID-authority group pairs attached to a component, designating users and the corresponding authority access they are being granted for all objects managed by this component or any of its descendants. It also contains the user ID-authority group pairs designating users who are restricted from performing actions at a specific component.

action. A task performed by the CMVC server and requested by a CMVC client. A CMVC action corresponds to issuing one CMVC command.

AIC. See *Advance interface composer*.

Advance Interface Composer (AIC). An AIX application development tool in the category known as “GUI builders.” It is used to design and implement an OSF/Motif GUI for an application.

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.

approver. A user who approves file changes required to resolve a defect or implement a feature in a release.

approver list. A list of CMVC user IDs attached to a release, representing the users who must approve file changes required to resolve a defect or implement a feature in that release.

ASCII. The standard coded character set using 7-bit characters (8th bit for parity) used widely on non-IBM mainframe computers.

authority. The right to access development objects and perform CMVC commands. See also *access list*, *base authority*, *explicit authority*, *implicit authority*, *restricted authority*, and *superuser privilege*.

base authority. The set of actions granted to a user whenever a user ID is created in a CMVC family.

base file name. The name assigned to the file outside of the CMVC server environment, excluding any directory names.

batch program. A program that reads its input from a file or device and writes its output to a file or device without the interaction of a user.

BMS. See Broadcast Message Server

Broadcast Message Server (BMS). A facility that coordinates the SDE WorkBench/6000 or HP SoftBench tools. Messages from tools are sent to the Broadcast Message Server which routes messages to other tools.

callback. C code that is associated with a widget or gadget which is executed when a specified event occurs. For example, a push button has an activate callback, which is executed when it is selected.

change control. The process of limiting and auditing changes to files through the mechanism of checking files in and out of a central, controlled storage location. Change control for an individual release can be integrated with problem tracking by specifying a process for that release that includes the track subprocess.

check in. The return of a CMVC file to version control.

check out. The retrieval of a revision of a CMVC file from version control.

child component. All components in each CMVC family, with the exception of the root component, must be created in reference to an existing component. The existing component is referred to as the parent component, while the new component becomes known as the child component. A parent component can have more than one child component. See also *component*.

client. A program that requests services from another program. A client program may execute on the same workstation as the server program, or on another workstation connected to the server workstation by means of a local area network (LAN). Contrast with *server*.

CLIST. A file containing a sequence of commands that serves as a mechanism on MVS for starting a COBOL program.

CMVC. See Configuration Management Version Control

CMVC client. The CMVC program that requests services of a CMVC server. Sometimes this term is used to refer to the workstation on which the CMVC client program has been installed. CMVC client programs can be installed on a variety of workstations executing operating systems from several vendors.

CMVC server. A CMVC program that manages file, release, and configuration data and services requests from CMVC clients. Sometimes the term is used to refer to the workstation or computer on which the CMVC server program is installed.

Configuration Management Version Control. An IBM product that provides configuration management, release management, problem tracking, change control, and version control functions for application development.

command. A request to perform an operation or run a program from the command-line interface. In CMVC, a command consists of the command name, one action flag, and zero or more attribute flags.

common file. A CMVC file that is contained in two or more releases and the same version of the file is the current version for those releases. See also *shared file*.

component. A CMVC object that simplifies project management, organizes project data into structured groups, and controls configuration management properties. Component owners can control access to development data (see *access list*) and configure notification about CMVC actions (see *notification list*). Components exist in a parent-child hierarchy, with descendent components inheriting access and notification information from ancestor components.

configuration management. The process of identifying, managing, and controlling software modules as they change over time.

context. A description of a data file or directory in the form *host dir file*, representing host name, directory path, and file name, which is used by SDE WorkBench/6000 and integrated tools. See also *tool context* and *message context*.

current directory. The default directory prepended to any relative path name or file name by the UNIX shell before using the file name or passing it on to any other programs, such as commands. Also known as the *present working directory* or *pwd*,

daemon. See *daemon process*.

daemon process. Daemon processes provide services that must be available at all times to more than one task or user. They are processes which run as a background task on UNIX.

database. A systematized collection of data that can be accessed and operated upon by a data processing system for a specific purpose.

defect. A CMVC object used to formally report a problem. The user who opens a defect is the defect originator.

delete. A CMVC action that results in the database record for a CMVC object, such as a file or user ID, being marked so that it is unusable by CMVC commands and queries. Most delected objects can be re-created.

destroy. A CMVC action that removes the database record for a CMVC object. The only CMVC object that can be destroyed is a file. Destroying a file removes the file record from the database on the CMVC server. although a destroyed file cannot be re-created, it will appear as part of an extracted level.

directory file list. A list of files and subdirectories of the current working directory displayed in the main window of SDE WorkBench/6000′s Development Manager program.

downsize. Migrating a mainframe application to a midrange or desktop computer. Sometimes also referred to as *rightsizing* in marketing literature.

EBCDIC. A coded character set of 256 8-bit characters used on IBM and other mainframes.

encapsulation. A program you write using, SDE Integrator/6000, that makes an application development tool appear, behave, and communicate like other tools that are integrated with SDE WorkBench/6000.

end user. A person using a computer program or an application.

environment. In CMVC, a label representing a user-defined testing domain for a particular release which is entered in an environment list entry. Also used as a field in the Defect record representing the environment where the problem occurred.

environment list. A CMVC object, associated with a track, which is a list of records containing two fields, one specifying the environment and the other a tester.

explicit authority. The ability of a particular CMVC user ID to perform an action against a CMVC object because that user ID was granted the authority to perform that action.

extract. A CMVC action performed on a file, level, or release that results in particular versions of the named file, or files associated with the level or release, being copied to a specified directory.

family. In CMVC, a logical organization of related development data. A single CMVC server can support multiple families. The data in one family cannot be accessed by CMVC IDs defined for another family.

family administrator. A CMVC user ID who is responsible for all non system-related tasks for one or more CMVC families such as planning, configuring, and maintaining the CMVC environment and managing user access to those families.

feature. A CMVC object used to formally request a functional addition or enhancement. The user who opens a feature is the feature originator.

file. A collection of data that is stored by the CMVC server and retrieved by a path name. Any text or binary file used in a development project can be created as a CMVC file. For example, source code, executable programs, documentation, or test cases. See also *common file*, *shared file*.

fix record. A status record that is associated with a track and is used to monitor the phases of change in each component that is affected by a defect or feature for a specific release.

function key. A key appearing at above or beside the normal character keys on a keyboard which can be programmed to perform particular functions in particular program contexts.

GUI. See *graphical user interface*.

graphical user interface (GUI). The OSF/Motif**-based CMVC graphical user interface program.

home directory. The directory users access when they log in.

host. Host node, host computer, or host system.

host list. A list associated with each CMVC user ID which indicates the client hosts that can access the CMVC server and act on behalf of the CMVC user. The list is used by the CMVC server to authenticate the identity of a CMVC client upon receipt of a CMVC command. Each entry consists of a login name, a CMVC user ID, and a host name.

implicit authority. The ability to perform an action against a CMVC object without being granted explicit authority. This authority is implicitly granted due to object ownership. Contrast with explicit authority and base authority.

inheritance. The passing of configuration management properties from parent component to child component. The configuration management properties that are inherited are access and notification. Inheritance in a component hierarchy is cumulative.

Internet protocol (IP). The protocol that provides the interface from the higher level host-to-host protocols to the local network protocols.

IP. Internet protocol

JCL. Job control language

job control language. On MVS, a command interpreter/programming language which 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.

level. A collection of tracks, which represent a set of changed files in a release.

level member. A track that has been added to a level.

lock. Prevents editing access to a file stored in the CMVC development environment so that only one user can make changes to a given file at one time.

login. Operating system user identification.

make. The make command assists you in maintaining a set of programs, usually pertaining to a particular software project. It does this by building up-to-date versions of programs.

makefile. This description file tells the **make** command how to build the target file, which files are involved, and what their relationships are to the other files in the procedure.

map. The process of reassigning the meaning of an object.

message context. The Broadcast Message Server uses a context field to pass file name information between the various SDE WorkBench/6000 or Hewlett-Packard SoftBench tools. The context is made up of the host, dir, and file.

message server. The facility that coordinates the SDE WorkBench/6000 or HP SoftBench tools. It receives messages from tools and routes them to other tools.

NetLS. See *Network License System*.

Network File System (NFS). A program that allows you to share files with other computers in one or more networks over a variety of machine types and operating systems.

Network Licensing System (NetLS). A program that controls the number of users who can simultaneously access CMVC or other products.

NFS. See *Network File System*.

notification list. A CMVC object allowing component owners to configure notification. A list of user ID-interest group pairs attached to a component, designating users and the corresponding notification interest they are being granted for all objects managed by this component or any of its descendants.

online program. A user provides input interactively to an online program and views its output on a display, panel or window.

Open Systems. A common type of operating system which is available on many different vendors′ computers, across which programs may be easily ported. The many different versions and variants of UNIX which are available on many brands of computers are considered open systems by most people. Typically, an open system supports *de facto* industry and *de jure* formal standard interfaces, subsystems, languages, and utilities. Many vendors offering non-UNIX operating systems that include these standard interfaces and utilities consider their operating systems to be open systems also.

originator. The user who opens a defect or feature and is responsible for verifying the outcome of the defect or feature on a Verification record. This responsibility can be reassigned.

OSF/Motif. A window manager X client application resold by many UNIX vendors with their X Windows Server product. This is a core part of the 2D Feature of AIXwindows Environment/6000. There is also an OSF/Motif Style Guide governing the “look and feel” characteristics of OSF/Motif compliant GUIs.

owner. The user who is responsible for a CMVC object in a CMVC family, either because they created the object or was assigned ownership of that object

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 a input/output screen or display map.

parent component. See *child component* and *component*.

path name. The name of the file under CMVC control. A path name can be a set of directory names and a base name or just a base name. It must be unique in the release that groups the files.

problem tracking. The process of tracking all reported defects through to resolution and proposed features through to implementation.

profile. A file containing customized settings for a system or user.

project. A project is a set of interfaces designed for a single application.

process. A combination of CMVC subprocesses, configured by the family administrator, that controls the general movement of CMVC objects (defects, features, tracks, and levels) from state to state in a component or release. See also *subprocess* and *state*.

query. A structure request for information from a database, for example, a search for all defects that are in the open state. See also *search*.

relative path name. The name of a directory or a file expressed as a sequence of directories followed by a file name, beginning with the current directory.

release. A CMVC object defined by a user to group all files that must be built, tested, and distributed as a single entity.

restricted authority. The restriction of a user′s ability to perform certain actions at a specific component.

root component. The initial component that is created when a CMVC family is configured. All components in a CMVC family are descendants of the root component. Only the root component has no parent component.

SCCS. See *Source code control system*

scope. A parameter in the TOOL statement for each SDE WorkBench or HP SoftBench tool. It defines the fields in the message context used by the Execution Manager to determine whether a particular tool can service a particular request.

search. The scanning of one or more data elements of a set in a database to find elements that have certain properties.

server. A workstation that performs a service for another workstation.

SEI. Software Engineering Institute

shared file. A file that is shared between two or more releases. See also *common file*.

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.

Software Engineering Institute. Carnegie-Mellon University ′s Software Engineering Institute is a research and development center funded by the United States federal government. Carnegie-Mellon University developed an assessment vehicle that was accepted by the U.S Department of Defense. This assessment allows to improve the software development processes will help achieve quality, productivity, and cycle-time reduction goals.

Source Code Control System (SCCS). The Source Code Control System (SCCS) is a complete system of commands that allows specified users to control and track changes made to an SCCS file. SCCS files allow several versions of the same file to exist simultaneously, which can be helpful when developing a project requiring many versions of large files. The SCCS commands support Multibyte Character Set (MBCS) characters.

state. Tracks, levels, features, and defects move through various states during their life cycles. The state of an object determines the actions that can be performed on it. See also *process* and *subprocess*.

subprocess. CMVC subprocesses govern the state changes for CMVC objects. The design, size, review (DSR) and verify subprocesses are configured for component processes. The track, approve, fix, level, and test subprocesses are configured for release processes. See also *process* and *state*.

superuser privilege. A user who is granted superuser privilege. Superuser privilege allows a user to perform any action available in the CMVC family.

system administrator. A user who is responsible for all system-related tasks involving the CMVC server, such as, installing, maintaining, and backing up the CMVC server and the relational database being used by the CMVC server.

TCP. Transmission control protocol.

Transmission control protocol. A communication protocol used in internet following the U.S. Department of Defense standards for internetworking protocol. See also *Internet protocol*.

tester. A user responsible for testing the resolution of a defect or the implementation of a feature for a specific level of a release and recording the results on a test record.

tool. In SDE terminology a tool is an encapsulated application.

tool context. The range of data for which a tool is registered to receive requests and perform actions.

track. A CMVC object created to monitor the progress of changes in a release to resolve a specific defect or implement a specific feature.

user. A person with an active user ID and access to one or more CMVC families.

User exit (UE). A user exit allows CMVC to call a user-defined program during the processing of CMVC transactions. User exits provide a means by which a user can specify additional actions that should be performed before completing or proceeding with a CMVC action.

UE. User exit

verification record. A status record which must be marked by the originator of a defect or a feature before the defect or feature can move to the closed state. This allows the originator to verify the resolution or implementation of the opened defect or feature.

version control. The storage of multiple versions of a single file along with information about each version.

view. An alternate and temporary representation of data from one or more tables.

working file. The currently checked-out version of a CMVC file.

X client. An application which makes calls to X Windows library subroutines to request that an X server program perform input/output at a graphical display.

X server. The X Windows software which manages the input/output resources of a graphical display (monitor, keyboard, pointing devices).

X station. A network-attached device which executes the X server software to control a display unit (monitor, keyboard, 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 an X station, a user must login to another computer on the network.

X Windows. A client-server graphical windowing product from MIT** which forms the basis of the 2D Feature of AIXwindows Environment/6000 along with OSF/Motif.

List of Abbreviations

4GL	Fourth Generation Language
ADT	Application Development Toolkit
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 Nationsl Standard Code for Information Interchange
BMS	Broadcast Message Server
CASE	Computer Aided Software Engineering
CLIST	Command List
CMVC	Configuration Management and Version Control
CUA	Common User Access
DB2	DATABASE 2
DNS	Domain Name Service
EBCDIC	Extended Binary Coded Decimal Interchange Code
FTP	File Transfer Protocol
GB	Gigabyte
GUI	Graphical User Interface
IBM	International Business Machines Corporation
IO	Input/output
IP	Internet Protocol
ISPF	Interactive System Panel Facility
ITSC	International Technical Support Center
ITSO	International Technical Support Organization
LAN	Local Area Network
LPP	Licensed Program Product
LPEX	Live Parsing Editor (PC and AIX version of LEXX)
MB	Megabyte
MVS	Multiple Virtual Storage
NCS	Network Computing Service
NFS	Network File System
NIS	Network Information System
NLS	National Language Support
OEM	Original equipment manufacturer
OSF	Open Software Foundation
OODB	Object Oriented Data Base
OS	Operating system
POWER	Performance Optimized With Enhanced RISC
PVCS	Program Version Control System
QA	Quality Assurance
RISC	Reduced Instruction Set Computer
SCCS	Software Change Control System
SCM	Software Configuration Management
SCRB	Software Change Review Board
SDE	Software Development Environment
SDRB	Software Design Review Board
SEE	Software Engineering Environment
SEI	Software Engineering Institute
SMP	Symmetrical multiprocessor
SNA	Systems Network Architecture
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

Special Characters

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

ITSO Technical Bulletin Evaluation RED000

Did You Say CMVC?

Publication No. GG24-4178-00

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.

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

Cut or Fold
Along Line

Printed in U.S.A.