

**CONSOLIDATED DOCUMENT MANAGEMENT SYSTEM
(CDOCS)
SYSTEM SUPPORT MANUAL**

Prepared for

**Nuclear Regulatory Commission
Contract NRC-02-93-005**

Prepared by

**Aaron R. DeWispelare
Robert L. Marshall
Joseph H. Cooper
David T. Lincoln
James L. Patty**

**Center for Nuclear Waste Regulatory Analyses
San Antonio, Texas**

July 1996

ABSTRACT

This manual is intended to support system level operations, maintenance, and code development of the Consolidated Document Management System (CDOCS). CDOCS was developed to address the spectrum of Nuclear Regulatory Commission (NRC) Division of Waste Management (DWM) document management needs. The primary focus of those needs is the technical and regulatory support for prelicensing and licensing activities associated with the High-Level Radioactive Waste Repository Program. The CDOCS system is a document management system that supports full-text search and retrieval of documents, and hyperlinking of images and documents.

CDOCS is a client server system in which functional support for the user interaction and interface is implemented on the client platform, and database functionality is implemented on the server platform. Client support in the CDOCs is implemented for four different hardware/software environments that are functionally equivalent but differ in some details of the presentation of the user interface. The client platforms and environments supported include personal computers (Windows and OS/2), SUN workstations (OS and Solaris) and Macintosh (System 7). The code for all platforms is consolidated in a single set of modules. The ease of use on each platform is achieved by use of the graphical user interface tool GALAXY. The server support is implemented on a UNIX-based SUN workstation that utilizes the facilities of the ORACLE database and the TOPIC full-text search and retrieval software. Communication between the client and server platforms is accomplished through Remote Procedure Call facilities. This isolates platform dependencies and permits the client and server implementations to be completely independent of each other.

CONTENTS

Section	Page
FIGURES	xi
TABLES	xiii
ACKNOWLEDGMENTS	xv
 1 INTRODUCTION	 1-1
1.1 OVERVIEW	1-1
1.1.1 Supporting Documentation	1-2
1.1.1.1 Documentation of the Former TDOCS System	1-2
1.1.1.2 Documentation of the Former RPD System	1-2
1.1.1.3 In Code Documentation	1-2
1.2 APPLICATIONS	1-3
1.2.1 Client	1-3
1.2.1.1 Graphical User Interface	1-3
1.2.1.2 Remote Procedure Call	1-3
1.2.1.3 Test	1-4
1.2.2 Server	1-4
1.2.2.1 Remote Procedure Call Code	1-4
1.2.2.2 Server Code	1-4
1.2.2.3 Kill Code	1-6
1.2.2.4 Make and Runtime—Make	1-6
1.2.2.5 Make and Runtime—Start Up	1-6
1.2.2.6 Make and Runtime—Versions	1-7
1.2.2.7 Make and Runtime—Initialization File	1-7
1.2.3 Batch	1-8
1.2.3.1 Batch Code	1-7
1.2.3.2 Make and Runtime—Make	1-8
1.2.3.3 Make and Runtime— Start UP	1-8
1.2.3.4 Make and Runtime— Versions	1-9
1.2.3.5 Make and Runtime—Initialization File	1-9
1.2.4 Report Writer	1-9
1.2.4.1 Word Perfect and Intellitag	1-9
1.2.4.2 Pieces, WPChar, and Writer	1-10
1.2.5 Help	1-10
1.3 TOOLS	1-10
1.3.1 ORACLE	1-10
1.3.1.1 Pro*C	1-11
1.3.1.2 Database Tables	1-11
1.3.1.3 SQL*Net	1-13
1.3.2 TOPIC	1-13
1.3.2.1 Directory Structure	1-13
1.3.2.2 Tools and Scripts	1-14

CONTENTS (Cont'd)

Section	Page
2 CDOCS FUNCTIONAL DESCRIPTION AND LOGIC FLOW	2-1
2.1 INTRODUCTION	2-1
2.1.1 Typographic Conventions	2-1
2.2 CLIENT SYSTEM TECHNICAL DOCUMENTATION	2-1
2.2.1 Overall System Flow	2-2
2.2.1.1 Client Server Interaction	2-2
2.2.1.2 Client Server Interaction Using Callbacks	2-4
2.2.1.3 Client Server Interaction Using the Main Event Loop	2-4
2.2.2 Multiplatform Implementation	2-7
2.2.3 Client Program Initiation	2-7
2.2.3.1 Main Program Initiation	2-8
2.2.3.2 Logon Processing	2-8
2.2.4 CDOCS CALLBACKS	2-10
2.2.5 Main Event Loop	2-15
2.2.6 Consolidated Document System Main Menu	2-21
2.2.7 CDOCS Main Menu—Operations Entry	2-22
2.2.7.1 Operations—Copying Regulatory Program and Open Item Records	2-22
2.2.7.2 Operations—Editing Regulatory Program and Open Item Records with WordPerfect	2-26
2.2.7.3 Operations Pull-down Menu— Change Password	2-27
2.2.7.4 Operations Pull-down Menu— Scan and OCR	2-28
2.2.7.5 Operations Pull-down Menu—Preferences	2-29
2.2.7.6 Operations—Exiting from the Consolidated Document System	2-29
2.2.8 Main Menu—Search Entry	2-29
2.2.8.1 Accessing the Search Facilities	2-30
2.2.8.2 CDOCS Search Screen	2-30
2.2.9 Main Menu—Custodian Entry	2-31
2.2.9.1 Custodian Pull-Down Menu	2-32
2.2.9.2 Submitting New CDOCS Records	2-32
2.2.9.3 Updating Records from the Consolidated Document Management System	2-38
2.2.9.4 The CDOCS Update Entry Screen	2-39
2.2.9.5 Deleting Records from the Consolidated Document System	2-42
2.2.9.6 The CDOCS Delete Screen	2-45
2.2.10 Custodian Functions For Regulatory Records	2-46
2.2.10.1 Defining New Regulatory Records	2-47
2.2.10.2 Enter New and Updated Records	2-52
2.2.10.3 Enter Open Item Tracking System Records	2-58

CONTENTS (Cont'd)

Section		Page
2.2.10.4	Format Check for New and Updated Regulatory Records	2-60
2.2.10.5	Format Check for Open Item Tracking System Records	2-63
2.2.10.6	Retire Regulatory Program Records	2-63
2.2.11	Custodian Functions For Library Control	2-65
2.2.11.1	Checking Out a Consolidated Document System Record	2-65
2.2.11.2	Checking In a CDOCS Record	2-69
2.2.12	Main Menu—Report Entry	2-71
2.2.12.1	Report Pull-down Menu	2-71
2.2.12.2	New Acquisitions—Functionality	2-74
2.2.12.3	Database Content (CDS, CDM, Open Items) Cascading Menu	2-77
2.2.12.4	CDOCS Report Display	2-79
2.2.12.5	Selecting, Displaying, and Printing Database Reports	2-79
2.2.13	Main Menu—System Entry	2-81
2.2.13.1	System Menu Processing	2-81
2.2.14	Main Menu—Help Entry	2-83
2.2.14.1	Help Screen Functionality	2-83
2.3	SERVER SYSTEM TECHNICAL DOCUMENTATION	2-87
2.3.1	Main Server Program Initialization	2-87
2.3.2	Initialization of the Synchronization Process	2-88
2.3.3	Synchronization Processing	2-88
2.3.3.1	Synchronization of TDOCS Documents	2-88
2.3.3.2	Synchronization of RPD Documents	2-90
2.3.4	Switch Function Call/Data Return Mechanism	2-90
2.3.5	Main Service Switch Function	2-90
2.4	BATCH SYSTEM TECHNICAL DOCUMENTATION	2-109
2.4.1	The Batch Process	2-109
2.4.1.1	Batch Command Line Parameters	2-109
2.4.2	Main Batch Program	2-110
2.4.2.1	Retrieve the Command Line Arguments	2-110
2.4.2.2	Initialize the Environment	2-111
2.4.2.3	Log In	2-111
2.4.2.4	Get the Report Date	2-112
2.4.2.5	Set the Working Directory	2-112
2.4.3	TDOCS Batch Execution	2-112
2.4.3.1	TDOCS Batch Execution—Open Batch Sets	2-114
2.4.3.2	TDOCS Batch Execution—Fetch Batch Sets	2-114
2.4.3.3	TDOCS Batch Execution—Update Documents	2-114
2.4.3.4	TDOCS Batch Execution—Submit Documents	2-116
2.4.3.5	TDOCS Batch Execution—Delete Documents	2-118

CONTENTS (Cont'd)

Section	Page
2.4.3.6	TDOCS Batch Execution—Batch Reports 2-119
2.4.4	RPD Batch Execution 2-120
2.4.4.1	RPD Batch Execution—Define 2-120
2.4.4.2	RPD Batch Execution—Checkin 2-121
2.4.4.3	RPD Batch Execution—Retire 2-123
2.4.5	Logout and Exit 2-123
3	COMPILATION AND EXECUTION INSTRUCTIONS 3-1
3.1	COMPILING CDOCS COMPONENTS 3-1
3.1.1	CDOCS Server 3-1
3.1.2	CDOCS Batch 3-4
3.1.3	CDOCS Clients 3-5
3.1.3.1	General 3-5
3.1.3.2	Solaris 3-5
3.1.3.3	SUN OS 3-6
3.1.3.4	Windows 3-6
3.1.3.5	OS/2 3-8
3.1.3.6	Macintosh 3-10
3.2	EXECUTION 3-10
4	INDEXED README FILES 4-1
4.1	READ.ME FILES INDEX 4-1
4.2	READ.ME FILE LISTINGS 4-2
4.2.1	Read.Me on How to Start CDOCS Batch 4-2
4.2.2	Read.Me on How to Start CDOCS Help System 4-3
4.2.3	Read.Me on How to Run SQLNet 4-3
4.2.4	Read.Me on How to Start CDOCS Server 4-4
4.2.5	Read.Me for Making and Running the Batch Server 4-4
4.2.6	Read.Me for Client Code for All Platforms (SUN OS, Solaris, Windows, OS/2, Macintosh) 4-6
4.2.7	Read.Me for Help System 4-7
4.2.8	Read.Me for Macintosh-Specific Applications for CDOCS 4-10
4.2.9	Read.Me for Making and Running the Server 4-10
4.2.10	Read.Me for Windows-Specific Applications for CDOCS 4-15
5	INSTALLATION 5-1
5.1	OVERVIEW 5-1
5.2	INSTALLATION ON A WINDOWS WORKSTATION 5-3
5.2.1	Ensure That the Workstation Meets the Minimum Hardware and Software Requirements for a Windows Workstation 5-3
5.2.2	Install TOPIC 4.0 5-3
5.2.3	Install the CDOCS 5-5

CONTENTS (Cont'd)

Section		Page
	5.2.3.1 To Install CDOCS on a Windows Workstation:	5-5
	5.2.3.2 To Install the CDOCS Files Required for a Windows Workstation on a Connected File Server	5-5
	5.2.4 Setup of Windows Workstation Environment	5-6
	5.2.5 Windows Workstation Installation Summary	5-7
5.3	INSTALLATION ON A SUN WORKSTATION	5-8
	5.3.1 Ensure That the Workstation Meets the Minimum Hardware and Software Requirements for a SUN Workstation	5-8
	5.3.2 Install TOPIC 4.0	5-9
	5.3.3 Install the CDOCS	5-9
	5.3.4 Setup of the SUN Workstation Environment	5-10
	5.3.5 SUN Workstation Installation Summary	5-11
5.4	INSTALLATION ON AN OS/2 WORKSTATION	5-11
	5.4.1 Ensure That the Workstation Meets the Minimum Hardware and Software Requirements for a Windows Workstation	5-12
	5.4.2 Install TOPIC 4.0	5-12
	5.4.3 Install the CDOCS	5-13
	5.4.3.1 To Install CDOCS on an OS/2 Workstation:	5-13
	5.4.3.2 To Install the CDOCS Files Required for an OS/2 Workstation to Access the CDOCS on a Connected File Server	5-14
	5.4.4 Setup of OS/2 Workstation Environment	5-15
	5.4.5 OS/2 Workstation Installation Summary	5-17
5.5	SELECTING USER PREFERENCES	5-18
6	STRUCTURE OF DATABASES	6-1
6.1	CDOCS ORACLE DATABASE	6-1
	6.1.1 Databases and the Database Administrator	6-1
	6.1.2 Instances	6-1
	6.1.3 Tablespaces	6-1
	6.1.4 Database Objects	6-1
6.2	TDOCS Objects	6-1
6.3	RPDOCS Objects	6-8
6.4	RPD OBJECTS	6-11
6.5	CONSOLIDATED DOCUMENTS SYSTEM TOPIC DATABASE DATA REPOSITORIES	6-14
	6.5.1 Directory Structures in the UNIX File System	6-14
	6.5.1.1 Archive Subdirectories	6-16
	6.5.1.2 Images Subdirectories	6-16
	6.5.1.3 Data Subdirectories	6-17
	6.5.2 Directory Structures in the TOPIC Specific File System	6-17
6.6	REPORT WRITER FILE STRUCTURE	6-19

CONTENTS (Cont'd)

Section	Page
7 INDEXED INTERNAL MODULE COMMENTS—CLIENT	7-1
7.1 CLIENT MODULE INDEX	7-1
7.2 CLIENT LISTING	7-6
7.2.1 Client C	7-6
7.2.2 Client H	7-113
8 INDEXED INTERNAL MODULE COMMENTS—SERVER/BATCH	8-1
8.1 SERVER/BATCH MODULE INDEX	8-1
8.2 SERVER/BATCH COMMENT LISTINGS	8-4
8.2.1 Server C	8-4
8.2.2 Batch C	8-86
8.2.3 Server H	8-122
8.2.4 Batch H	8-145

FIGURES

Figure		Page
2-1	Client-server flow	2-3
2-2	Callback functionality	2-5
2-3	Main event loop	2-6
2-4	DWM logon screen	2-10
2-5	CDOCS—CDOCS main menu	2-23
2-6	CDOCS—Operations pull-down menu	2-24
2-7	Copy a regulatory program record	2-24
2-8	File chooser input screen	2-25
2-9	Edit a regulatory program record using WordPerfect screen	2-27
2-10	CDOCS—Search selection screen	2-31
2-11	CDOCS—NRC Custodian pull-down menu	2-32
2-12	CDOCS—CNWRA Custodian pull-down menu	2-33
2-13	CDOCS Submit—NRC Technical Document entry screen	2-35
2-14	CDOCS Submit—extended entry-field screen	2-36
2-15	CDOCS Submit—directory where files reside	2-37
2-16	CDOCS Update screen	2-40
2-17	CDOCS update entry screen	2-42
2-18	CDOCS Update confirmation screen	2-43
2-19	CDOCS delete screen	2-44
2-20	CDOCS Delete conformation screen	2-46
2-21	CDOCS Delete response screen	2-47
2-22	CDOCS process a regulatory record cascading menu	2-48
2-23	Define a regulatory program record input screen	2-49
2-24	Define a regulatory program record document type selection	2-50
2-25	Define a regulatory program record	2-51
2-26	Define an OITS record	2-53
2-27	Enter a regulatory program record input screen	2-54
2-28	Enter a regulatory program record document type selection	2-55
2-29	Enter an OITS record	2-58
2-30	Format check a regulatory program record screen	2-61
2-31	Retire a regulatory program record screen	2-64
2-32	CDOCS checkout entry screen	2-67
2-33	CDOCS checkin entry screen	2-70
2-34	Report pull-down menu	2-72
2-35	Circulation reports selection screen	2-74
2-36	New Acquisitions Report selection screen	2-75
2-37	New Acquisitions Report selection example	2-75
2-38	New acquisitions report screen	2-76
2-39	Database content cascading menu	2-78
2-40	Regulatory records status report	2-80
2-41	CDOCS system pull-down menu	2-83
2-42	CDOCS User ID maintenance screen	2-84
2-43	CDOCS help pull-down menu	2-85

FIGURES (cont'd)

Figure		Page
2-44	CDOCS help index	2-86
2-45	CDOCS help screen	2-87
2-46	Synchronization	2-89
2-47	Batch	2-113
5-1	Sample installation layout	5-2
5-2	TOPIC Custom Installation Options screen	5-4
5-3	CDOCS Preferences screen	5-20
6-1	Directory structures in the UNIX file system	6-15

TABLES

Table	Page
2-1	CDOCS Conditional Compilation Defines 2-7
2-2	CDOCS Callbacks 2-11
2-3	CDOCS Event Definitions 2-16
2-4	Effect of sharable flag during record update 2-41
2-5	Service requests and functions 2-91
2-6	Content and Status Report Requests 2-99
2-7	Initialization of the data set flag 2-111
3-1	CDOCS executables 3-1
3-2	CDOCS server build summary 3-3
3-3	CDOCS batch build summary 3-4
3-4	Solaris client make information 3-6
3-5	SUN OS client make information 3-7
3-6	Windows 3.1 client make information 3-7
3-7	rpc16.dll make information 3-8
3-8	OS/2 development packages 3-8
3-9	OS/2 client make information 3-9
3-10	Macintosh client make information 3-10
4-1	Read.me index 4-1
5-1	Windows workstation installation summary list 5-7
5-2	SUN workstation installation summary list 5-11
5-3	OS/2 workstation installation summary list 5-17
6-1	TDOCS 6-2
6-2	TDOCS_AUTHORS 6-3
6-3	TDOCS_ADDRESSEES 6-3
6-4	TDOCS_DOCUMENT_STATUS 6-3
6-5	TDOCS_ASSIGNED_CODES 6-4
6-6	TDOCS_ASSIGNED_NUMBERS 6-4
6-7	TDOCS_REPORTS 6-4
6-8	TDOCS_DOCUMENT_SETS 6-5
6-9	TDOCS_CIRCULATION 6-5
6-10	TDOCS_HYDRO 6-6
6-11	TDOCS_NIST_ABSTRACTS 6-6
6-12	TDOCS_NIST 6-7
6-13	TDOCS_NIST_REVIEWS 6-8
6-14	TDOCS_NIST_KEYWORDS 6-8
6-15	PEOPLE 6-9
6-16	RPDOCS_USERS 6-9
6-17	RPDOCS_NAMES 6-9
6-18	RPDOCS_HELP 6-10

TABLES (cont'd)

Table		Page
6-19	RPDOCS_HELP_INDEX	6-10
6-20	RPDOCS_RIVILEGES	6-10
6-21	RPD_DOCUMENT_SETS	6-11
6-22	RPD_REPORTS	6-11
6-23	RPD_PIECES	6-12
6-24	RPD	6-13
7-1	Client c index	7-1
8-1	Server c/pc index	8-1
8-2	Batch c/pc index	8-3

ACKNOWLEDGMENTS

This manual was prepared to document work performed by the Center for Nuclear Waste Regulatory Analyses (CNWRA) for the Nuclear Regulatory Commission (NRC) under Contract No. NRC-02-93-005. The activities reported here were performed on behalf of the NRC Office of Nuclear Material Safety and Safeguards (NMSS), Division of Waste Management (DWM). The report is an independent product of the CNWRA and does not necessarily reflect the views or regulatory position of the NRC.

The Consolidated Document Database System (CDOCS) currently runs on the following hardware/software environments:

- SUN (IPX or later)
 - OS Version 4.1.3
 - Solaris Version 2.3
- Personal Computer (80486 or later)
 - Windows Version 3.1
 - OS/2 Version 2.1 or later
- Macintosh
 - System 7 or later

CDOCS currently uses the following commercial products:

- ORACLE Version 7.1.6.4
- TOPIC Versions 3.1.5 or 4.0
- GALAXY Versions 2.0 or 2.5

The source code is under configuration control at the CNWRA using Source Code Control System (SCCS) through SPARCworks/Teamwear. The computer code for CDOCS is currently being placed under Control of Technical Operating Procedure-018, "Development and Control of Scientific and Engineering Software".

1 INTRODUCTION

1.1 OVERVIEW

The Consolidated Document Management System (CDOCS) was developed to address the spectrum of Nuclear Regulatory Commission (NRC) Division of Waste Management (DWM) document management needs. The primary focus of those needs is the technical and regulatory support for preclicensing and licensing activities associated with the high-level radioactive waste repository program. The CDOCS system¹ is a document management system that supports full-text search and retrieval of documents, and hyperlinking of images and documents.

This manual is intended to support system level operations, maintenance, and further code development. To use this manual effectively, the system level personnel have to be experienced UNIX users who are proficient with the "C" language, and the software products TOPIC, ORACLE, and GALAXY. All directory paths are stated using the system set-up at the Center for Nuclear Waste Regulatory Analyses (CNWRA). The NRC will likely have system architectural differences which will require appropriate changes in the prefix portions of the directory paths.

Chapter 1 of this manual overviews the basic structure of the client, server, batch processing server (here after called "batch" which is used to process documents into the TOPIC database), and communications protocol. Chapter 2 provides a more detailed functional flow and description of each of the client, server and batch. Chapter 3 contains compiling instructions for the CDOCS code and Chapter 4 provides an indexed listing of the system read.me files. Chapter 5 delineates installation of both the server/batch and the client (on all supported platforms). Chapter 6 describes the structure of both the TOPIC and ORACLE databases. Chapter 7 contains an indexed set of internal code documentation for the client and Chapter 8 contains the indexed internal code comments for the sever and batch. Appendix A documents the input side of the report writer including document technical definitions (DTD) and WordPerfect Macros, and Appendix B provides a similar description for the output of the report writer for all standard reports which are produced automatically by the system. Appendix C lists the makefiles for the client, server, and batch compiles

The rest of this chapter briefly describes the system client, server and batch code portions and functionality. It provides references to documentation, overview of applications, explanation of functionality, insights into tools, and a preview of directory structures. Its intended audience is a software developer who knows C, ORACLE, TOPIC, remote procedure call (RPC), and UNIX (cshell, bourne shell, and awk).

¹ Initially, two separate systems were developed, the Technical Document Reference Database System (TDOCS) for managing technical documents and the Regulatory Program Database System (RPD) for the management of regulatory documents. RPD also included the former Open Item Tracking System (OITS). Since TDOCS and RPD used the same technology, these were merged into one combined application which resulted in the development of CDOCS.

1.1.1 Supporting Documentation

Requirements definition, system design, users guides, etc. are included in previous published reports which are available through the CNWRA library. The code itself is internally documented as well. The latest user guide for the CDOCS system is entitled "Draft User Guide, Consolidated Document System (CDOCS), Version 1.0."

1.1.1.1 Documentation of the Former TDOCS System

The following documents are available for the original TDOCS system:

- Technical Reference Document Database System (TDOCS) Requirements Definition
- Technical Reference Document Database System (TDOCS) Design Plan
- Proposed Technical Reference Document Database System (TDOCS) Design
- User's Guide for Prototype of the Technical Reference Document Database System (TDOCS)

These documents provide insight into the requirements, development plans and capability of TDOCS.

1.1.1.2 Documentation of the Former RPD System

The following documents are available for the original RPD system:

- Review and Analysis of the PASS/PADB System for Systematic Regulatory Analysis.
- Development Plan for the PASS/PADB System Design Version 3.0
- User's Guide for Regulatory Program Database (RPD) Version 1.0
- User's Guide for Regulatory Program Database (RPD) Version 2.0

The review and analysis provides insight into the reasons for transitioning from the Program Architecture Support System (PASS)/Program Architecture Database (PADB) system to the current one along with justification of tools. The development plan provides insight into early (1993) system design. The user's guide provides insight into RPD from a user's point of view.

1.1.1.3 In Code Documentation

The code is internally documented. Each module starts with a brief description of its purpose. Each function has a header that includes function name, purpose, return values, calling and called functions. High level functions provide more algorithmic detail. In line comments are used to make the understanding of the code easier. Chapters 7 and 8 provide an indexed listing of the internal code comments.

1.2 APPLICATIONS

The CDOCS applications are client, server, batch, and others. The client and server communicate using RPC on a transaction basis, the client requesting and the server providing services including all ORACLE and TOPIC access. Batch consists of long running functions removed from the server. Other applications include help and report writer utilities.

1.2.1 Client

The client has a graphical user interface (GUI) and an RPC interface. There is also test code with American Standard Code for Information Interchange (ASCII) and RPC interfaces. A client connects once and makes RPC requests to the server which returns results. Requests and results send or retrieve user information; help topics and text; document lists, headers, and files; and report lists and text.

1.2.1.1 Graphical User Interface

The interface has a GUI written in C and Visix Galaxy. Galaxy provides GUI support for multiple platforms, currently Solaris, SUN OS, Windows, Macintosh, and OS/2. The server is completely independent of this interface.

1.2.1.2 Remote Procedure Call

The client communicates with the server through SUN RPC. On the client side the RPC interface is coded in three layers. At the highest layer are RPC-prefixed calls like RPCCConnect, RPDGetUserInfo, RPCGetTDOCSRecord, etc. At the middle layer are two functions, _RPCCConnect and _RPDRequest which format the packets to be sent and assign returned results. The middle layer is #ifdef'd for each platform type in _RPCCConnect and _RPCRequest. RPC conversion (xdr) functions at the lowest layer encode and send, and receive and decode data. The server is dependent on this interface for receiving data. The following modules code this interface (under rpdocs/development/):

* rpc/da.h	-	RPC definitions and prototypes
* rpc/dadefs.h	-	request ids and error ids
* rpc/da_xdr.c	-	lowest layer data transfer functions
* rpc/da_clnt.c	-	lowest layer connect and request functions
* rpc/clntrpc.h	-	middle layer definitions and prototypes
* rpc/clntrpc.c	-	middle layer connect and request functions
* rpc/client.h	-	highest layer definitions and prototypes
* rpc/client.c	-	highest layer connect and request functions

The Windows port required 32- to 16-bit thunking. The 32-bit GUI side uses special Watcom functions to do this when passing data to a 16-bit dynamic load library (DLL). The DLL is compiled with Microsoft C 7.0, and uses, in addition to the above listing, the following modules:

* rpc/makefile.win	-	make file
* rpc/read.me.windows	-	how to make the DLL
* rpc/rpcl6.c	-	DLL code

1.2.1.3 Test

Small ASCII-interfaced test applications have been coded. Each of these tests one or two services; the whole range of services is covered. But they are fairly hard coded and easy to modify. The test code is located in:

* interface	-	base test code, makefile
* interface/TEST_RPDOCS	-	shared CDOCS tests
* interface/TEST_TDOCS	-	TDOCS tests
* interface/TEST_SYNC	-	synchronization tests
* interface/TEST_RPD	-	RPD tests

Note that the makefile needs to be edited to set input and output for each test.

1.2.2 Server

The server, coded in C, multitasks and has RPC, ORACLE, and TOPIC interfaces. On start up, it reads a versioned initialization file, registers with the RPC port mapper and tests access to ORACLE and TOPIC. On receiving a request from a client, the server processes this request and returns results to the client; if it is multitasking, it forks a child to handle each request and the child returns results before exiting. Access to ORACLE is through Pro*C, and access to TOPIC is through system calls.

The code for the server can be found in three places: rpc, server, and kill.

1.2.2.1 Remote Procedure Call Code

The following modules code the RPC interface:

* rpc/da.h	-	RPC definitions and prototypes
* rpc/dadefs.h	-	request ids and error ids
* rpc/da_xdr.c	-	data transfer functions
* rpc/da_svc.c	-	main and initialization functions

1.2.2.2 Server Code

The following modules code the server:

\$ server/read.me.server	-	comments on server operation
\$ server/makefile	-	makefile
\$ server/da_svc_*.ini	-	initialization file
\$ server/da_svc_proc.c	-	service procedure
\$ server/dat.c	-	packet decoding, results encoding
\$ server/dat.h	-	packet decoding, results encoding
\$ server/db.h	-	general database header
\$ server/max.h	-	database column lengths
\$ server/db.pc	-	database VARCHARS
\$ server/doc.h	-	file read and write header

\$ server/doc.c	-	file read and write functions
\$ server/env.h	-	header for ini file reader
\$ server/env.c	-	ini file reader
\$ server/log.h	-	transaction logging header
\$ server/log.c	-	transaction logging functions
\$ server/mem.h	-	memory header
\$ server/mem.c	-	memory functions
\$ server/sllist.h	-	header for single link list
\$ server/sllist.c	-	"generic" single link list
\$ server/usr.pc	-	rpdocs user functions
\$ server/rpt.h	-	header for "generic" reports
\$ server/rpt.c	-	"generic" reports
\$ server/rpd.h	-	rpdocs header
\$ server/rpd.pc	-	rpdocs general database
\$ server/rpdchk.pc	-	rpdocs checkin
\$ server/rpddef.pc	-	rpdocs definition
\$ server/rpddtd.h	-	header for rpdocs SQL*D
\$ server/rpddtd.pc	-	rpdocs SQL*DTD
\$ server/rpddoc.pc	-	rpdocs documents
\$ server/rpdret.pc	-	rpdocs retire
\$ server/rpdrpt.h	-	header for rpdocs reports
\$ server/rpdrpt.pc	-	rpdocs reports
\$ server/rpdrw.h	-	header for rpdocs report writer
\$ server/rpdrw.pc	-	rpdocs report writer
\$ server/rpdtop.pc	-	rpdocs topic
\$ server/tdocs.h	-	tdocs header
\$ server/tdocschk.pc	-	tdocs checkin/checkout
\$ server/tdocsdat.pc	-	tdocs data
\$ server/tdocsdel.pc	-	tdocs deletion
\$ server/tdocsdoc.h	-	header for tdocs files
\$ server/tdocsdoc.pc	-	tdocs files
\$ server/tdocshyd.h	-	header for tdocs hydrology
\$ server/tdocshyd.pc	-	tdocs hydrology
\$ server/tdocslbl.h	-	header for tdocs labels
\$ server/tdocslbl.pc	-	tdocs labels
\$ server/tdocsruns.pc	-	tdocs names
\$ server/tdocsnst.h	-	header for tdocs NIST
\$ server/tdocsnst.pc	-	tdocs NIST
\$ server/tdocsrec.pc	-	tdocs document records
\$ server/tdocsrpt.h	-	header for tdocs reports
\$ server/tdocsrpt.pc	-	tdocs reports
\$ server/tdocssub.pc	-	tdocs submission
\$ server/tdocsin.pc	-	tdocs sync in
\$ server/tdocsout.pc	-	tdocs sync out
\$ server/tdocsupd.pc	-	tdocs update

Note that the former TDOCS-specific modules are prefixed TDOCS, former RPD-specific modules are prefixed RPD; otherwise modules are shared. That convention, TDOCS and RPD prefixes, is followed through in function names. Other conventions: function names use mixed case, generally upper for the beginning of each word; variable names use mixed case; defines are all upper case; prototypes are defined in header files unless they are static.

Two scripts are kept with the server, restartda, to kill the current and restart another server process; and whichda, to list all running servers.

1.2.2.3 Kill Code

Kill is a simple client that connects to the server and requests that it exit. It is called by the restartda script. It is coded in these modules:

```
* kill/makefile    -   make file
* kill/kill_da.h   -   header file
* kill/kill_da.c   -   application code
```

1.2.2.4 Make and Runtime—Make

In order to compile the server executable, change to the server directory and run the “make” utility. For debugging run “make debug;” and for a release version run “make release”—both makes force a complete remake. In order to install the server in its proper place, run “make install.”

1.2.2.5 Make and Runtime—Start Up

```
This is used to run a version of the da_server, su to topic40
su - topic40 # the dash is required
sh bourne - shell required
```

and run

```
da-server <V> <U>/<P>@<D>
```

where <V> is the version number (e.g., 205, 206, etc.), <U> and <P> are a dba's user id and password, < D > is the database instance (e.g., prod1).

If you do not run as topic40 you will get file permission problems. Currently, only the database name prod1 is defined. All transaction requests, inputs, and results are logged in /db3/run*/sam*/server/svc_log at the CNWRA in files identified by server version and start up date.

/etc/rc.local lines should read:

...

```
su oracle -c /usr/oraclefbn/dbstart;\
su - topic40 -c '/db3/run*/sam*/server/da_server < V > < U > / < P > @ < D >';\
```

...

with an su to topic40 for each < V > that needs to run.

1.2.2.6 Make and Runtime—Versions

The server has versions of 205, 206, and 207 (207 is the production server connected to the production database at the NRC, 206 is the production server connected to the production database at the CNWRA, and 205 is a test server connected to a test database). The server version is identified by the first command line argument passed to the server. This number is combined with 0x02000000 to generate an RPC program number. The client preference files must be set to match this version number.

1.2.2.7 Make and Runtime—Initialization File

The server and batch read an initialization file on start up. This file is named

* server/da_svc_ < V > . ini

where < V > is the version number. The initialization file specifies the following information:

* run time

— task	=	whether or not to multitask
— sync	=	whether and how to synchronize
— log	=	where to write log files
— adm	=	who to send fail mail to
— bat	=	da_batch location
— sgml	=	location of Standard Generalized Markup Language (SGML) DTDs, macros, etc.

* local system

— proddata	=	oracle production instance name
— proddocs	=	topic production directory
— testdata	=	oracle test instance name
— testdocs	=	topic test directory

* remote system

— proddata	=	oracle production instance name
— proddocs	=	topic production directory
— testdata	=	oracle test instance name
— testdocs	=	topic test directory

Note that run time information must be filled in; local and remote values may be left empty. When debugging turn multitasking and synchronization off. Both multitasking and synchronization involve forks the debugger cannot accommodate.

1.2.3 Batch

The batch code is similar to the server code, except RPC has been stripped out and error handling is different.

1.2.3.1 Batch Code

The following modules have code related to batch operations:

* batch/read.me.batch	-	comments on batch operations
* batch/makefile	-	make file for batch
* batch/batch.c	-	main module
* batch/batch.h	-	tdocs/rpd shared header
* server/sllist.c	-	single linked list
* server/sllist.h	-	single linked list
* batch/batchrpt.c	-	batch report
* batch/batchrpt.h	-	header for batch report
* batch/batchdb.pc	-	database VARCHARS
* batch/batchdb.h	-	database access
* server/max.h	-	database column max's
* server/env.h	-	header for reading ini file
* server/env.c	-	reads ini file
* batch/rpd.h	-	common rpd header
* batch/rpdbat.pc	-	main rpd batch
* batch/rpdchk.pc	-	rpd checkin
* batch/rpddef.pc	-	rpd define
* batch/rpdret.pc	-	rpd retire
* batch/rpdwr.pc	-	rpd report writer
* batch/pieces.h	-	rpd report writer
* batch/writer.h	-	rpd report writer
* batch/tdocs.h	-	common tdocs header
* batch/tdocsbat.pc	-	main tdocs batch
* batch/tdocsdoc.pc	-	tdocs document handling
* batch/tdocsrec.pc	-	tdocs record handling
* batch/tdocstop.pc	-	tdocs topic handling
* batch/tdocsrpt.pc	-	tdocs report handling
* batch/tdocsrpt.h	-	tdocs report handling

1.2.3.2 Make and Runtime—Make

This is used to make the server run “make” in the batch directory. For debugging run “make debug” is run. And for a release version run “make release” is run. Both “makes” force a complete remake. In order to install the batch in its proper place, run “make install.”

1.2.3.3 Make and Runtime— Start UP

In order to run batch, after logging in to topic40, su to topic40

su - topic40 (the dash is required)

sh (bourne shell required)

and run

```
da_batch <V> <U>/<P>@<D> <S>
```

where < V > is the version number, < U > and < P > are a dba's user id and password, < D > is instance, and < S > is the data to batch.

If you don't run as topic40 you will get file permission problems.

Batch needs to find an initialization file, da_Svc_< V > . ini, in the server start up directory. This file describes the locations of TOPIC and SGML (used by the report writer function of old RPD) directories.

1.2.3.4 Make and Runtime— Versions

Even though batch is started as a version, there are no real batch versions per se, since it must read the proper initialization file.

1.2.3.5 Make and Runtime—Initialization File

Batch reads the same initialization file as the server (see Section 3.2.4.4).

1.2.4 Report Writer

The report writer is a complex of WordPerfect (now Novell) macros, DTDs, rules, etc., run through WordPerfect and Intellitag; along with pieces, wpchar, and writer utilities. CDOCS (RPD) documents are fed through a WordPerfect styler macro, Intellitag, and pieces; CDOCS (RPD) output is fed through writer, wpchar, and a back end macro.

1.2.4.1 WordPerfect and Intellitag

WordPerfect and Intellitag are vintage 5.1 SUN OS applications that must be run in background in Solaris binary compatibility mode. Everything must be set up perfectly for these to run in the background. Thus the server (and batch) must be started in bourne shell in the background and the system calls to launch them must also invoke bourne shell and be in the background also. Both can run indefinitely even if something goes wrong, e.g., a failed search, and the application waits for a user response to a warning prompt. If the system changes over time, this process will have to be tuned periodically. All WordPerfect and Intellitag files are located as follows:

* /db3/SGML/dtd	=	SGML DTDS
* /db3/SGML/macros	=	WordPerfect macros
* /db3/SGML/rules	=	SGML rules
* /db3/SGML/vacant	=	defined templates

1.2.4.2 Pieces, WPChar, and Writer

Database access code for the report writer can be found in:

* writer/database/makefile	=	test make file
* writer/database/database.h	=	database defines
* writer/database/database.pc	=	database access module
* writer/database/piece.pc	=	piece data access module
* writer/database/query.pc	=	query data access module
* writer/database/main.c	=	test module

1.2.5 Help

The GUI help system was recently moved to the database. To facilitate help input and maintenance, two utilities were written, one command-line based loader and dumper, the other a GUI based maintenance tool. The loader/dumper modules are in:

* help/ASCII/makefile	=	makefile
* help/ASCII/read.me.help	=	explains help
* help/ASCII/db.h	=	database access defines
* help/ASCII/max.h	=	host variable lengths
* help/ASCII/db.pc	=	database access functions
* help/ASCII/rec.h	=	C help record
* help/ASCII/io.h	=	file system defines
* help/ASCII/io.c	=	file system functions
* help/ASCII/da_help.c	=	main module

The GUI maintainer code is partially in

* help/db.h	=	database access defines
* help/max.h	=	host variable lengths
* help/db.pc	=	database access functions

1.3 TOOLS

1.3.1 ORACLE

ORACLE is the relational database management system. For COT software, it seems to lack robustness in operations, being intolerant to even minor changes in the operating environment. Once workarounds have been figured out, it runs smoothly.

1.3.1.1 Pro*C

ORACLE Pro*C is used to precompile embedded SQL into C code. Look at the server's make file to see how to set up rules and dependencies. The following configuration file has been set up for default precompilation:

```
$ORACLE-HOME/proc/pmscfg.h
```

The convention is to name modules with embedded SQL *.pc; Pro*C converts them into *.c. Other conventions followed are:

- * Pro*C host and indicator variables are all declared in db.pc and external as needed elsewhere
- * host variables are all declared as varchars with defined maximum widths (H_ *_MAX in max.h)
- * host variables are prefixed h_-, indicators i_-

With Pro*C version 2.1.6 the only problem encountered so far has been that it cannot understand the following syntax:

```
typedef struct tX {  
int a;  
char b;  
}  
tX, /* structure */  
*pX, /* structure pointer */  
**hX; /* structure pointer pointer */
```

It stumbles on the commas (,) so typedef as

```
typedef struct tX {  
int a;  
char b;  
}  
tX; /* structure */  
typedef tX *pX; /* structure pointer */  
typedef tX **hX; /* structure pointer pointer */
```

Later versions of Pro*C are supposed to fix this bug.

1.3.1.2 Database Tables

The following tables are accessed:

- | | | |
|---------------------|---|------------------------|
| * RPDOCS_USERS | - | userids and privileges |
| * RPDOCS_NAMES | - | user names |
| * RPDOCS_PRIVILEGES | - | privilege descriptions |
| * RPDOCS_HELP | - | help headers and text |
| * RPDOCS_HELP_INDEX | - | help index |

* PEOPLE	-	CNWRA user names
* TDOCS	-	tdocs headers
* TDOCS_ADDRESSEES	-	header addressees
* TDOCS_AUTHORS	-	header authors
* TDOCS_ASSIGNED_CODES	-	header assigned codes
* TDOCS_ASSIGNED_NUMBERS	-	assigned numbers
* TDOCS_CIRCULATION	-	document circulation
* TDOCS_REPORTS	-	tdocs reports
* TDOCS_DOCUMENT_SETS	-	old tdocs document sets, now new CDOCS partitions
* TDOCS_DOCUMENT_STATUS	-	document status description
* TDOCS_HYDRO	-	hydrology records
* TDOCS_NIST	-	NIST records
* TDOCS_NIST_ABSTRACTS	-	NIST abstracts
* TDOCS_NIST_KEYWORDS	-	NIST keywords
* TDOCS_NIST_REVIEWS	-	NIST reviews
* TDOCS_NUREG	-	NUREG partition information
* TDOCS_REP	-	CNWRA technical report partition information
* RPD	-	rpdocs records
* RPD_DOCUMENT_SETS	-	old rpd document sets, now new CDOCS partitions
* RPD_REPORTS	-	rpdocs report writer reports
* RPD_PIECES	-	rpdocs report writer pieces

See the following sql scripts in the db workspace directory:

* db/rpdocs_users_create.sql	-	drop and create
* db/rpdocs_help_create.sql	-	drop and create
* db/misc_people_create.sql	-	drop and create
* db/tdocs_1st_create.sql	-	} ordered create
* db/tdocs_2nd_create.sql	-	
* db/tdocs_3rd_create.sql	-	
* db/tdocs_docset_insert.sql	-	document set data
* db/tdocs_major_drop.sql	-	ordered drop
* db/tdocs_minor_drop.sql	-	ordered drop
* db/tdocs_hydro_create.sql	-	create
* db/tdocs_nist_create.sql	-	create
* db/tdocs_nureg_create.sql	-	create
* db/tdocs_rep_create.sql	-	create
* db/rpd_1st_create.sql	-	} ordered create
* db/rpd_2nd_create.sql	-	
* db/rpd_3rd_create.sql	-	
* db/rpd_all_drop.sql	-	ordered drop
* db/rpd_docset_insert.sql	-	document set data

1.3.1.3 SQL*Net

SQL*Net is used for all access to the database so the server can connect to any given database instance, e.g., prod1, test1, test2, etc.

SQL*Net is set up with ORACLE's netman. The network definition file, readable by netman, can be found in:

- * \$ORACLE_HOME/network/netdef.net

Note that in defining connections use full paths instead of \$ORACLE_HOME, which while valid does not work.

It generates a configuration file:

- * \$ORACLE_HOME/network/config/netdef/samson_world/listener.ora

Because of a bug, this file needs to be edited manually. At the end of the file is a SID_LIST - remove excess close parentheses ()). Then copy to

- * /var/opt/oracle/listener.ora

where lsnrctl, the listener control application, will read it when passed the start argument:

- * lsnrctl start

The listener must be started as oracle.

1.3.2 TOPIC

TOPIC is usually accessed by means of system calls—though in some cases file i/o and directory functions are used as well. Manipulating TOPIC is a matter of manipulating files and running scripts that run TOPIC utilities.

1.3.2.1 Directory Structure

The TOPIC directory structure is large:

- * instance = /u03/prod1, u02/test1, etc.
 - +incoming = where batch expects to find files
 - +outgoing = where sync out puts and sync in expects to find files
 - +data = ascii load and link files and file lists
 - +archive = binary versions of files
 - +images = image files
 - +reports = where reports are generated and stored
 - +etc = standard TOPIC directories

In the case of the old RPD partitions, for each document set partition P there is a P directory under incoming and a p_P (active) and a p_P_a (archived) directory under data and archive; file lists are stored in the data directory. In the case of TDOCS, for each document set partition P there is a P directory under incoming and outgoing (if the set is sharable), a p_P directory under data and archive as well as p_Pn subpartition subdirectories under each p_P, and bin and txt directories under archive/p_P/p_Pn and load (files for indexing) and link (files for linking) directories under data/p_P/p_Pn; file lists are stored at the p_P level. TDOCS is more complex because of limitations on the number of command line arguments (wild cards expanded) unix commands like mv, cp, rin, etc. can handle.

1.3.2.2 Tools and Scripts

Related unix script and awk files (in the TOPIC tools directory, and subdirectories RPD-AWK and SECURITY) are:

* rpdocs_delete_doc	-	topic delete script
* rpdocs_dum_asc.awk	-	script to generate vacant ascii cds, cdm, rps, oits documents
* rpdocs_dum_sgm.awk	-	script to generate vacant sgml cds, cdm, rps, oits documents
* rpdocs_insert_doc	-	topic insert script
* rpdocs_link_doc	-	topic ascii/WP link script
* rpdocs_secure_doc	-	topic secure script
* rpdocs_cds.awk	-	awks cds doe set, doe num, and title
* rpdocs_cdm.awk	-	awks cdm doe set, doe num, and title
* rpdocs_rps.awk	-	awks rps doe set, doe num, and title
* rpdocs_oits.awk	-	awks oits doe set, doe num, and title
* rpdocs_retire.awk	-	awks new status and partition for retires
* rpdocs_load_part	-	load and link for tdocs
* rpdocs_tdocs.lnk	-	tdocs link control

1.4 SYSTEM CONFIGURATION

The Consolidated Document Database System (CDOCS) currently runs on the following hardware/software environments:

- SUN (IPX or later)
 - OS Version 4.1.3
 - Solaris Version 2.3
- Personal Computer (80486 or later)
 - Windows Version 3.1
 - OS/2 Version 2.1 or later

- Macintosh
 - System 7 or later

CDOCS currently uses the following commercial products:

- ORACLE Version 7.1.6.4
- TOPIC Versions 3.1.5 or 4.0
- GALAXY Versions 2.0 or 2.5

In the future, updates to the commercial software may be required when new versions are introduced.

The source code is under configuration control at the CNWRA using Source Code Control System (SCCS) through SPARCworks/Teamwear. The computer codes for CDOCS are currently controlled under the CNWRA Software Configuration Procedure TOP-018.

2 CDOCS FUNCTIONAL DESCRIPTION AND LOGIC FLOW

This chapter provides technical documentation of the functions and procedures for the CDOCS system.

2.1 INTRODUCTION

The technical documentation for the CDOCS system is intended to provide guidance to enable qualified technical staff to understand the structure of the code and the relationships between the various functions. A thorough understanding of the C programming language, the structured query language (SQL), remote procedure call (RPC) programming, the Galaxy graphical user interface (GUI) system, and the Oracle database is assumed.

2.1.1 Typographic Conventions

Throughout the CDOCS system document, the following typographic conventions are used:

- Screen menu names, menu items, and push-button names appear in italics with initial caps or as they appear on the screen

Example: Select *CDOCS* from the menu bar of the *CDOCS Main Menu*

- Function and variable names appear in italics with caps as required to represent case-sensitive names

Example: *TDOCSCheckinNamesCancel*

- User input and system prompts appear in bold Courier typeface

Example: Enter the path and file name: **c:\wp51\wp.exe**

- Keystroke input from the keyboard appears as initial caps inside square brackets

Example: Press [F1], then [Return]

- Keystroke input from the keyboard using combinations of keys (e.g., hold down the Ctrl key and press H) appears as initial caps, with the combination of keys joined by a plus sign, inside square brackets

Example: Press [Ctrl+H]

2.2 CLIENT SYSTEM TECHNICAL DOCUMENTATION

The CDOCS client system is event driven. Therefore, the flow of control in the system is not linear, but is determined by the sequence of events initiated by the user. Accordingly, the primary emphasis of the CDOCS client technical documentation is the identification of events and callbacks, and the association of these events and callbacks with the relevant code.

The organization of the CDOCS client system technical documentation follows the CDOCS User Guide so that actions and events initiated by the user may be readily associated with the corresponding code. Tables for callbacks and events are included for reference, but flowcharts have not been included because such visual representations are not appropriate for event-driven systems.

2.2.1 Overall System Flow

The CDOCS is a client server system in which functional support for the user interaction and interface is implemented on the client platform, and database functionality is implemented on the server platform. Client support in the CDOCS is implemented for four different hardware/software environments that are functionally equivalent but differ in some details of the presentation of the user interface. The server support is implemented on a Unix-based Sun Workstation that utilizes the facilities of the Oracle database and the Topic full-text search and retrieval software. Communication between the client and server platforms is accomplished through Remote Procedure Call (RPC) facilities. This isolates platform dependencies and permits the client and server implementations to be completely independent of each other.

2.2.1.1 Client Server Interaction

Figure 2-1 illustrates the overall flow between the client and server modules. The client process is completely event driven. User actions, such as selecting a push-button or drop down list on the screen, cause events to be recognized and associated functions to be executed. These user-initiated events are recognized in two ways. Some events are recognized through a large case statement, known as the main event loop, that is embedded in the main client program. Many user events set conditions that are recognized by the main event loop, and the associated client functions are called. Other events are associated with particular Galaxy screen elements through callbacks. When these screen elements are selected by the user, the corresponding callbacks are activated and their associated functions are called. It is important to note that while the main event loop and the Galaxy callback methods of recognizing events are independent and mutually exclusive for a particular event, they are entirely equivalent. When an event is recognized, the associated client function is called, either by a callback or through the main event loop. The client function performs specific processing for the event and then calls another function to set up the RPC call to the server. Generally, the client functionality is suspended until the server responds to the RPC call.

The main server program receives the RPC packet and passes it to the *svc_N* function for interpretation. The *svc_N* function is very similar to the client's main event loop. The service request is interpreted through a large case statement and the associated function is called to perform the desired processing. The server processing functions differ considerably in the specific processing performed, but they are similar in that they interact with the database, the file system and Topic to store and retrieve information. All database activities are SQL statements and queries that are prepared and passed to Oracle. Topic facilities are generally accessed by building a script and then invoking that script to execute a Topic utility for inserting a record or updating a Topic index. Other capabilities, such as invoking WordPerfect, copying a file, deleting a file, or moving a file from one directory to another, are performed by building a Unix script and then executing that script. When the requested server function completes, a return code along with any data is passed back to the client through RPC.

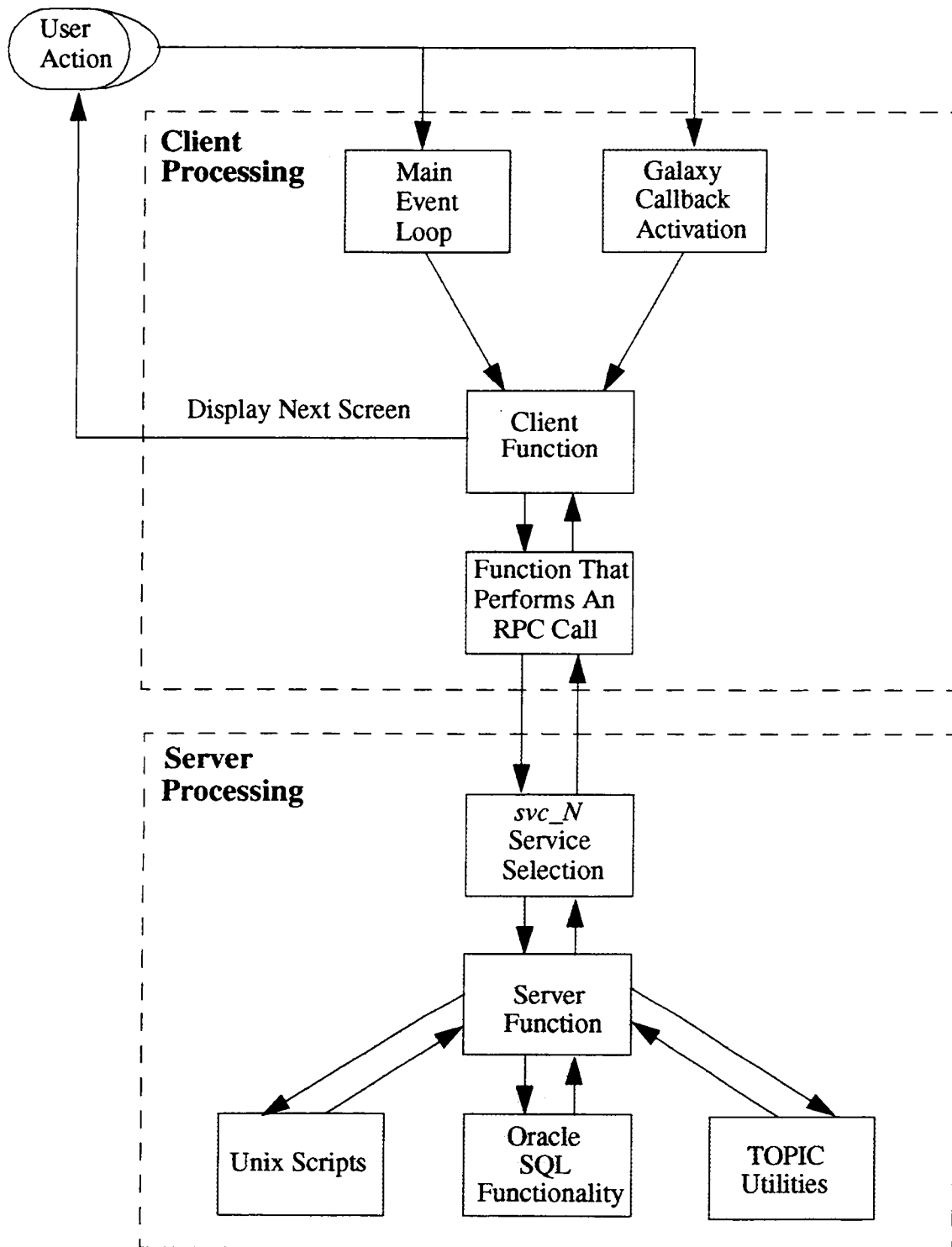


Figure 2-1. Client-server flow

The client function examines the return code and data received from the server through the RPC call, and performs any additional processing that may be required. When all client processing for the event is complete, the client function loads the next screen to be displayed and terminates. This causes the desired screen to be displayed, and the client functionality then waits for the next user-initiated event.

2.2.1.2 Client Server Interaction Using Callbacks

Figure 2-2 illustrates the flow of client and server processing that results from the user selecting the *Submit* push-button from the *CDOCS Submit* screen. When the user selects the *Submit* push-button, the *submitButton* callback is activated. This causes the *buttonSubmit* function to be called. After validating and formatting the document submission, the *buttonSubmit* function calls the *RPDSubmitRecord* function to format the RPC message and pass the document submission information to the server. The *RPDSubmitRecord* function makes the RPC call to the server with a request code of *TDOCS_SUBMIT_DOC_REQUEST*.

When the server receives the RPC call, the request code and data are stored and passed to the *svc_N* function for interpretation. The *svc_N* function calls the *TDOCSSubmitDocument* function to perform the server processing for the submit transaction. During the processing of the submit transaction, the submitted files must be moved and renamed. The function prepares a Unix script and executes it with a *System* command. Information for the new record must also be inserted in the SQL database. The function prepares the SQL insert statement and passes it to Oracle for execution. If the submitted document is to be shared, the document files must be stored for subsequent synchronization. The function prepares a Unix script to move the files and executes it with a *System* command. When the *TDOCSSubmitDocument* function completes all server processing for the submit transaction, the return code and document number are passed back to the client through RPC.

The client function stores and examines the return code and document number received from the server through the RPC call, and performs any additional processing that may be required. When all client processing for the submit transaction is complete, the *RPDSubmitRecord* client function loads the next screen to be displayed and terminates. This causes the desired screen to be displayed, and the client then waits for the next user-initiated event.

2.2.1.3 Client Server Interaction Using the Main Event Loop

Figure 2-3 illustrates the flow of client and server processing that results from the user selecting the *Copy* push-button from the *Copy a Regulatory Record* screen. When the user selects the *Copy* push-button, the main event loop recognizes the *_intern_FileCopyOkayButton* event. This causes the *filecopy* function to be called. After validating and formatting the copy request, the *filecopy* function calls the *RPCCopyWP* function to format the RPC message and pass the document copy request to the server. The *RPCCopyWP* function makes the RPC call with a request code of *RPD_BIN_DOC_GET_REQUEST*.

When the server receives the RPC call, the request code and data are stored and passed to the *svc_N* function for interpretation. The *svc_N* function calls the *RPCGetCDSCDMDoc* function to perform the server processing for the file copy transaction. During the processing of the file copy, control information must be retrieved from the SQL database to obtain the partition and file name of the requested file. The function prepares the SQL select statement and passes it to Oracle for execution. The function then reads the binary file, and formats and stores it. When the *RPCGetCDSCDMDoc* function completes

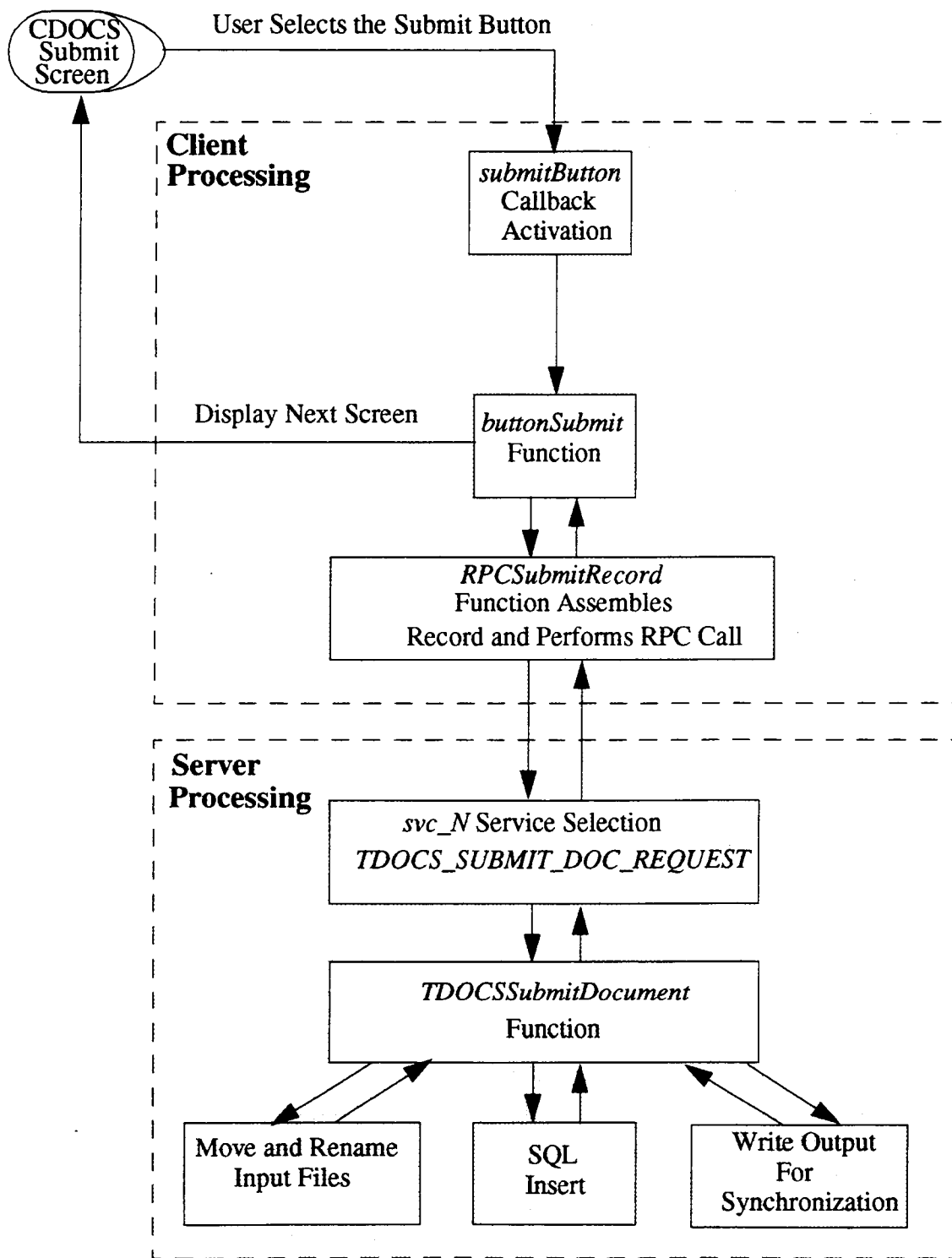


Figure 2-2. Callback functionality

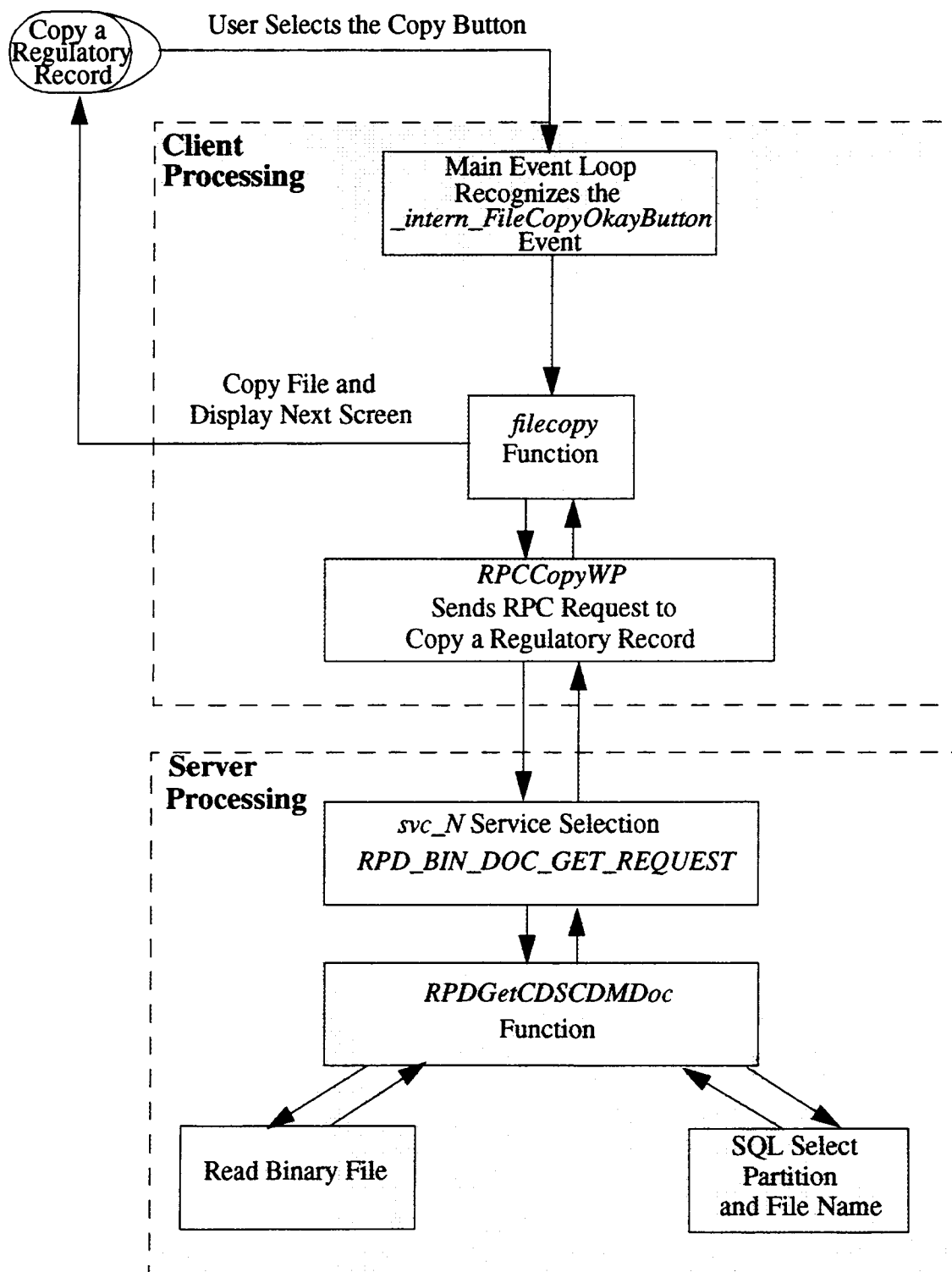


Figure 2-3. Main event loop

all server processing for the file copy transaction, the return code and file data are passed back to the client through RPC.

The client function stores and examines the return code and file data received from the server through the RPC call. When all client processing for the file copy transaction is complete, the *filecopy* client function loads the next screen to be displayed and terminates. This causes the new screen to be displayed, and the client then waits for the another user-initiated event.

2.2.2 Multiplatform Implementation

CDOCS has been implemented in a multiplatform environment allowing it to be accessed by users in diverse hardware/software environments. While CDOCS functions in the same way in all of the supported hardware/software environments, the appearance and function of the system varies slightly from platform to platform. The code for all platforms is consolidated in a single set of modules that include *ifdef* statements to condition platform-dependent code. Table 2-1 documents the variables used in *define* statements to control the generation of code for specific platforms:

Table 2-1. CDOCS Conditional Compilation Defines

Define	Platform	Compiler
<code>__sparc</code>	Sun Microcomputer (SMC)	cc
<code>__WATCOMC__</code> <code>__WINDOWS_386__</code>	MS windows	Watcom C/C++
<code>__MSC__</code> used to build the rpc16.dll only	MS windows	Microsoft C7
<code>__OS2__</code>	IBM OS/2	IBM C/C++ Set
<code>THINK_C</code>	Apple Macintosh	Symantic C

2.2.3 Client Program Initiation

CDOCS is normally started by selecting it on the screen display.

- On a Windows workstation, the CDOCS icon on the Windows desktop is double clicked, and the *CDOCS application* icon appears. The *CDOCS application* icon is double clicked.
- On a Sun workstation, the cursor is positioned on the desktop area of the screen and the right mouse button is pressed (for MOTIF environment). A pull-down menu of functions appears.
- On an OS/2 workstation, the CDOCS icon on the OS/2 desktop is double clicked.

When CDOCS is selected in this way, the system starts and displays the *CDOCS Logon* screen, if required. Users at the CNWRA invoke the *main* program without the “-u” parameter. This causes the

DWM Logon screen to appear. Users at the NRC invoke the *main* program with the “-u” parameter. This causes the logon to occur automatically, but the logon is transparent to the user and the *DWM Logon* screen does not appear.

2.2.3.1 Main Program Initiation

The primary client module, named *main.c*, is started and command line parameters are passed. If the program is invoked with an “-i” command line option, this option supersedes every other option and causes the program to start in “install mode” so that preferences can be set on the local workstation (See Section 2.2.6.5).

Processing of Command Line Parameters

The *init* function is called to initialize the program and load resources. No parameters are passed to the *init* function, and a value of *Null* is returned. The selector for each tag is in the VRE, and it is stored in the command dictionary.

Initialization of Screens and Callbacks

The *CDOCS Main Menu*, with its associated pull-down and cascading menus, is the primary mechanism for selecting client functions. The code that implements this functionality is associated with menu entries through the event loop in the *main* program and through callbacks that associate Galaxy screen elements with the appropriate code. The *TDOCSSStart* function is called to initialize the CDOCS system. *TDOCSSStart* sets up the environment by loading all the dialog boxes into memory and establishing callbacks to the numerous Galaxy buttons and other resources that initiate CDOCS functions. The *TDOCSSStart* function calls a succession of other functions to load the screens into memory and sets callbacks for menu and dialog items.

Initialization of the Help Screens

After the *TDOCSSStart* function is called, the *LoadHelp* function is executed to load the Help and Help Index dialog boxes into memory and set callbacks for the menu and dialog items.

Initialization of the Search Functionality

The *SearchCallBack* function is called to set a callback for the *Search* entry in the *CDOCS Main Menu*. When the *Search* entry is selected from the menu bar of the *CDOCS Main Menu*, the callback for the *MenuSearch* item is activated and the *Search* function is executed to start the TOPIC application software.

2.2.3.2 Logon Processing

A logon is performed for all users, but the type of logon processing depends on command line parameters. For CNWRA users, Database Custodian users, and Database Administrator users, the logon causes the *DWM Logon* screen to appear. For NRC users, the logon is performed automatically, and the *DWM Logon* screen does not appear.

Logon for Setting Preferences

If the command line parameters include the “-i” option, the *basicLogin* function is called by the *main* function. The *basicLogin* function ignores the database connection and displays the *CDOCS Main Menu* with only those menu options required to update the preferences on the client platform.

Automatic Logon for NRC Users

If an “-u” option without an “-i” option is included in the command line, the *automaticLogin* function is called by the *main* program. If the “-i” option is not included in the command line, the code checks to see if the “-u” option was provided. If the “-u” option is present, the *automaticLogin* function is executed to perform an automatic logon to the Oracle database.

The *automaticLogin* function connects the client automatically to the server. The *automaticLogin* function defaults to the CDOCS application server. The *automaticLogin* function connects to the server using the *RPCConnect* function located in *clntrpc* file. The username and the password are hard coded on the parameter list. Therefore, if the *nrc_user*’s password is changed, this hard coded information must be changed as well and the application must be recompiled to generate new executable code.

If the *automaticLogin* function is successful, the function gets the user’s login name and the user privilege. Menu selections are added based on privilege level. If no errors are detected, the function opens the *CDOCS Main Menu* with the appropriate menu selections.

Logon for CNWRA Users

If neither the “-i” nor the “-u” option is provided, the login function is executed and the *DWM Logon* screen is presented to the user. The login function connects to the server. If the connection to the server is successful, the login function adds menu selections based on the user’s privilege level, and opens the *CDOCS Main Menu* with these menu selections.

DWM Logon Screen Functionality

When the command line parameters indicate that a logon is required, the system displays the *DWM Logon* screen (Figure 2-4) to permit the User ID and password to be entered. The system verifies the User ID and the password. If they are valid, the *CDOCS Main Menu* appears. If the User ID or password is not valid, the system displays an error message, and requests re-entry of the User ID and password. If the User ID and password cannot be entered correctly, the user may exit from the system by selecting the *Exit* push-button or pressing the *Esc* key

DWM Logon Screen —Exit Push-Button Processing

The *_intern_LogonCancelButton* event results from selecting the *Exit* push-button at the bottom of the *DWM Logon* screen. This event causes the *veventStopProcessing* function to be executed to terminate the CDOCS system.

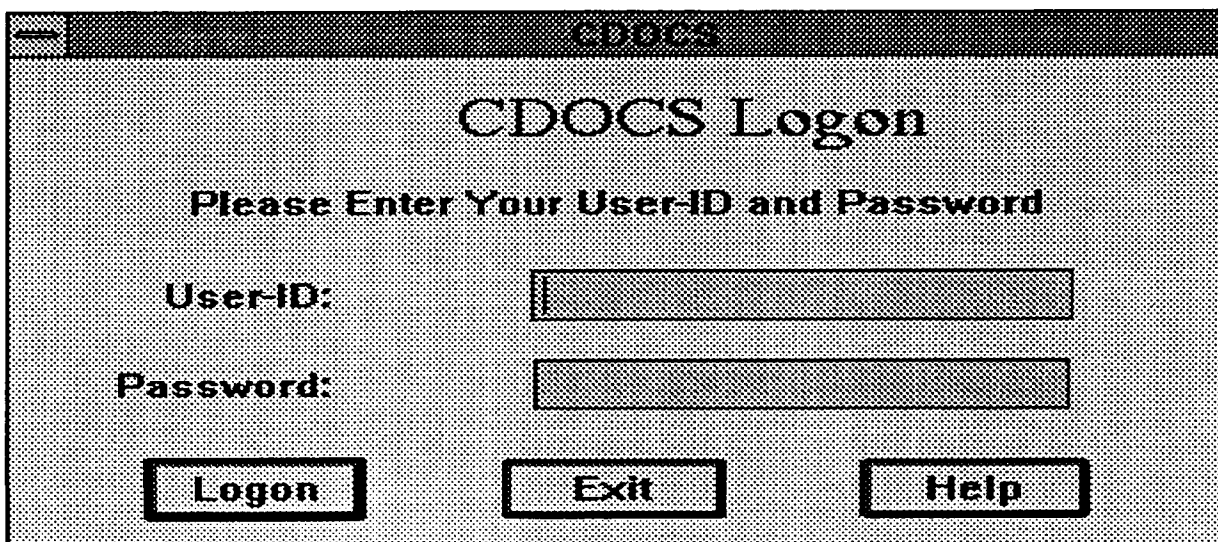


Figure 2-4. DWM logon screen

DWM Logon Screen —Logon Push-Button Processing

The *_intern_LogonOkayButton* event results from selecting the *Logon* push-button at the bottom of the *DWM Logon* screen. This event causes the *login* function to process the logon request, set menu options based on the user's privilege level, and open the *CDOCS Main Menu* with the appropriate menu selections.

2.2.4 CDOCS CALLBACKS

CDOCS utilizes both callbacks and a main event loop to respond to events initiated by the user. A number of callbacks are established during initialization so that associated functions are called

automatically when items are clicked on the Galaxy screens. Most of the callbacks are established in the *TDOCSSStart* function or by other functions called by *TDOCSSStart*. Table 2-2 documents the callbacks that are set for this purpose.

Table 2-2. CDOCS callbacks

Internal Variable Name	Function Called	Description of Callback
MAIN PROGRAM CALLBACKS—Set by <i>SearchCallBack</i>		
<i>MenuSearch</i>	<i>Search()</i>	Activated by selecting the <i>Search</i> entry from the <i>CDOCS Main Menu</i> .
SEARCH CALLBACKS—Set by <i>LoadSearchDialog</i>		
<i>SearchCancel</i>	<i>SearchProc —closeSearch()</i>	Activated by selecting the <i>Cancel</i> push-button from the screen.
<i>SearchOK</i>	<i>SearchProc—BuildLaunchCmd (searchList)</i>	Activated by selecting the <i>OK</i> push-button from the <i>CDOCS Search</i> screen.
<i>SearchSave</i>	<i>SearchProc —SaveSearchQuery (searchList)</i>	Activated by selecting the <i>Save</i> push-button from the <i>CDOCS Search</i> screen.
SUBMIT SCREEN CALLBACKS—Set by <i>TDOCSSStart</i>		
<i>submitButton</i>	<i>buttonSubmit</i>	Activated by selecting the <i>Submit</i> push-button from the <i>CDOCS Submit</i> screen.
<i>clearButton</i>	<i>ResetSubmit</i>	Activated by selecting the <i>Reset</i> push-button from the <i>CDOCS Submit</i> screen.
<i>tdocsCloseButton</i>	<i>buttonClear</i>	Activated by selecting the <i>Cancel</i> push-button from the <i>CDOCS Submit</i> screen.
<i>TDOCSSubmitHelp</i>	<i>Help</i>	Activated by selecting the <i>Help</i> push-button from the screen.
<i>tdocsContainer</i>	<i>expandView</i>	Activated by selecting the “...” push-button from the <i>CDOCS Submit</i> screen.
<i>documentMenu</i>	<i>OptionProc</i>	Activated by selecting the <i>Document Type</i> pull-down indicator from the <i>CDOCS Submit</i> screen.
UPDATE ENTRY SCREEN—Set by <i>LoadTDOCUpdate</i>		
<i>Update OK Tag</i>	<i>TDOCUpdateProc—TDOCUpdateGetRec()</i>	Activated by selecting the <i>OK</i> push-button from the <i>CDOCS Update</i> entry screen.
<i>Update Cancel Tag</i>	<i>TDOCUpdateProc—vdialogclose (TDOCUpdateDialog)</i>	Activated by selecting the <i>Close</i> push-button from the <i>CDOCS Update</i> entry screen.

Table 2-2. CDOCS Callbacks (cont'd)

Internal Variable Name	Function Called	Description of Callback
DELETE DOCUMENT SCREEN CALLBACKS—Set by <i>LoadDelete</i>		
<i>Delete OK Tag</i>	<i>DeleteProc—loadSubmit</i>	Activated by selecting the <i>Delete</i> push-button from the <i>CDOCS Delete</i> document screen.
<i>Delete Cancel Tag</i>	<i>DeleteProc—vdialogClose (deleteDialog)</i>	Activated by selecting the <i>Close</i> push-button from the <i>CDOCS Delete</i> document screen.
<i>TDOCSDeleteHelp</i>	<i>Help</i>	Activated by selecting the <i>Help</i> push-button from the <i>CDOCS Delete</i> document screen.
CHECK IN NAMES CALLBACKS—Set by <i>LoadTDOCSCheckinNames</i>		
<i>TDOCSCheckinNamesOK</i>	<i>CheckinNamesProc—RPCCheckinNames()</i>	Activated by selecting the <i>OK</i> push-button from the <i>Checkin Names</i> screen.
<i>TDOCSCheckinNamesCancel</i>	<i>CheckinNamesProc—vdialogClose (TDOCSCheckinNamesDialog)</i>	Activated by selecting the <i>Close</i> push-button from the <i>Checkin Names</i> screen.
FORCE CHECKOUT CALLBACKS—Set by <i>LoadForceCheckout</i>		
<i>TDOCSForceCancel</i>	<i>vdialogClose (TDOCSCheckinNamesDialog)</i>	Activated by selecting the <i>Cancel</i> push-button from the <i>CDOCS Checkout error message</i> screen.
<i>TDOCSForceOK</i>	<i>ForceCheckoutProc</i>	Activated by selecting the <i>OK</i> push-button from the <i>CDOCS Checkout error message</i> screen.
REPORT WRITER SELECTION LIST CALLBACKS—Set by <i>LoadMenuReportWriter</i>		
<i>DatabaseReportCloseButton</i>	<i>MenuReportWriterProc—vdialogClose (menuReportWriterDialog)</i>	Activated by selecting the <i>Close</i> push-button from the <i>Database Reports Selection List</i> screen.
<i>DatabaseReportViewButton</i>	<i>MenuReportWriterProc—RPCReportWriter</i>	Activated by selecting the <i>View</i> push-button from the <i>Database Reports Selection List</i> screen.
<i>DatabaseReportsSubsetButton</i>	<i>MenuReportWriterProc—loadSubsetReport</i>	Activated by selecting the <i>Subset</i> push-button from the <i>Database Reports Selection List</i> screen.

Table 2-2. CDOCS Callbacks (cont'd)

Internal Variable Name	Function Called	Description of Callback
REFERENCE REPORTS SELECTION SCREEN CALLBACKS—Set by <i>LoadRefReport</i>		
<i>ReferenceReportCloseButton</i>	<i>RefReportProc— vdialogClose(refReportDialog)</i>	Activated by selecting the <i>Close</i> push-button from the <i>CDOCS Reference Report</i> selection screen.
<i>ReferenceReportOKButton</i>	<i>RefReportProc— RPCGetReferenceReport</i>	Activated by selecting the <i>OK</i> push-button from the <i>CDOCS Reference Report</i> selection screen.
CIRCULATION REPORTS CALLBACKS—Set by <i>LoadCirculation</i>		
<i>CirculationCancelButton</i>	<i>CirculationProc—vdialogClose (circulationDialog)</i>	Activated by selecting the <i>Close</i> push-button from the <i>CDOCS Circulation Reports</i> selection screen.
<i>CirculationOKButton</i>	<i>CirculationProc— RPCGetCirculationReport</i>	Activated by selecting the <i>OK</i> push-button from the <i>CDOCS Circulation Reports</i> selection screen.
<i>TDOCSCirculationHelp</i>	<i>Help</i>	Activated by selecting the <i>Help</i> push-button from the <i>CDOCS Circulation Reports</i> selection screen.
NEW ACQUISITIONS REPORT CALLBACKS—Set by <i>LoadNewDocs</i>		
<i>NewDocs OK Button</i>	<i>NewDocsProc</i>	Activated by selecting the <i>OK</i> push-button from the <i>CDOCS New Acquisitions Reports</i> selection screen.
<i>NewDocs Cancel Button</i>	<i>NewDocsProc</i>	Activated by selecting the <i>Close</i> push-button from the <i>CDOCS New Acquisitions Reports</i> selection screen.
<i>TDOCSNewDocsHelp</i>	<i>Help</i>	Activated by selecting the <i>Help</i> push-button from the <i>CDOCS New Acquisitions Reports</i> selection screen.
REPORT DISPLAY SCREEN CALLBACKS—Set by <i>loadreport</i>		
<i>TDOCSCirculationHelp</i>	<i>Help</i>	Activated by selecting the <i>Help</i> push-button from the <i>CDOCS Report display</i> screen
SAVE AS SCREEN CALLBACKS—Set by <i>LoadSaveAs</i>		
<i>ReportHelp</i>	<i>SaveAsProc</i>	Activated by selecting the <i>Help</i> push-button from the <i>CDOCS Save As display</i> screen

Table 2-2. CDOCS Callbacks (cont'd)

Internal Variable Name	Function Called	Description of Callback
CHECKOUT SCREEN CALLBACKS—Set by <i>LoadCheckout</i>		
<i>Checkout OK Tag</i>	<i>CheckoutProc</i>	Activated by selecting the <i>OK</i> push-button from the <i>CDOCS Checkout</i> screen.
<i>Checkout Cancel Tag</i>	<i>CheckoutProc</i>	Activated by selecting the <i>Close</i> push-button from the <i>CDOCS Checkout</i> screen
<i>TDOCSCheckoutHelp</i>	<i>Help</i>	Activated by selecting the <i>Help</i> push-button from the <i>CDOCS Checkout</i> screen
CHECKIN SCREEN CALLBACKS—Set by <i>LoadTDOCSCheckin</i>		
<i>Checkin OK Tag</i>	<i>CheckinProc</i>	Activated by selecting the <i>OK</i> push-button from the <i>CDOCS Checkin</i> screen.
<i>Checkin Cancel Tag</i>	<i>CheckinProc</i>	Activated by selecting the <i>Close</i> push-button from the <i>CDOCS Checkin</i> screen
<i>TDOCSCheckinHelp</i>	<i>Help</i>	Activated by selecting the <i>Help</i> push-button from the <i>CDOCS Checkin</i> screen
CONFIRM SCREEN CALLBACKS—Set by <i>LoadConfirm</i>		
<i>TDOCSConfirmOK</i>	<i>ConfirmProc</i>	Activated by selecting the <i>OK</i> push-button from the <i>CDOCS Confirm</i> screen
<i>TDOCSConfirmCancel</i>	<i>ConfirmProc</i>	Activated by selecting the <i>Cancel</i> push-button from the <i>CDOCS Confirm</i> screen
<i>TDOCSConfirmHelp</i>	<i>Help</i>	Activated by selecting the <i>Help</i> push-button from the <i>CDOCS Confirm</i> screen
DEFINE A REGULATORY RECORD CALLBACKS—Set by <i>loadDefine</i>		
<i>DefineHelp</i>	<i>Help</i>	Activated by selecting the <i>Help</i> push-button from the <i>Define a Regulatory Program Record</i> entry screen.
ENTER A REGULATORY RECORD CALLBACKS—Set by <i>loadCheckin</i>		
<i>CheckinHelp</i>	<i>Help</i>	Activated by selecting the <i>Help</i> push-button from the <i>Enter a Regulatory Program Record</i> entry screen.
FORMAT CHECK A REGULATORY RECORD CALLBACKS—Set by <i>loadFormatCheck</i>		
<i>FormatCheckHelp</i>	<i>Help</i>	Activated by selecting the <i>Help</i> push-button from the <i>Format Check a Regulatory Program Record</i> entry screen.

Table 2-2. CDOCS Callbacks (cont'd)

Internal Variable Name	Function Called	Description of Callback
FORMAT CHECK ERRORS CALLBACKS—Set by <i>formatCheck</i>		
<i>FormatCheckAbortHelp</i>	<i>Help</i>	Activated by selecting the <i>Help</i> push-button from the <i>Format Check Errors</i> screen.
RETIRE CALLBACKS—Set by <i>loadRetirement</i>		
<i>RetirementHelp</i>	<i>Help</i>	Activated by selecting the <i>Help</i> push-button from the <i>Retire a Regulatory Program Record</i> screen.
RETIRE CONFIRMATION CALLBACKS—Set by <i>retire</i>		
<i>RetireConfirmHelp</i>	<i>Help</i>	Activated by selecting the <i>Help</i> push-button from the <i>Retire a Regulatory Program Record Confirmation</i> screen.
USER ID MAINTENANCE CALLBACKS—Set by <i>loadUserIDs</i>		
<i>intern_SystemUserIDsListView</i>	<i>selectIDChange</i>	Activated by selecting the <i>User ID</i> list view in the <i>User ID Maintenance</i> screen.
<i>SystemUserIDsHelp</i>	<i>Help</i>	Activated by selecting the <i>Help</i> push-button from the <i>User ID Maintenance</i> screen.
HELP CALLBACKS—Set by <i>LoadHelp</i>		
<i>HelpCancel</i>	<i>HelpProc</i>	Activated by selecting the <i>Close</i> push-button in <i>CDOCS Help</i> screen.
<i>HelpIndexCancel</i>	<i>HelpIndexProc</i>	Activated by selecting the <i>Close</i> push-button in <i>CDOCS Help Index</i> screen.
<i>HelpIndexOK</i>	<i>HelpIndexProc</i>	Activated by selecting the <i>OK</i> push-button in <i>CDOCS Help Index</i> screen.
DOCUMENT MENU CALLBACKS—Set by <i>TDOCSSStart</i>		
<i>documentMenu</i>	<i>OptionProc</i>	

2.2.5 Main Event Loop

When the logon processing has been completed, the *veventProcess* function in the *main.c* program is executed to start the main event loop. The *veventProcess* function continues to execute until the Exit option is selected from the Operations pull-down menu. When the *veventProcess* main event loop function terminates, the *RPCDisconnect*, *vstartupTDOCSSStop*, and *cleanup* functions are executed to terminate the CDOCS application.

The CDOCS main event loop is controlled by a large case statement in the `issueCmds` function. Table 2-3 outlines the cases in the `issueCmds` function and the resulting function executions.

Table 2-3. CDOCS event definitions

Internal Variable Name	Function Called	Description
CDOCS Main Menu		
<code>_intern_MenuHelpHelp</code>	<code>BuildHelpIndex</code>	Results from selecting the <i>Help</i> entry from the <i>Help</i> pull-down menu.
Operations Pull-Down Menu		
<code>_intern_MenuOperationsCopy</code>	<code>loadFileCopy</code>	Results from selecting the <i>Copy (CDS, CDM, Open Items)</i> entry from the <i>Operations</i> pull-down menu.
<code>_intern_MenuOperationsWP</code>	<code>loadFileWP</code>	Results from selecting the <i>WP (CDS, CDM, Open Items)</i> entry from the <i>Operations</i> pull-down menu.
<code>_intern_MenuOperationsChangePW</code>	<code>loadchangePWdialog</code>	Results from selecting the <i>Change Password</i> entry from the <i>Operations</i> pull-down menu.
<code>_intern_MenuOperationsScan</code>	<code>onDemandScan</code>	Results from selecting the <i>Scan and OCR</i> entry from the <i>Operations</i> pull-down menu.
<code>_intern_MenuOperationsExit</code>	<code>veventStopProcessing</code>	Results from selecting the <i>Exit</i> entry from the <i>Operations</i> pull-down menu.
DWM Logon		
<code>_intern_LogonOkayButton</code>	<code>login</code>	Results from selecting the <i>Logon</i> push-button from the <i>DWM Logon</i> screen.
<code>_intern_LogonCancelButton</code>	<code>veventStopProcessing</code>	Results from selecting the <i>I</i> push-button from the <i>DWM Logon</i> screen.
Copy a Regulatory Program Record		
<code>_intern_FileCopyOkayButton</code>	<code>filecopy</code>	Results from selecting the <i>Copy</i> push-button from the <i>Copy a Regulatory Program Record</i> screen.
<code>_intern_FileCopyCancelButton</code>	<code>closeFileCopy</code>	Results from selecting the <i>Close</i> push-button from the <i>Copy a Regulatory Program Record</i> screen.
<code>_intern_FileCopyDocType</code>	<code>loadDocs</code>	Results from selecting the <i>Document Type</i> drop down list from the <i>Copy a Regulatory Program Record</i> screen.
<code>_intern_FileCopyChooserButton</code>	<code>filecopyfilechooser</code>	Results from selecting the <i>File Chooser</i> push-button from the <i>Copy a Regulatory Program Record</i> screen.

Table 2-3. CDOCS Event Definitions (Cont'd)

Internal Variable Name	Function Called	Description
Edit a Regulatory Program Record Using WordPerfect		
<i>_intern_FileWPOkayButton</i>	<i>fileWP</i>	Results from selecting the <i>Edit</i> push-button from the <i>Edit a Regulatory Program Record Using WordPerfect</i> screen.
<i>_intern_FileWPCancelButton</i>	<i>closeFileWP</i>	Results from selecting the <i>Close</i> push-button from the <i>Edit a Regulatory Program Record Using WordPerfect</i> screen.
<i>_intern_FileWPDocType</i>	<i>loadDocs</i>	Results from selecting the <i>Document Type</i> drop down list from the <i>Edit a Regulatory Program Record Using WordPerfect</i> screen.
Change Password		
<i>_intern_ChangePWButton</i>	<i>changePassword</i>	Results from selecting the <i>Change Password</i> entry from the <i>Operations</i> pull-down menu.
<i>_intern_PWCloseButton</i>	<i>i</i>	Results from selecting the <i>Close</i> push-button from the <i>Change Password</i> screen.
Preferences		
<i>_intern_MenuOperationsPrefs</i>	<i>loadPrefsdialog</i>	Results from selecting the <i>Preferences</i> entry from the <i>Operations</i> pull-down menu.
<i>_intern_PrefsSaveButton</i>	<i>savePrefs</i>	Results from selecting the <i>Save</i> push-button from the <i>Preferences</i> screen.
<i>_intern_PrefsCancelButton</i>	<i>vdialogClose</i> (<i>prefsdialog</i>)	Results from selecting the <i>Close</i> push-button from the <i>Preferences</i> screen.
Custodian Pull-Down Menu		
Define a Regulatory Program Record		
<i>_intern_DefineDefine</i>	<i>loadDefine</i>	Results from selecting the <i>Define a Regulatory Record</i> entry in the <i>Process a Regulatory Record</i> cascading menu.
<i>_intern_DefineDocType</i>	<i>Code embedded in the case statement in the main program</i>	Results from selecting the <i>Document Type</i> pull-down indicator from the <i>Define a Regulatory Record</i> screen or <i>Define an OITS Record</i> screen.
<i>_intern_DefineOkayButton</i>	<i>define</i>	Results from selecting the <i>Define</i> push-button from the <i>Define a Regulatory Record</i> screen or <i>Define an OITS Record</i> screen.
<i>_intern_DefineCancelButton</i>	<i>closeDefine</i>	Results from selecting the <i>Close</i> push-button from the <i>Define a Regulatory Record</i> screen or <i>Define an OITS Record</i> screen.

Table 2-3. CDOCS Event Definitions (Cont'd)

Internal Variable Name	Function Called	Description
Enter a Regulatory Program Record		
<i>_intern_CheckinCheckin</i>	<i>loadCheckin</i>	Results from selecting the <i>Enter a Regulatory Program Record</i> entry in the <i>Process a Regulatory Record</i> cascading menu.
<i>_intern_CheckinDocType</i>	<i>Embedded code within the main event loop.</i>	Results from selecting the <i>Document Type</i> drop-down indicator from the <i>Enter a Regulatory Program Record</i> screen or <i>Enter an OITS Record</i> screen.
<i>_intern_CheckInFileChooserButton</i>	<i>checkinfilechooser</i>	Results from selecting the <i>File Chooser</i> push-button from the <i>Enter a Regulatory Program Record</i> screen or <i>Enter an OITS Record</i> screen.
<i>_intern_CheckinOkayButton</i>	<i>checkin</i>	Results from selecting the <i>Enter</i> push-button from the <i>Enter a Regulatory Program Record</i> screen or <i>Enter an OITS Record</i> screen.
<i>_intern_CheckinCancelButton</i>	<i>closeCheckin</i>	Results from selecting the <i>Close</i> push-button from the <i>Enter a Regulatory Program Record</i> screen or <i>Enter an OITS Record</i> screen.
<i>_intern_CheckinAbortCloseButton</i>	<i>vdialogClose (checkinabortdialog) abortcheckin</i>	Results from selecting the <i>Close</i> push-button from the <i>Enter a Regulatory Program Record</i> screen or <i>Enter an OITS Record</i> screen.
<i>_intern_CheckinForceForceButton</i>	<i>forcecheckin</i>	Results from selecting the <i>Force Enter</i> push-button from the <i>Enter a Regulatory Program Record</i> screen or <i>Enter an OITS Record</i> screen.
<i>_intern_CheckinForceCancelButton</i>	<i>abortcheckin</i>	Results from selecting the <i>Close</i> push-button from the <i>Enter a Regulatory Program Record</i> error message screen or <i>Enter an OITS Record</i> error message screen.
Format Check a Regulatory Program Record		
<i>_intern_Checkin</i>	<i>FormatloadFormatCheck</i>	Results from selecting the <i>Enter a Regulatory Record</i> entry in the <i>Process a Regulatory Record</i> cascading menu.
<i>_intern_FormatCheckDocType</i>	<i>Embedded code within the main event loop.</i>	Results from selecting the <i>Document Type</i> pull-down indicator from the <i>Format Check a Regulatory Record</i> screen or <i>Format Check an OITS Record</i> screen.
<i>_intern_FormatCheckFileChooser Button</i>	<i>formatcheckfilechooser</i>	Results from selecting the <i>File Chooser</i> push-button from the <i>Format Check a Regulatory Record</i> screen or <i>Format Check an OITS Record</i> screen.

Table 2-3. CDOCS Event Definitions (Cont'd)

Internal Variable Name	Function Called	Description
<i>_intern_FormatCheckOkayButton</i>	<i>formatCheck</i>	Results from selecting the <i>Format Check</i> push-button from the <i>Format Check a Regulatory Record</i> screen or <i>Format Check an OITS Record</i> screen.
<i>_intern_FormatCheckCancelButton</i>	<i>closeFormatCheck</i>	Results from selecting the <i>Close</i> push-button from the <i>Format Check a Regulatory Record</i> screen or <i>Format Check an OITS Record</i> screen.
<i>_intern_FormatCheckAbortClose Button</i>	<i>vdialogClose</i> (<i>formatcheckaboridialog</i>)	Results from selecting the <i>Cancel</i> push-button from the <i>Format Check a Regulatory Record</i> screen or <i>Format Check an OITS Record</i> screen.
Retire a Regulatory Program Record		
<i>_intern_MainRetirement</i>	<i>loadRetirement</i>	Results from selecting the <i>Retire a Regulatory Record</i> entry in the <i>Process a Regulatory Record</i> cascading menu.
<i>_intern_RetirementDocType</i>	<i>Embedded code within the main event loop that calls the loadDocs function</i>	Results from selecting the <i>Document Type</i> pull-down indicator from the <i>Retire a Regulatory Record</i> screen.
<i>_intern_RetirementOkayButton</i>	<i>retire</i>	Results from selecting the <i>Retire</i> push-button from the <i>Retire a Regulatory Record</i> screen.
<i>_intern_RetirementCancelButton</i>	<i>closeRetirement</i>	Results from selecting the <i>Close</i> push-button from the <i>Retire a Regulatory Record</i> screen.
<i>_intern_ConfirmRetireRetire</i>	<i>reallyretire</i>	Results from selecting the <i>Retire</i> push-button from the <i>Retire a Regulatory Record</i> confirmation screen.
<i>_intern_ConfirmRetireCancel</i>	<i>dontretire</i>	Results from selecting the <i>Close</i> push-button from the <i>Retire a Regulatory Record</i> confirmation screen.
Update a CDOCS Record		
<i>_intern_CustodianUpdate</i>	<i>TDOCSUpdate</i>	Results from selecting the <i>Update</i> entry in the from the <i>CDOCS Update</i> screen.
Delete a CDOCS Record		
<i>_intern_CustodianDelete</i>	<i>Delete</i>	Results from selecting the <i>Delete</i> entry in the <i>CDOCS Custodian</i> pull-down menu.

Table 2-3. CDOCS Event Definitions (Cont'd)

Internal Variable Name	Function Called	Description
Submit a CDOCS Record		
<i>_intern_CustodianSubmit</i>	<i>loadSubmit</i> [(vchar *)"Submit"]	Results from selecting the <i>Submit</i> entry in the <i>CDOCS Custodian</i> pull-down menu.
<i>_intern_tdocsCloseButton</i>	<i>closeSubmit</i>	Results from selecting the <i>Close</i> push-button from the <i>CDOCS Submit</i> screen.
Reports—Database Content Cascading Menu		
<i>_intern_MenuReportContent</i>	<i>loadreport</i> (RPD_RP_CONTENT _RPT_REQUEST)	Results from selecting the <i>Regulatory Records Content Report</i> entry in the <i>Database Content</i> (CDS, CDM, Open Items) cascading menu.
<i>_intern_MenuReportStatus</i>	<i>loadreport</i> (RPD_RP_STATUS_RPT _REQUEST)	Results from selecting the <i>Regulatory Records Status Report</i> entry in the <i>Database Content</i> (CDS, CDM, Open Items) cascading menu.
<i>_intern_ReportOITSStatus</i>	<i>loadreport</i> (RPD_OI_STATUS_RPT _REQUEST)	Results from selecting the <i>OITS Status Report</i> entry in the <i>Database Content</i> (CDS, CDM, Open Items) cascading menu.
<i>_intern_MenuReportDump</i>	<i>loadreport</i> (RPD_DBA_RPT _REQUEST)	Results from selecting the <i>Database Report</i> entry in the <i>Database Reports</i> (CDS, CDM, Open Items) cascading menu.
<i>_intern_MenuReportDBA</i>	<i>loadreport</i> (RPD_SHORT_DBA_RPT _REQUEST)	Results from selecting the <i>Database Report</i> entry in the <i>Database Reports</i> (CDS, CDM, Open Items) cascading menu.
Report Pull-down Menu		
<i>_intern_MenuReportDBStats</i>	<i>loadreport</i> (TDOCS _STATS_RPT_REQUEST)	Results from selecting the <i>Database Statistics</i> entry in the <i>Report</i> pull-down menu.
<i>_intern_tdocsLabels</i>	<i>GetReportFileName</i> (0)	Results from selecting the <i>Labels</i> entry in the <i>Report</i> pull-down menu.
<i>_intern_tdocsCirculation</i>	<i>GetReportFileName</i> (2)	Results from selecting the <i>Circulation</i> entry in the <i>Report</i> pull-down menu.
<i>_intern_tdocsNewDocs</i>	<i>GetReportFileName</i> (3)	Results from selecting the <i>New Acquisition</i> entry in the <i>Report</i> pull-down menu.
<i>_intern_MenuReportWriter</i>	<i>MenuReportWriter</i>	Results from selecting the <i>Database Reports</i> entry in the <i>Report</i> pull-down menu.
<i>_intern_MenuReportReference</i>	<i>RefReport</i>	Results from selecting the <i>Reference Reports</i> entry in the <i>Report</i> pull-down menu.

Table 2-3. CDOCS Event Definitions (Cont'd)

Internal Variable Name	Function Called	Description
Reports Display		
<i>_intern_ReportPrintButton</i>	<i>printreport</i>	Results from selecting the <i>Print</i> push-button from the <i>CDOCS</i> report display screen.
<i>_intern_ReportCloseButton</i>	<i>vdialogClose</i> (<i>reportdialog</i>)	Results from selecting the <i>Close</i> push-button from the <i>CDOCS</i> report display screen.
Save As Display		
<i>_intern_TDOCSSaveAsPrint_</i>	<i>printreport</i>	Results from selecting the <i>Save As</i> push-button from the <i>CDOCS</i> report display screen.
User-ID Maintenance		
<i>_intern_MenuUserIDs</i>	<i>loadUserIDs</i>	Results from selecting the <i>User IDs</i> entry from the <i>System</i> pull-down menu.
<i>_intern_SystemUserIDsAddButton</i>	<i>addUserIDs</i>	Results from selecting the <i>Add</i> push-button from the <i>User ID Maintenance</i> screen.
<i>_intern_SystemUserIDsChangeButton</i>	<i>changeUserIDs</i>	Results from selecting the <i>Change</i> push-button from the <i>User ID Maintenance</i> screen.
<i>_intern_SystemUserIDsDeleteButton</i>	<i>deleteUserIDs</i>	Results from selecting the <i>Delete</i> push-button from the <i>User ID Maintenance</i> screen.
<i>_intern_SystemUserIDsCloseButton</i>	<i>vdialogClose</i> (<i>systemuseridsdialog</i>)	Results from selecting the <i>Close</i> push-button from the <i>User ID Maintenance</i> screen.
Help		
<i>_intern_MenuHelpAbout</i>	<i>openDialog</i> (<i>aboutdialog</i>)	Results from selecting the <i>About</i> entry from the <i>Help</i> pull-down menu.
<i>_intern_AboutCloseButton</i>	<i>vdialogClose</i> (<i>aboutdialog</i>)	Results from selecting the <i>Close</i> entry from the <i>About CDOCS</i> screen.
Circulation Control		
<i>_intern_CustodianCheckout</i>	<i>Checkout</i>	Results from selecting the <i>Checkout Document</i> entry from the <i>Custodian</i> Pull-down Menu.
<i>_intern_CustodianCheckin</i>	<i>TDOCSCheckin</i>	Results from selecting the <i>Checkout Document</i> entry from the <i>Custodian</i> Pull-down Menu.

2.2.6 Consolidated Document System Main Menu

After CDOCS has been successfully started, the *CDOCS Main Menu* screen is displayed (Figure 2-5). The menu bar in this screen contains the major functions that the user is permitted to perform. The options displayed depend on the privileges associated with the user.

Operations—Copy a regulatory program or Open Item Tracking System (OITS) record, use WordPerfect to edit a regulatory program or OITS record, change passwords (for CNWRA users), or exit from CDOCS

Search—Use information searches to find and access CDOCS records

Custodian—Perform operations to submit, update, delete, check-out/in CDOCS records and define, checkin, and retire regulatory program and OITS records

Report—Access CDOCS report facilities to display and/or print standard reports

System—Use system maintenance functions

Help—Access CDOCS help facilities to get more information about how to use the CDOCS

2.2.7 CDOCS Main Menu—Operations Entry

The *Operations* main menu entry permits general functions that are available to all users to be performed. Selecting *Operations* from the menu bar of the *CDOCS Main Menu* causes the *Operations* pull-down menu (Figure 2-6) to appear. Depending on the privilege level of the user, this pull-down menu may contain functions for (i) copying a regulatory program or OITS record, (ii) using WordPerfect to edit a regulatory program or OITS record, (iv) changing user passwords, (v) scanning and OCR, (vi) setting preferences, or (vii) exiting CDOCS.

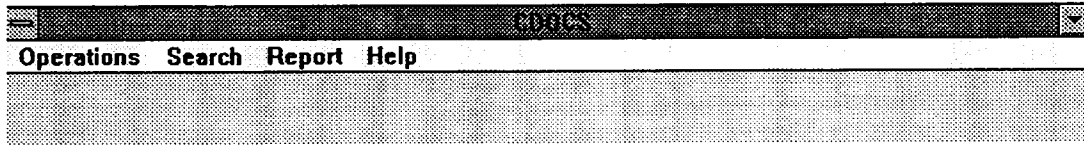
2.2.7.1 Operations—Copying Regulatory Program and Open Item Records

The *Copy* entry in the *Operations* pull-down menu may be used to make electronic copies of information in the database by copying the record to diskettes or the local hard disk. The *Operations* entry is selected from the menu bar in the *CDOCS Main Menu* (Figure 2-5). The system displays a pull-down menu (Figure 2-6) that contains an option to copy regulatory program or OITS records. When the *Copy* entry in the *Operations* pull-down menu is selected, the `_intern_MenuOperationsCopy` event is recognized by the main event loop. This causes the `loadFileCopy` function to be executed, and the *Copy a Regulatory Program Record* screen (Figure 2-7) is displayed.

Copy a Regulatory Program Record—Functionality

The *Copy a Regulatory Program Record* screen is initially displayed for the default document type, and the review plan numbers and titles for all documents of that type are displayed in the list view. When a different document type is selected from the *Document Type* pull-down list, the *Copy a Regulatory Program Record* screen is displayed again for the selected document type. The list view portion of the *Copy a Regulatory Program Record* screen contains the Review Plan number and Title of all records of the selected type. The cursor is positioned in the list view and the line that contains the desired record is selected. The full path and file name of the destination file are entered in the *Filename for Copy* entry field.

If the full path and file name of the existing file is not known, the CDOCS file management system may be requested to find the desired file by selecting the *File Chooser* push-button next to the *Filename for Copy* entry field. This causes the `_intern_FileCopyChooserButton` event to be recognized by the main event loop, and the `filecopyfilechooser` function is executed. This causes a list of files in the



CDOCS—CDOCS main menu

Figure 2-5. CDOCS—CDOCS main menu

current directory to be displayed on the *File Chooser* input screen (Figure 2-8), and the cursor may be positioned on a filename to select the desired file. If the desired file is in a different directory than the one being displayed by the file chooser, the cursor may be used to select a different directory. This causes the names of the files in the new directory to be displayed so that the desired file may be selected. When the desired file name has been selected, the *OK* push-button on the *File Chooser* screen is selected. This causes the path and file name of the selected file to be copied to the *Filename for Copy* entry field, and the previous screen is displayed. The currently selected path and file name are automatically passed back to the previous screen. The *Cancel* push-button is selected to exit from the *File Chooser* screen.

When all of the information is correct, the *Copy* push-button at the bottom of the screen is selected. The record is copied to the specified destination file. The *Close* push-button at the bottom of the screen is selected to exit.

Copy a Regulatory Program Record—Document Type Pull-down List Processing

If the *Document Type* pull-down list indicator is selected, the *_intern_FileCopyDocType* event is recognized by the main event loop. This causes the *loadDocs* function to execute and a pull-down list of document types is displayed.

Copy a Regulatory Program Record—File Chooser Push-button Processing

If the *File Chooser* push-button next to the *Filename for Copy* entry field is selected, the *_intern_FileCopyChooserButton* event is recognized by the main event loop, and the *filecopyfilechooser* function is executed. This opens the file chooser dialog and stores the selected file name when the file chooser dialog terminates.

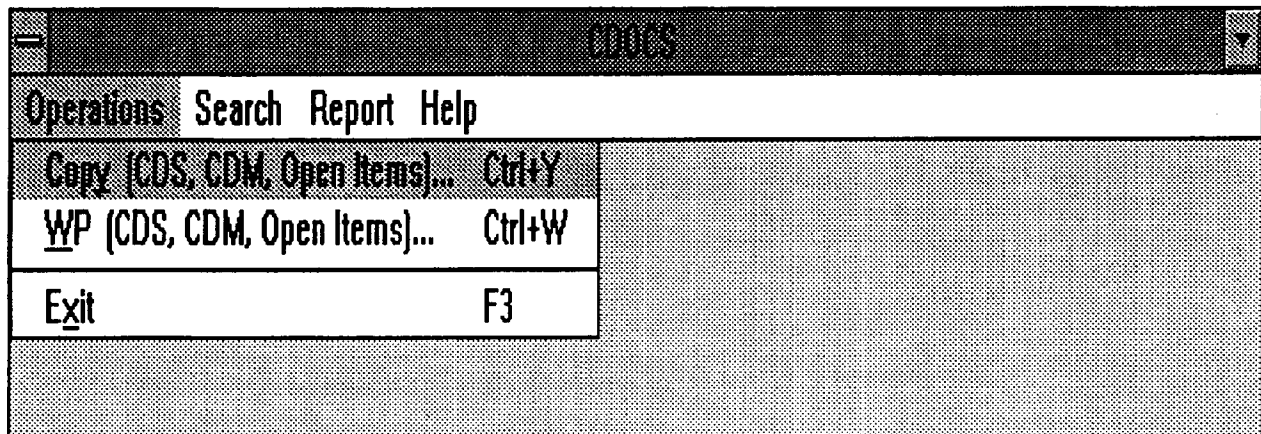


Figure 2-6. CDOCS—Operations pull-down menu

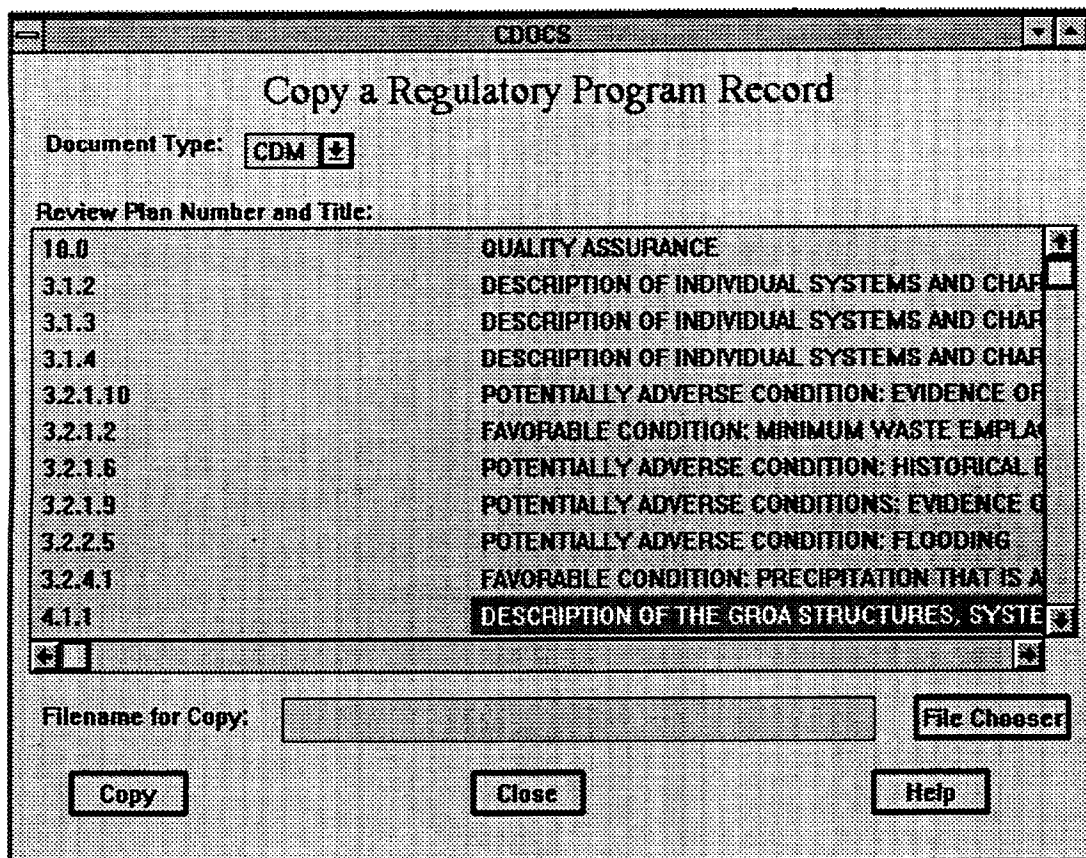


Figure 2-7. Copy a regulatory program record

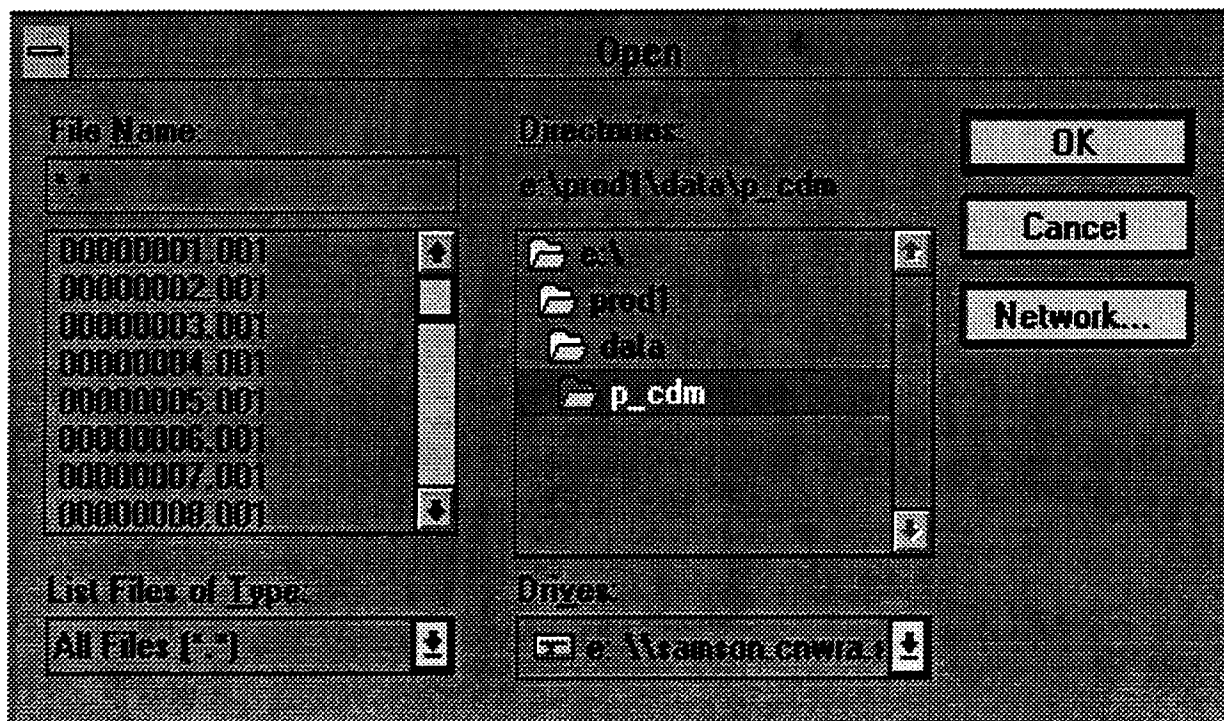


Figure 2-8. File chooser input screen

Copy a Regulatory Program Record—Copy Push-button Processing

If the *Copy* push-button is selected, the *_intern_FileCopyOkayButton* event is recognized by the main event loop, and the *filecopy* function is executed. The *filecopy* function gets the document type and file name. If the file name is null, an error message is displayed. The row count of the selected document name in the list view is retrieved. If this row count is null, an error message is displayed. The review plan number is retrieved and the *vrPCCopyWP* function is executed to copy the desired file from the server to the client location. If no error codes are returned from the *vrPCCopyWP* function, a message is displayed indicating a successful completion of the copy operation and the function returns.

Copy a Regulatory Program Record—Close Push-button Processing

If the *Close* push-button is selected, the *_intern_FileCopyChooserButton* event is recognized by the main event loop, and the *closeFileCopy* function is executed. The *closeFileCopy* function closes the file copy dialog and returns.

2.2.7.2 Operations—Editing Regulatory Program and Open Item Records with WordPerfect

The *WordPerfect* entry in the *Operations* pull-down menu may be used to access copies of information in the database using WordPerfect Version 5.1. The general process for editing a regulatory program or OITS record using WordPerfect is to (i) select the record to be edited from a list of available records, and (ii) start the WordPerfect software so that the selected record can be edited. The records edited through WordPerfect may not be stored directly in CDOCS, but may be saved on a diskette or on the local hard disk.

Edit a Regulatory Program or OITS Record Using WordPerfect—Functionality

The *Operations* entry is selected from the menu bar of the *CDOCS Main Menu*, and the system displays a pull-down menu that contains an option for editing regulatory program or OITS records using WordPerfect. The *WordPerfect* option in the *Operations* pull-down menu is selected, and the *Edit a Regulatory Program Record Using WordPerfect* screen is displayed (Figure 2-9). The *Edit a Regulatory Program Record Using WordPerfect* screen is initially displayed for the default document type, and the review plan numbers and titles for all documents of that type are displayed in the list view. When a different document type is selected from the *Document Type* pull-down list, the *Edit a Regulatory Program Record Using WordPerfect* screen is displayed again for the selected document type. The list view portion of the *Edit a Regulatory Program Record Using WordPerfect* screen contains the Review Plan number and Title of all records of the selected type. The cursor is positioned in the list view and the line that contains the desired record is selected. When the selection is correct, the *Edit* push-button at the bottom of the screen is selected. This causes the system to start WordPerfect using the selected record. The *Close* push-button at the bottom of the screen is selected to exit.

Edit a Regulatory Program or OITS Record Using WordPerfect—Document Type Pull-down List Processing

If the *Document Type* pull-down list indicator is selected, the *_intern_FileCopyDocType* event is recognized by the main event loop. This causes the *loadDocs* function to be executed and a pull-down list of document types is displayed.

Edit a Regulatory Program or OITS Record Using WordPerfect—Edit Push-button Processing

If the *Edit* push-button is selected, the *_intern_FileWPOkayButton* event is recognized by the main event loop, and the *fileWP* function is executed. This function is platform dependent and the code generation is conditioned by *ifdef* statements. In general, the function gets the document type and file name. If the file name is null, an error message is displayed. The row count of the selected document name in the list view is retrieved. If this row count is null, an error message is displayed. The review plan number is retrieved and the WordPerfect application is executed to edit the desired file.

Edit a Regulatory Program or OITS Record Using WordPerfect—Close Push-button Processing

If the *Close* push-button is selected, the *_intern_FileWPCancelButton* event is recognized by the main event loop, and the *closeFileWP* function is executed. The function closes the WordPerfect edit dialog and returns.

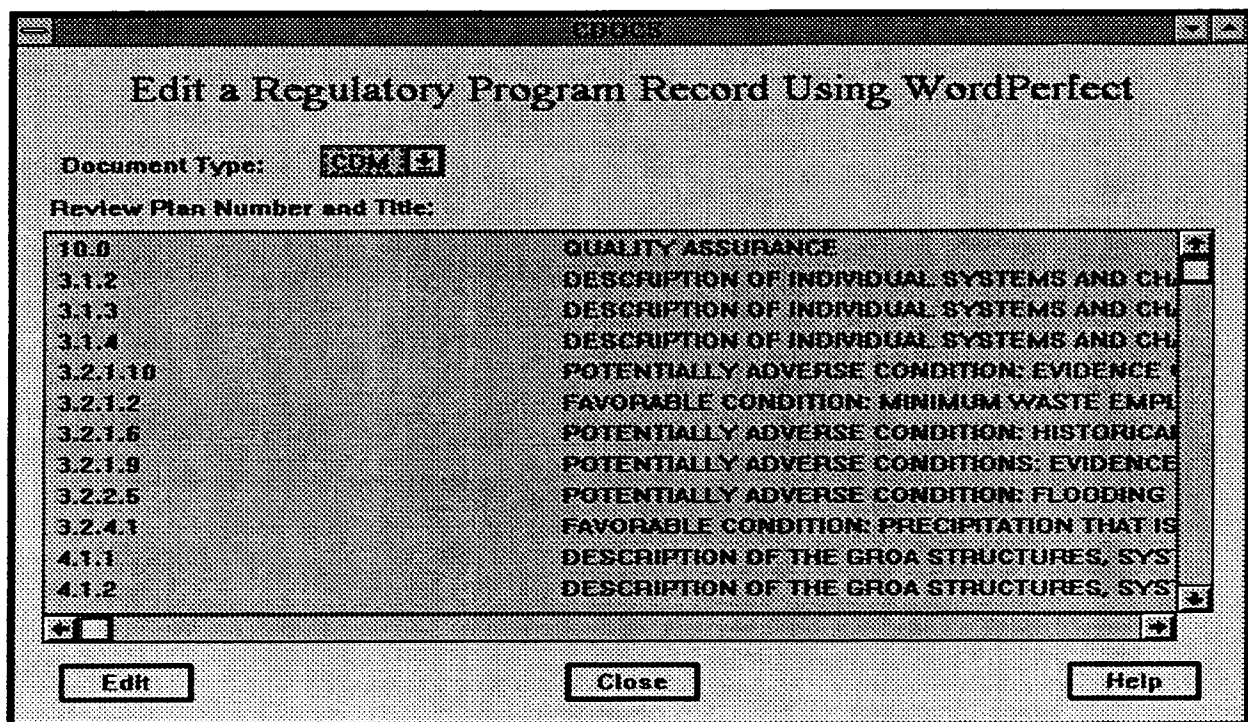


Figure 2-9. Edit a regulatory program record using WordPerfect screen

2.2.7.3 Operations Pull-down Menu— Change Password

When the *Change Password* entry is selected from the *Operations* pull-down menu, the `_intern_MenuOperationsChangePW` event is recognized by the main event loop. This causes the `loadchangePWdialog` function to be executed, and the *Change Password* screen is displayed.

Change Password—Functionality

When the *Change Password* screen appears, the new password is entered and then re-entered for verification. When the *Change* push-button is selected, if the passwords are the same and are valid, the password is changed and the dialog is closed. If the password is missing, invalid, or if the two entered passwords are not the same, an error message is displayed. If the *Close* push-button is selected, the dialog is closed and the password is not changed.

Change Password—Menu Option Processing

When the *Change* push-button is selected, the *_intern_MenuOperationsChangePW* event is recognized by the main event loop. This causes the *loadchangePWdialog* function to be executed. The function initializes the *Help* button and opens the change password dialog. The *SetupChangePWField* function is executed to set up the text filters to use when changing passwords.

Change Password—Change Push-button Processing

If the *Change* push-button is selected, the *_intern_ChangePWButton* event is recognized by the main event loop. This causes the *changePassword* function to be executed. The passwords are validated for length. If the password is null, an error message is displayed. If the password is less than 4 characters or greater than 16 characters in length, an error message is displayed. The two passwords are compared. If they are not identical, an error message is displayed. If no errors are detected, the *vRPCChangePassword* function is executed to change the password. If the password update is successful, a confirmation message is displayed and the dialog is closed.

Change Password—Close Push-button Processing

If the *Change* push-button is selected, the *_intern_PWCloseButton* event is recognized by the main event loop. This causes the *vdialogClose(changePWdialog)* function to be executed, and the dialog is closed.

2.2.7.4 Operations Pull-down Menu— Scan and OCR

Selected users have scan capable workstations. The *Operations* pull-down menu for these users contains an entry for *Scan and OCR*.

Scan and OCR—Functionality

When the *Scan and OCR* entry is selected from the *Operations* pull-down menu, the *_intern_MenuOperationsScan* event is recognized by the main event loop. This causes the *onDemandScan* function to be executed, which starts the M-PRO series scanning software for on-demand scanning. This function can only be run on a PC machine that has the Calera M-PRO Series software available. A SUN Workstation would execute SCANWORKS Software.

Scan and OCR—Processing

When the *onDemandScan* function starts, it checks to see if the scanning software is already running. If either SCANWORKS or M-PRO is already running, the function restores the window and brings it to the front. If the scanning software is not running, the function initiates SCANWORKS or M-PRO, if the software is available.

2.2.7.5 Operations Pull-down Menu—Preferences

If the *Preferences* entry is selected from the *Operations* pull-down menu, the *_intern_MenuOperationsPrefs* event is recognized by the main event loop. This causes the *loadPrefsdialog* function to be executed, and the *CDOCS Preferences* screen is displayed

Preference Screen—Functionality

When the *Preferences* entry is selected from the *Operations* pull-down menu, the preferences screen is displayed with the current preference settings for the client platform. The current preferences are stored in the *main.vr* resource file in the CDOCS home directory on the client workstation. When the information has been updated on the screen, the *Save* push-button at the bottom of the screen is selected to save the preferences and update the *main.vr* resource file in the CDOCS home directory on the client workstation. The *Close* push-button at the bottom of the screen is selected to exit from the *Preferences* screen. If the *Close* push-button is selected to exit before the *Save* push-button is selected, the updated preferences are not saved.

Preference Screen—Save Push-button Processing

When the *Save* push-button is selected from the *Preferences* screen, the *_intern_PrefsSaveButton* event is recognized by the main event loop. This causes the *savePrefs* function to be executed. The entered preference values are retrieved from the dialog. A pointer is established to the preferences resource file, and the preference values are stored.

Preference Screen—Close Push-button Processing

When the *Close* push-button is selected from the *Preferences* screen, the *_intern_PrefsCancelButton* event is recognized, and the *Preferences* dialog is closed.

2.2.7.6 Operations—Exiting from the Consolidated Document System

The user may exit from the CDOCS system by selecting the *Exit* entry from the *Operations* pull-down menu.

Operations Pull-down Menu—Exit Functionality

When the *Operations* entry is selected from the menu bar of the *CDOCS Main Menu*, the system displays a pull-down menu that includes an option for exiting CDOCS. If the *Exit* entry is selected from the *Operations* pull-down menu, the *_intern_MenuOperationsExit* event is recognized by the main event loop. This causes the *veventStopProcessing* function to be executed. The system stops the main event loop, closes the users security permissions and authorities, and terminates the CDOCS session.

2.2.8 Main Menu—Search Entry

CDOCS provides powerful and easy-to-use facilities for identifying and viewing records. The TOPIC search and retrieval software is used to find CDOCS records by searching for specific words or phrases in: (i) the text of the document, and/or (ii) document header fields.

2.2.8.1 Accessing the Search Facilities

TOPIC full text search and retrieval facilities are accessed by selecting the *Search* entry from the menu bar of the *CDOCS Main Menu*. During the initialization of the main program, the *SearchCallBack* function is executed to set a callback for the *Search* entry in the *CDOCS Main Menu*. When the *Search* entry is selected from the menu bar of the *CDOCS Main Menu*, the callback for the *MenuSearch* item is activated and the *Search* function is executed.

Search Menu Entry—Functionality

When the *Search* entry is selected from the menu bar of the *CDOCS Main Menu*, the *search* function is executed. The purpose of this function is (i) to initialize the search selections and TOPIC user preferences, and (ii) to open the *CDOCS Search* selection dialog. A variable, *scratchSearch*, is set to *TRUE* to indicate that an unnamed TOPIC preferences file is to be created. This variable is set to *FALSE* if the *Save* push-button is selected from the *CDOCS Search* selection screen. Three major functions are called by *Search*:

- *BuildSearchQueries*—to initialize the selections on the *CDOCS Search* selection screen
- *GetSearchSources*—to store the default selections for the TOPIC preferences
- *openDialog (SearchDialog)*—to display the *CDOCS Search* selection screen (Figure 2-10)

2.2.8.2 CDOCS Search Screen

The *CDOCS Search* selection screen is displayed for either the NRC or CNWRA, depending on the user type. The desired CDOCS document sets to be searched are selected from the Selection List and the *OK* push-button at the bottom of the screen is clicked. The system initializes the user preferences based on the user selections and user class and starts the TOPIC software. The system starts the TOPIC search and retrieval facilities and displays a TOPIC Query Entry screen for simple queries.

CDOCS Search Screen—Cancel Push-button Processing

If the *Cancel* push-button is selected from the *CDOCS Search* selection screen, the *SearchProc* function calls the *SearchCancel* function to terminate the TOPIC search request.

CDOCS Search Screen—OK Push-button Processing

If a push-button other than the *Cancel* push-button is selected from the *CDOCS Search* selection screen, the *SearchProc* function calls the *SearchSelectProc* function to get a blank delimited list of selected items. If the list is null, an error message is displayed. The name of the TOPIC preference file is obtained for the saved query and the preferences are retrieved. The *BuildLaunchCmd* function is called to create the TOPIC preference file and the command script that starts the TOPIC application. If the *scratchSearch* variable is *TRUE*, the *BuildLaunchCmd* function calls the *WritePrefFile* function to generate and save the TOPIC preferences file. The *WritePrefFile* function calls the *SetTopicSources* and *SetTopicPrivilege* functions to process the requested selections from the *CDOCS Search* selection screen and generate TOPIC preferences and sources. The *starttopic* function is called to execute the script and

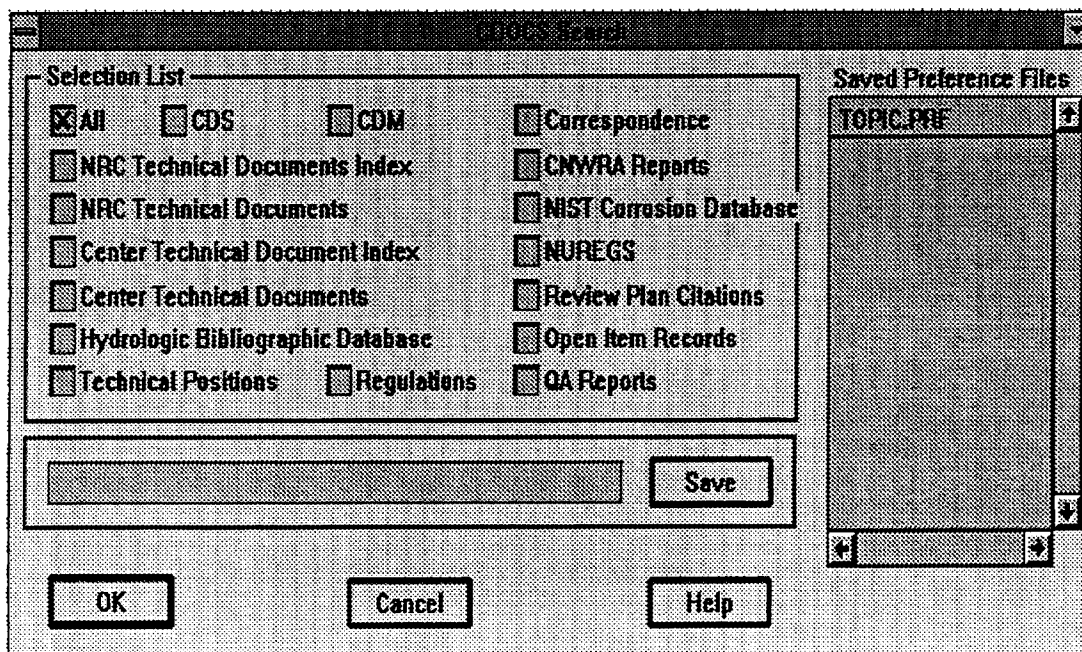


Figure 2-10. CDOCS—Search selection screen

start TOPIC. When the TOPIC application terminates, the *SearchProc* function calls the *closeSearch* function to terminate the search.

CDOCS Search Screen—Save Push-button Functionality

When the *Search* push-button is selected the TOPIC preferences are generated and saved, but the TOPIC application is not started. To start the TOPIC application the *OK* push-button must be selected.

If a push-button other than the *Cancel* push-button is selected from the *CDOCS Search* selection screen, the *SearchProc* function calls the *SearchSelectProc* function to get a blank delimited list of selected items. If the list is null, an error message is displayed. The name of the TOPIC preference file is obtained for the saved query is retrieved. If the *Save* push-button has been pressed, the *SaveSearchQuery* function is called to get the preference file name. The *SaveSearchQuery* function calls the *BuildPrefFileName* to obtain a file name for the TOPIC preferences file. Then it calls the *WritePrefFile* function to generate and save the TOPIC preferences file. The *WritePrefFile* function calls the *SetTopicSources* and *SetTopicPrivilege* functions to process the requested selections from the *CDOCS Search* selection screen and generate TOPIC preferences and sources.

2.2.9 Main Menu—Custodian Entry

CDOCS provides capabilities for submitting, updating, deleting, checking in, and checking out CDOCS records, and maintaining regulatory program and OITS records. These facilities are available to the database custodian through the *Custodian* entry in the menu bar of the *CDOCS Main Menu*.

2.2.9.1 Custodian Pull-Down Menu

When the *Custodian* entry is selected, a pull-down menu appears that contains entries for each of the maintenance functions available to the database custodian. Figure 2-11 illustrates the *Custodian* pull-down menu for NRC database custodians. Figure 2-12 illustrates the *Custodian* pull-down menu for CNWRA database custodians. The *Custodian* pull-down menu permits access to the following maintenance functions:

Submit Record—Use CDOCS maintenance facilities to add new CDOCS records

Update Record—Use CDOCS maintenance facilities to update existing CDOCS records

Delete Record—Use CDOCS maintenance facilities to delete existing CDOCS records

Process a Regulatory Record —Access facilities for maintenance of regulatory program and OITS records

In addition to these capabilities, the CNWRA *Custodian* pull-down menu (Figure 2-12) contains the following functions that pertain to circulation control at the CNWRA library:

Checkout Document—Use CDOCS maintenance facilities to check a hard-copy document out of the CNWRA library

Checkin Document—Use CDOCS maintenance facilities to check a hard-copy document into the CNWRA library

Custodian Pull-down Menu—Functionality

When the *Custodian* entry is selected from the *CDOCS Main Menu*, the system displays a pull-down menu containing options to *Submit Record*, *Update Record*, *Delete Record*, and *Process a Regulatory Record* (Figure 2-11 or 2-12). The desired function is selected from the *Custodian* pull-down menu. This activates a callback or signals an event that is recognized by the main event loop function, *veventProcess*, and the system executes the function associated with the selected event.

2.2.9.2 Submitting New CDOCS Records

When *Submit Record* is selected from the *Custodian* pull-down menu, the *_intern_CustodianSubmit* event is recognized by the main event loop. This causes the *loadSubmit* function to be executed with a parameter of “Submit”. The CDOCS document submission dialog is actually used for submission, updates and deletion confirmations. Therefore, the *loadSubmit* function modifies the dialog before the screen is displayed.

CDOCS Submit Screen—Initialization Processing

The *loadSubmit* function is called with a parameter that indicates the type of screen desired (i.e., submit, update, or delete). It changes the visible labels on the submit button to “Submit”, “*Update*”, or “*Delete*”, calls the *SetFieldEntryButtons* function, and opens the dialog to display the screen. The

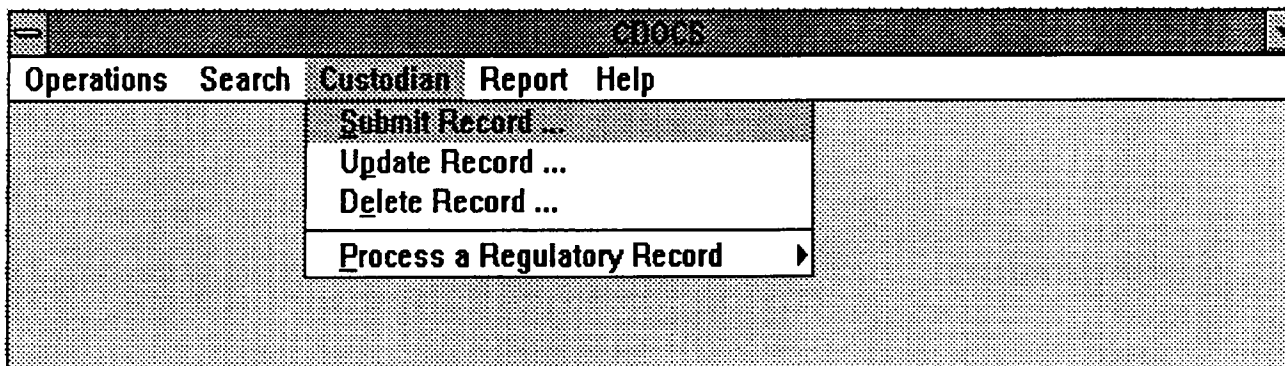


Figure 2-11. CDOCS—NRC Custodian pull-down menu

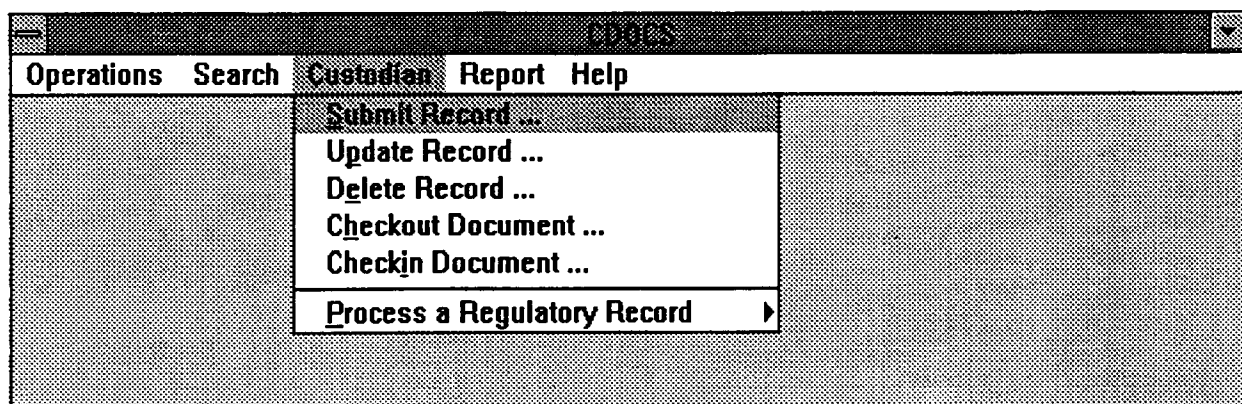


Figure 2-12. CDOCS—CNWRA Custodian pull-down menu

SetFieldEntryButtons function calls the *checkForLoadOrSubmit* function which checks the passed parameter and modifies the dialog as needed. If the parameter is “Submit”, the *ReconnectClear* function is called to enable the *Clear* push-button. If the parameter is “Update”, or “Delete”, the *DisconnectClear* function is called to disable the *Clear* push-button. The screen title is adjusted to reflect the type of processing.

CDOCS Submit Screen—Functionality

The *Submit a CDOCS Record* entry screen (Figure 2-13) appears with the NRC Technical Document Set in the *Document Set* pull-down as the default. CDOCS contains a number of document sets, such as NRC Technical Documents, CNWRA Technical Documents, CNWRA Reports, and NUREGS. Fields that are not applicable to the currently selected document set, such as *Addressee(s)* and *Duration*, are grayed out and the field cannot be accessed. The *Submit a CDOCS Record* screen includes both mandatory and optional fields. Optional fields are indicated by an asterisk (*) next to the field name on the left side of the screen.

The *Document Number* field is used to display the document number assigned by the system after the *Submit* push-button is selected. Therefore, no information is entered in the *Document Number* field.

The user enters data in the fields. At the right side of several of the fields, such as *Author(s)*, *Title*, and *Publication*, there is a small square push-button with three dots. This push-button is used to expand the entry field when there is more than one line of information to input. When this button is selected, the *tdocsContainer* callback is activated and the *expandView* function is called to display a new screen with a scrollable entry field (Figure 2-14). Several lines of information may be entered through this screen. When the information has been entered, the *OK* push-button at the bottom of the screen is selected, and the information that was entered is copied to the corresponding data field.

The *Directory where files reside* field appears at the bottom of the screen. This field is initialized to “None” indicating a header only document. If the document includes text, word processing, and/or image files, the directory where the files are stored must be specified in this field. The directory name may be entered or the drop-down indicator at the right of this field may be used to select the directory where the text, word processing, and image files for the document that is being submitted are located (Figure 2-15). If the directory does not appear in the drop-down list, the path to the directory must be keyed. The *Submit* push-button at the bottom of the screen is selected to submit the record to CDOCS.

CDOCS Submit Screen—Document Set drop-down Functionality

When the *Document Set* drop-down indicator is selected the screen is refreshed. During this screen refresh process, the current selected entry in the *Document Set* drop-down list is examined and the screen format is adjusted to accommodate the document type.

CDOCS Submit Screen—Document Set drop-down Processing

When the *Document Set* drop-down indicator is selected, the *documentMenu* callback is activated and the *OptionProc* function is called. The *OptionProc* function calls the *buttonClear* function which resets the display of the document number and then calls the *SetFieldEntryButtons* function to configure

Document Set: NRC Technical Documents
Document Number:

Document Header:

Author(s)	Sagar, B.	<input type="button" value="v"/> <input type="button" value="v"/> <input type="button" value="v"/> <input type="button" value="v"/>
Addressee(s)		
Title	Flow Modeling in Heterogeneous Media in the Context of Geologic Nuclear Wa	
Publication	CNWRA	
Document Date	01 Oct 1995	<input type="button" value="v"/> <input type="button" value="v"/> <input type="button" value="v"/> <input type="button" value="v"/>
Code(s)	818.2	
Number(s) *		
Duration		
Note *		<input type="button" value="v"/>
Sharable	<input checked="" type="checkbox"/>	*Optional Values

Directory where files reside: C:\SUBMIT\A983

Figure 2-13. CDOCS Submit—NRC Technical Document entry screen

the screen and enable or disable buttons. This function also calls functions that perform RPC calls to the server to get a list of valid document types and their specifications. From this list, the *Document Type* drop-down list is refreshed. The *SetFieldEntryButtons* function then calls the *EnableFieldValues* function to configure the visible fields on the screen. During the execution of *EnableFieldValues* the selected document type is examined and if it is “QA” the screen is configured for a QA record.

CDOCS Submit Screen—Submit Push-button Functionality

When the *Submit* push-button is selected, the transaction is validated. If errors are found, appropriate error messages are displayed. If no errors are found, the submit transaction is accepted, an advisory message is displayed, and a document number is assigned to the record. The document number is displayed in the *Document Number* field at the top of the screen.

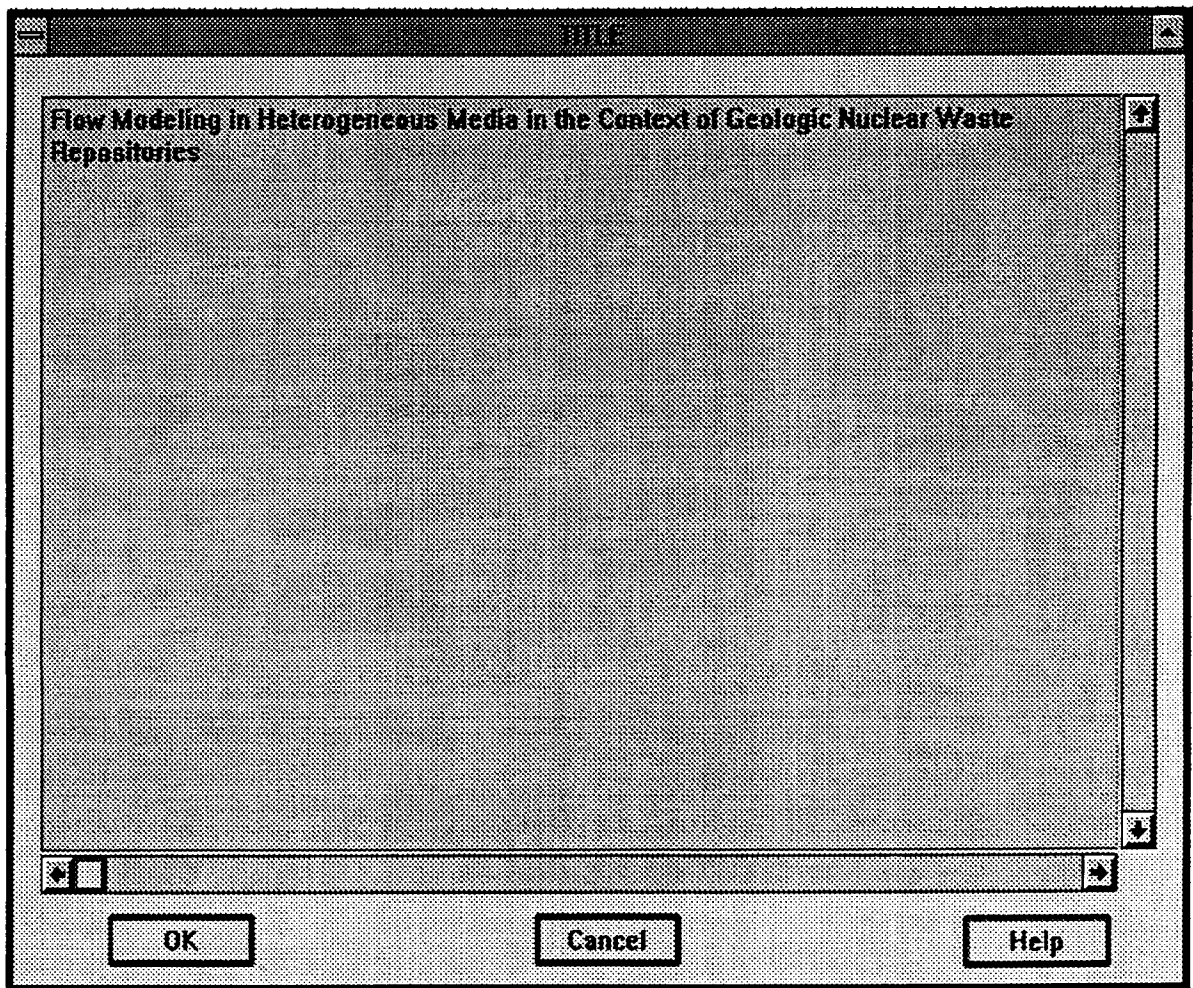


Figure 2-14. CDOCS Submit—extended entry-field screen

CDOCS Submit Screen—Submit Push-button Processing

When the *Submit* push-button is selected, the *submitButton* callback is activated and the *buttonSubmit* function is called. This function is used by submit, update, and delete transactions. Therefore the code distinguishes the transaction type by examining the screen title. A character string is initialized and the document number is copied to this string for update transactions. The *writeTDOCSinfo* function is then called to format the submit or update transaction and pass it to the server for processing.

The *writeTDOCSinfo* function assembles the information for the new record from the dialog. The directory path is obtained from the pull-down list of directories and stored after removing extraneous white space. A string named *finalstr* is created and field data for the new record is appended to it. As each field is accessed, the *ValidateSubmitField* function is called to validate the format of the information in that

The screenshot shows a window titled "CDOCS Submit". At the top, there are two fields: "Document Set:" with a dropdown menu showing "NRC Technical Documents" and a small icon, and "Document Number:" with an empty text box. Below this is a section labeled "Document Header:" containing several fields: "Author(s)" with "Sager, B.", "Addressee(s)" (empty), "Title" with "Flow Modeling in Heterogeneous Media in the Context of Geologic Nuclear Wa", "Publication" with "CNWRA", "Document Date" with "01 Oct 1995", "Code(s)" with "810.2", "Number(s) *" (empty), "Duration" (empty), and "Note *" (empty). To the right of these fields are three small icons. Below the "Document Header:" section is a "Sharable" checkbox which is checked, and a label "*Optional Values". At the bottom, there is a "Directory where files reside" field with a dropdown menu showing "C:\SUBMIT\A983". Below this field is a "Submit" button. To the right of the "Submit" button is a "Help" button. The background of the window is a light gray with a subtle grid pattern.

Figure 2-15. CDOCS Submit—directory where files reside

field. The first field validation error terminates the *writeTDOCSinfo* function. If no field errors are found, the *writeTDOCSinfo* function calls the *submitRecord* function to transfer the record submission request to the server and obtain the new document number, and passes the finalstr variable that contains the record field information.

The *submitRecord* function calls the *RPCSubmitRecord* function to transfer the submission request to the server. Following the execution of *RPCSubmitRecord*, the *checkForSQLErrors* function is called to check for and display any SQL errors. If errors are detected, the *DelSubmitFiles* function is called to remove the submitted files for the rejected record from the submit directory. If the transfer is successful the returned document number is displayed in the document number field of the *CDOCS Submit* screen and a confirmation message is displayed.

CDOCS Submit Screen—Reset Push-button Functionality

After the submit transaction has been processed and the advisory message has been cleared, the screen is locked, but the information entered in the previous submit transaction remains in the input fields. This action permits efficient submission of sets of related documents by altering only the information that has changed. The *Reset* push-button at the bottom of the screen may be selected to enable the entry screen and permit entry of the information for the next record.

CDOCS Submit Screen—Reset Push-button Processing

The document submission screen is reset by selecting the *Reset* push-button at the bottom of the screen. When the *Reset* push-button is selected, the *clearButton* callback is activated and the *ResetSubmit* function is called to clear the document number, disable the *Reset* button, and enable the *Submit* push-button.

CDOCS Submit Screen—Close Push-button Functionality

The document submission process is terminated by selecting the *Close* push-button at the bottom of the screen. When the *Close* push-button is selected, the *tdocsCloseButton* callback is activated and the *buttonClear* function is called. All of the extended entry fields are cleared and the document number is reset and displayed. The *SetFieldEntryButtons* function is called with a parameter of *Submit* to restore the label on the *Submit* push-button. The list of directories for submitted files is reset to the top entry and the *ResetSubmit* function is called to clear the document number, disable the *Reset* button, and enable the *Submit* push-button.

2.2.9.3 Updating Records from the Consolidated Document Management System

CDOCS document updates are always handled as a complete replacement of the existing document header record and, optionally, a replacement of the document text and image files. If new text and image files are submitted, the existing files are deleted, and the new files are added to the CDOCS database. Updating a record requires that the user specify the document set and document number that was assigned automatically when the record was submitted to CDOCS. These two data elements (document set and document number) uniquely identify the record to be updated.

During system initialization in the main program, *TDOCSStart* function is called to setup callbacks. The *TDOCSStart* function calls the *LoadTDOCSUpdate* function to load the dialog and set the callbacks for the *OK*, *Close*, and *Help* push-buttons. The callbacks for the *OK* and *Close* push-buttons are both set to call the *TDOCSUpdateProc* function. When executed, the *TDOCSUpdateProc* function checks the type of request internally and processes the dialog item as required.

CDOCS Update Entry Screen—Functionality

When the *Update Record* entry is selected from the *Custodian* pull-down menu, the *_intern_CustodianUpdate* event is recognized by the main event loop. This causes the *TDOCSUpdate* function to be executed, and the *CDOCS Update* screen (Figure 2-16) is displayed to permit entry of the *Document Set* and *Document Number* for the record to be updated. The *CDOCS Update* screen appears with the *Document Set* drop-down list initialized to NRC Technical Documents by default. To select a different document set, (e.g., Low-Level Waste), the *Document Set* must be changed by selecting the

Document Set drop-down indicator and then selecting the desired document set. The number of the document to be updated is entered in the *Document Number* entry field.

CDOCS Update Screen Processing

The *TDOCSUpdate* function calls *BuildDocSet* to generate the entries for the *Document Set* drop-down list on the *TDOCSUpdateDialog* dialog, and then opens the dialog. This causes the *CDOCS Update* screen to be displayed for entry of the *Document Set* and *Document Number*.

CDOCS Update Screen—Document Set Drop-down List Processing

The *Document Set* drop-down list is initialized to NRC Technical Documents by default. When the *Document Set* drop-down indicator is selected, the drop-down list of available document sets appears, and the user may select a new entry. The selected entry is retrieved during the execution of *TDOCSUpdateProc* when the *OK* push-button is selected. The retrieved document set is stored for use by the *TDOCSUpdateGetRec* function.

CDOCS Update Screen—OK Push-button Processing

When the *OK* push-button is selected, the *Update OK Tag* callback is activated and the *TDOCSUpdateProc* function is called. The *TDOCSUpdateProc* function recognizes the tag for the *OK* push-button at the bottom of the screen, strips the white space from the document number, retrieves the document set from the dialog, and calls the *TDOCSUpdateGetRec* function to retrieve the record and display the *CDOCS Update* entry screen (Figure 2-17).

The *TDOCSUpdateGetRec* function calls the *getTDOCSinfo* function to retrieve the record, and *getTDOCSinfo* calls *RPCGetRec*, passing the document set and document number. If the record retrieval is successful the fields returned from the server are loaded into the dialog, the *Sharable* flag is set and a success code is returned. If the record is retrieved successfully, the *loadSubmit* function is called to display the *CDOCS Update* screen and the *displayDocumentNo* function is called to display the document number. If the record retrieval is not successful, an error message is displayed.

CDOCS Update Screen—Close Push-button Processing

The *CDOCS Update* entry screen is closed and the CDOCS update processing is terminated by selecting the *Close* push-button at the bottom of the screen. When the *Close* push-button is selected, the *Update Cancel Tag* callback is activated and the *TDOCSUpdateProc* function is called. The *TDOCSUpdateProc* function recognizes the tag for the *Close* push-button, sets the document number to null and closes the dialog.

2.2.9.4 The CDOCS Update Entry Screen

The *CDOCS Update* entry screen uses the same dialog as the *CDOCS Submit* screen but certain fields and controls are modified. The *Document Number* field displays the number entered in the *CDOCS Update* screen. This number cannot be changed. The document header information is entered in the appropriate fields of the *CDOCS Update* entry screen (Figure 2-17). A shared document cannot be updated as unshared. If the *Sharable* button is grayed in the update screen, then the document is shared. To update

The image shows a graphical user interface window titled "CDOCS Update". Inside the window, there are two labeled input fields. The first is "Document Set" with a text box containing "NRC Technical Documents" and a small downward-pointing arrow icon to its right. The second is "Document Number" with a text box containing "N199601020001". Below these fields, there are three rectangular buttons arranged horizontally: "OK", "Close", and "Help". The window has a standard title bar with a minimize button on the left and a maximize button on the right.

Figure 2-16. CDOCS Update screen

and unshare a document, the document must be resubmitted as unshared. The effect of the shareable flag during a record update is defined in Table 2-4. If new ASCII text, word processing, or image files are to be submitted with the update, the path to the directory must be selected from the *Directory where files reside* drop-down menu, or keyed into the field. If no files are being submitted, "None" is entered in the *Directory where files reside* field. If no files are submitted, only the document header is updated. When all of the updated information has been entered, the *Update* push-button at the bottom left of the screen is selected to update the record. When the update has been successfully completed, a confirmation screen is displayed (Figure 2-18). The *Close* push-button is selected to exit the record update screen.

Table 2-4. Effect of sharable flag during record update

Prior Sharing Status	Updated Sharing Status	Local Database	Remote Database
Shared	Unshared	A shared document cannot be updated as unshared. The unshared update must be submitted as a new document	
Unshared	Shared	Record is updated	Record is submitted
Shared	Shared	Record is updated.	Record is updated.
Unshared	Unshared	Record is updated.	No action.

CDOCS Update Entry Screen—Update Push-button Functionality

When the *Update* push-button is selected, the transaction is validated. If errors are found, appropriate error messages are displayed. If no errors are found, the update transaction is accepted, an advisory message is displayed, and a document number is assigned to the record. The document number is displayed in the *Document Number* field at the top of the screen.

CDOCS Update Entry Screen—Update Push-button Processing

When the *Update* push-button is selected, the *submitButton* callback is activated and the *buttonSubmit* function is called. This function is used by submit, update, and delete transactions.

Therefore the code distinguishes the transaction type by examining the screen title. A character string is initialized and the document number is copied to this string for update transactions. The *writeTDOCSinfo* function is then called to format the update transaction and pass it to the server for processing.

The *writeTDOCSinfo* function retrieves the name of the directory where the data files are stored, if any, and strips the extraneous white space. The function creates a character string, *newstr*, retrieves the information from each field and appends it to the string. As each field is accessed, the *ValidateSubmitField* function is called to validate the format of the information in that field. The first field validation error terminates the *writeTDOCSinfo* function. If no field errors are found, the *writeTDOCSinfo* function calls the *updateRecord* function to transfer the record update request to the server. The *updateRecord* function calls the *GetSubmitInfo* function to copy the data files to the submit directory and then calls the *RPCUpdateRecord* function to pass the record to the server. Following the completion of the *RPCUpdateRecord* function, the *checkForSQLErrors* function is called to check for any errors. If errors are found, an error message is displayed and the *DelSubmitFiles* function is called to remove the submitted files for the rejected record from the submit directory. If the update is successful, the function displays an update confirmation record.

CDOCS Update

Document Set: Document Number:

Document Header:

Author(s)

Addressee(s)

Title

Publication

Document Date

Code(s)

Number(s) *

Duration

Note *

Sharable ☒ *Optional Values

Directory where files reside

Figure 2-17. CDOCS update entry screen

The *updateRecord* function calls the *RPCUpdateRecord* function to transfer the update request to the server. If the transfer is successful the returned document number is displayed in the *Document Number* field of the *CDOCS Update* screen and a confirmation message is displayed (Figure 2-18) .

CDOCS Update Entry Screen—Close Push-button Functionality

The document update process is terminated by selecting the *Close* push-button at the bottom of the screen. When the *Close* push-button is selected, the *tdocsCloseButton* callback is activated and the *buttonClear* function is called. This function calls the *resetDocumentNo* and *displayDocumentNo* functions to clear and display the document number, resets the pull down list of directories to the top entry, and calls the *ResetSubmit* function to clear the document number, disable the *Reset* button, and enable the *Submit* button.

2.2.9.5 Deleting Records from the Consolidated Document System

CDOCS includes facilities for deleting records from the document sets. Deletion of records requires that the user specify the document set and the document number that was assigned automatically when the record was submitted to CDOCS. These two data elements (document set and document number)

CDOCS Update

Document Set: NRC Technical Documents Document Number: N199601020001

Document Header:

Author(s) Sagar, B.

Addressee(s)

Title Heterogeneous Media in the Context of Geologic Nuclear Waste Repositories

Publication CNWRA

Document Date 01 OCT 1995

Code(s) 810.2

Number(s) *

Duration

Note *

Shareable ☒ *Optional Values

CDOCS Status

Document successfully updated.

OK

Directory where files reside C:\SUBMIT\A983

Update Close Help

Figure 2-18. CDOCS Update confirmation screen

uniquely identify the record to be deleted.

During system initialization in the main program, *TDOCSSStart* function is called to setup callbacks. The *TDOCSSStart* function calls the *LoadDelete* function to load the dialog and set the callbacks for the *OK*, *Close*, and *Help* push-buttons. The callbacks for the *OK* and *Close* push-buttons are both set to call the *DeleteProc* function. The *DeleteProc* function checks the processing request internally and processes the dialog item as required.

CDOCS Delete Document Screen Functionality

When the *Delete Record* entry is selected from the *Custodian* pull-down menu, the *_intern_CustodianDelete* event is recognized by the main event loop. This causes the *Delete* function to be executed, and the *CDOCS Delete* screen (Figure 2-19) is displayed to permit entry of the *Document Set* and *Document Number* for the record to be deleted.

The *CDOCS Delete* screen appears with the *Document Set* drop-down list initialized to NRC Technical Documents by default. To select a different document set, (e.g., Low-Level Waste), the

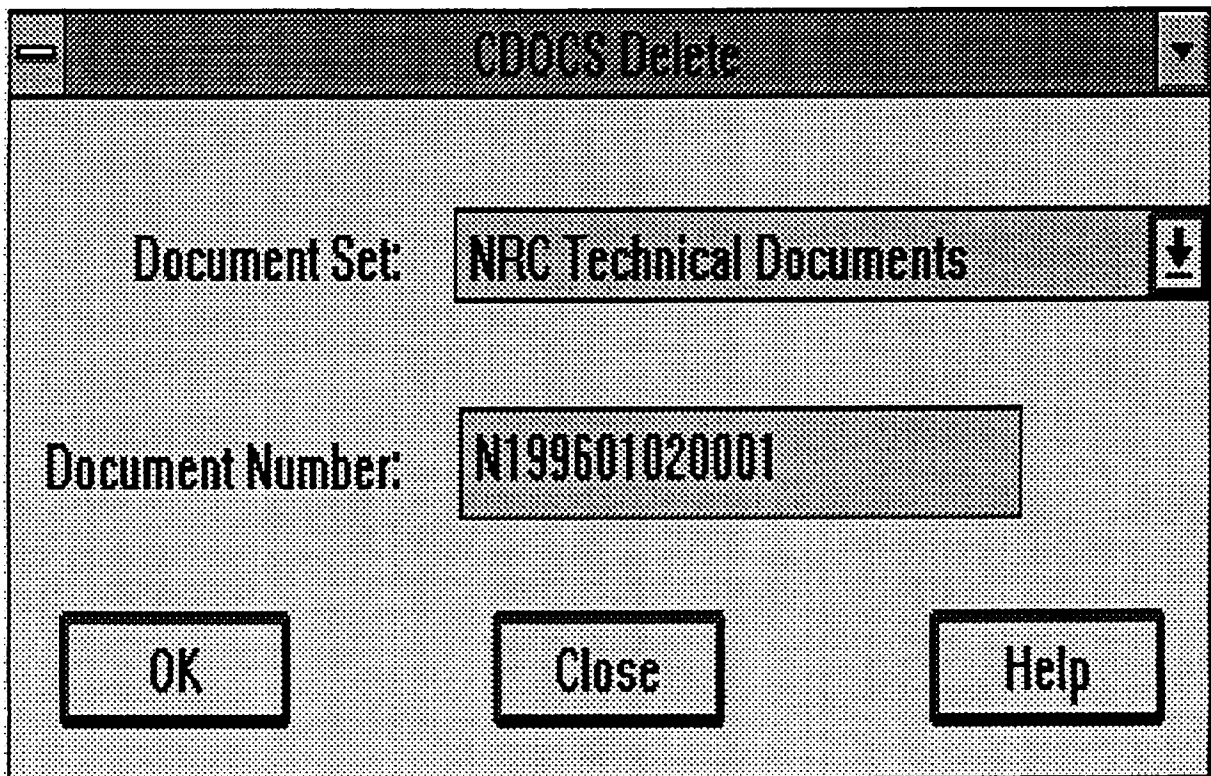


Figure 2-19. CDOCS delete screen

Document Set must be changed. This is done by selecting the *Document Set* drop-down indicator and then selecting the desired document set. The number of the document to be updated is entered in the *Document Number* entry field.

CDOCS Delete Document Screen—Processing

The *Delete* function calls the *BuildDocSet* function to generate the entries for the *Document Set* drop-down list on the *deleteDialog* dialog, and then opens the dialog. This causes the *CDOCS Delete* screen to be displayed for entry of the *Document Set* and *Document Number*.

CDOCS Delete Document Screen—Close Push-button Processing

The *Delete Document* entry screen is closed and the CDOCS delete processing is terminated by selecting the *Close* push-button at the bottom of the screen. When the *Close* push-button is selected, the *Delete Cancel Tag* callback is activated and the *DeleteProc* function is called. The *DeleteProc* function recognizes the tag for the *Close* push-button, and closes the dialog.

CDOCS Delete Document Screen—OK Push-button Processing

When the *OK* push-button is selected, the *Delete OK Tag* callback is activated and the *DeleteProc* function is called. The *DeleteProc* function recognizes the tag for the *OK* push-button at the bottom of the screen, strips the white space from the document number, retrieves the document set from the dialog, and calls the *getTDOCSinfo* function to retrieve the record. If the record retrieval is successful the *loadSubmit* function is called to display the *CDOCS Delete* confirmation screen (Figure 2-20), and the *displayDocumentNo* function is called to display the document number. If the record retrieval is not successful, an error message is displayed.

2.2.9.6 The CDOCS Delete Screen

The shareable flag is set as required by selecting the Shareable box. The result of deleting shared files is defined below:

- Shared record—deleted in both local and remote databases.
- Unshared record—deleted in local database; no action is taken at remote database.

The user verifies that the document displayed is the document to be deleted, and presses the *Delete* push-button at the bottom of the screen. When the document has been deleted, the system displays a *Document Deleted* response screen (Figure 2-21). If the document displayed is not the document to be deleted, press the *Close* push-button is selected to exit the screen without deleting the document.

CDOCS Delete Screen—Delete Push-button Processing

When the *Delete* push-button is selected, the *submitButton* callback is activated and the *buttonSubmit* function is called. This function is used by submit, update, and delete transactions. Therefore the code distinguishes the transaction type by examining the screen title. If the screen title contains the string “Delete”, the *deleteRecord* function is called to delete the record and then the *buttonClear* function is called to clear the buttons.

CDOCS Delete Screen—Close Push-button Processing

The document delete process is terminated by selecting the *Close* push-button at the bottom of the screen. When the *Close* push-button is selected, the *tdocsCloseButton* callback is activated and the *buttonClear* function is called. This function performs any required database modification and closes the dialog.

CDOCS Delete

Document Set: Document Number:

Document Header:

Author(s)

Addressee(s)

Title

Publication

Document Date

Code(s)

Number(s) *

Duration

Note *

Sharable ☒ *Optional Values

Directory where files reside

Figure 2-20. CDOCS Delete conformation screen

2.2.10 Custodian Functions For Regulatory Records

When the *Custodian* entry is selected, a pull-down menu appears that contains entries for each of the maintenance functions available to the database custodian. Figure 2-11 illustrated the *Custodian* pull-down menu for the NRC database custodian. Figure 2-12 illustrated the *Custodian* pull-down menu for CNWRA database custodians. When the *Process a Regulatory Record* entry is selected from the *Custodian* pull-down menu a cascading menu appears that contains options for maintaining regulatory records.(Figure 2-22). The *Process a Regulatory Record* cascading menu permits access to the following maintenance functions:

Define a Regulatory Record—Use CDOCS maintenance facilities to define new regulatory records.

Retire a Regulatory Record—Use CDOCS maintenance facilities to retire existing regulatory records.

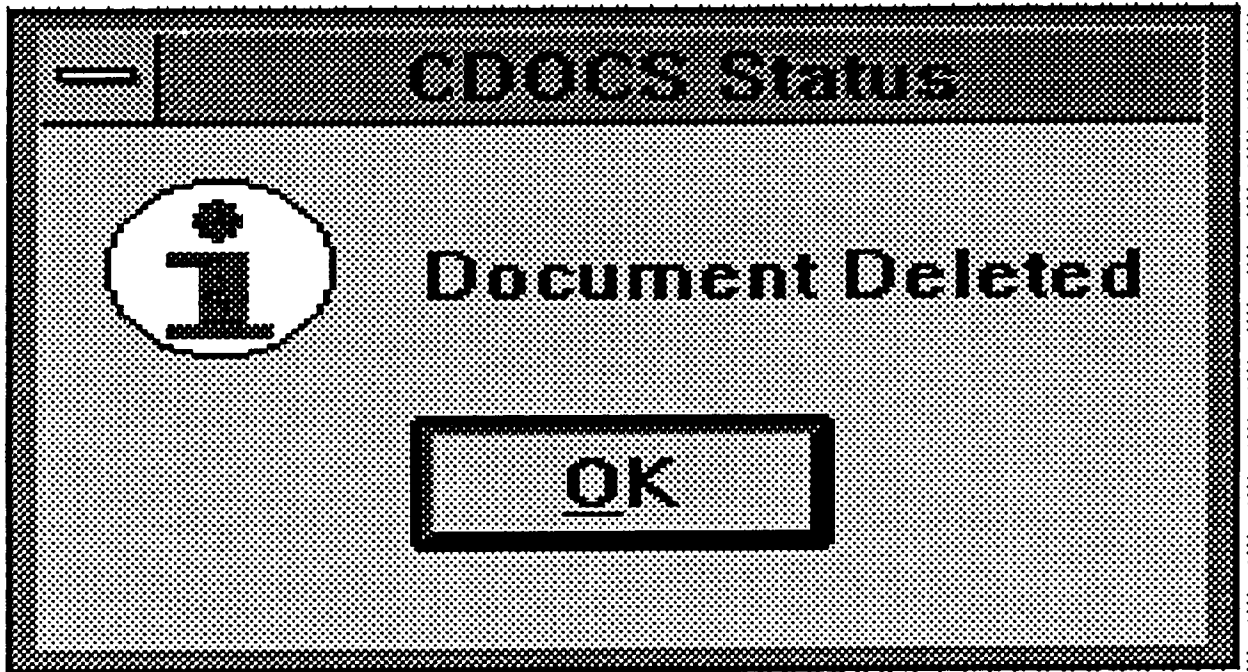


Figure 2-21. CDOCS Delete response screen

Format Check a Regulatory Record—Use CDOCS maintenance facilities to check the format of a regulatory record.

Enter a Regulatory Record—Use CDOCS maintenance facilities to enter a new version of a regulatory record.

2.2.10.1 Defining New Regulatory Records

Record header information, such as title and review plan number for regulatory program records, and OITS identifier and OITS topic for OITS records, must be preloaded in the database before textual data associated with the regulatory records may be checked in. Conceptually, the record header preloading process creates a place for the data, and the entry process actually stores the textual information. When a record is preloaded, the system reserves space in the relational database and in the full-text repository by formatting and storing placeholder records that contain the text. The record definition and header preloading process provides an entry screen through which the document type, review plan number, and title are entered.

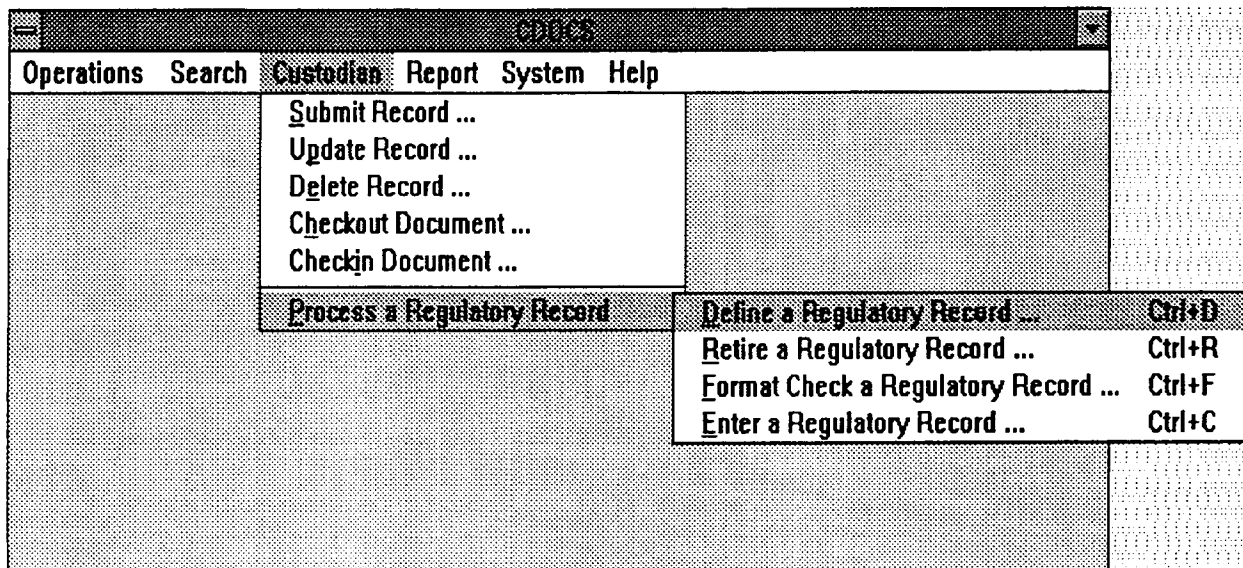


Figure 2-22. CDOCS process a regulatory record cascading menu

Define a Regulatory Record—Processing

When the *Define a Record* entry is selected from the *Process a Regulatory Record* cascading menu, the *_intern_DefineDefine* event is recognized by the main event loop. This causes the *loadDefine* function to be executed. The *loadDefine* function calls the *loadDocTypes* function, which then calls the *RPCOpenDocTypeCursor* and *RPCFetchDocTypeRow* functions to obtain a list of valid document types and load them into the *Document Type* drop-down list. The *loadDefine* function then calls the *openDialog* function to open the *Define* dialog and display the *Define a Regulatory Program Record* entry screen (Figure 2-23).

Define a Regulatory Record Entry Screen—Functionality

The *Document Type* field is initialized to a default value. If a different document type is required, the *Document Type* pull-down indicator is selected to display the pull-down list of document types (Figure 2-24), and the desired document type [Compliance Determination Strategy (CDS), Compliance Determination Method (CDM), Review Plan Preparatory Section (RPS), or OITS] is selected.

The review plan number for the new record is entered in the *Review Plan Number* entry field (Figure 2-25), and the title for the new record is entered in the *Title* entry field. When all of the information is correct, the *Define* push-button is selected at the bottom of the *Define a Regulatory Program Record* entry screen to accept the record and update the database. If an error is detected (e.g.,

Define a Regulatory Program Record

Document Type: CDS

Review Plan Number: 12.0

Title: EVIDENCE OF EXTREME EROSION

Define
Close
Help

Figure 2-23. Define a regulatory program record input screen

the review plan number already exists for the type of document selected), an error message will display. When the update has been completed, the *Define* push-button returns to its normal appearance, and a message displays indicating whether or not the definition was successful. The *Close* push-button at the bottom of the screen is selected to exit from the *Define a Regulatory Program Record* screen.

Define a Regulatory Record Entry Screen—Document Type Drop-down Processing

When the *Document Type* drop-down indicator is selected from the *Define a Regulatory Program Record* screen, the `_intern_DefineDocType` event is recognized by the main event loop. This causes code embedded within the case statement in the main program to be executed. The title of the screen is examined, and if the character string “OITS” is found, the screen is initialized to the proper format for OITS records. Otherwise, the screen is initialized to the proper format for CDS, CDM, and RPS records.

Define a Regulatory Record Entry Screen—Define Push-button Processing

When the *Define* push-button is selected from the *Define a Regulatory Program Record* screen, the `_intern_DefineOkayButton` event is recognized by the main event loop. This causes the define function to be called. The length of the *Review Plan Number* is checked and if the field is empty, and error

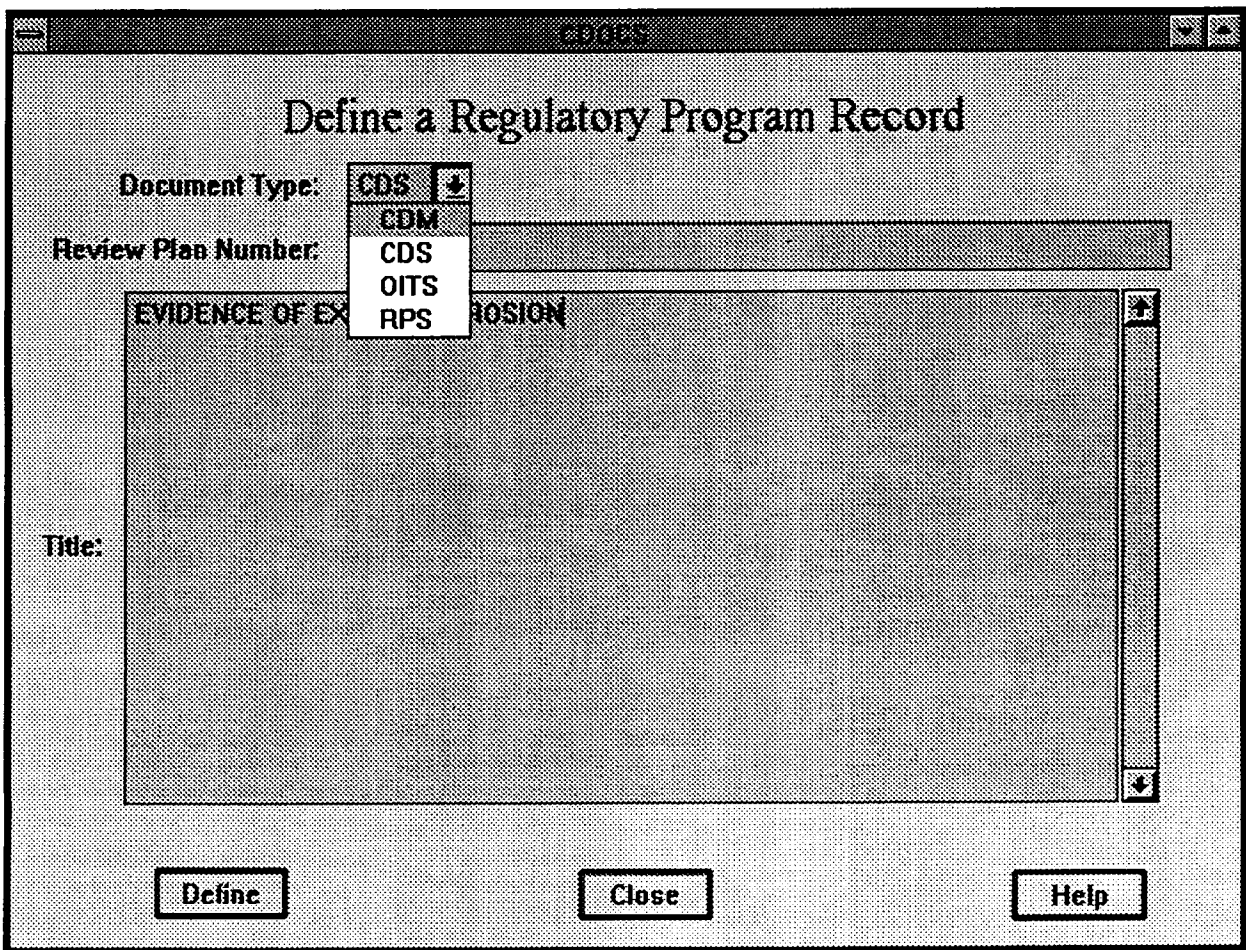


Figure 2-24. Define a regulatory program record document type selection

message is displayed. The *Review Plan Number* is checked for invalid characters and an error message is displayed if necessary. The *Title* is checked for validity and an error message is displayed if the *Title* is missing. If no errors are found, the *RPCDefineDocument* function is called to define the record. If the record definition is successful, a confirmation message is displayed.

Define a Regulatory Record Entry Screen—Close Push-button Processing

When the *Close* push-button is selected from the *Define a Regulatory Program Record* screen, the *_intern_DefineCancelButton* event is recognized by the main event loop. This causes the *closeDefine* function to be called.

Defining New Open Item Tracking System Records

Record header information for OITS records, such as OITS identifier and topic, must be preloaded in the database before textual data associated with the records may be entered. As with regulatory program records, the record header preloading process creates a place for the data, and the check in process actually stores the textual information.

Define a Regulatory Program Record

Document Type:

Review Plan Number:

Title:

Figure 2-25. Define a regulatory program record

Define an OITS Record—Processing

The *Define a Regulatory Program Record* screen is used for all regulatory record types (CDS, CDM,RPS, and OITS). The OITS records differ from the other types of regulatory records and require different information. Therefore, when the *Document Type* is changed to OITS, the *Define a Regulatory Program Record* screen is reconfigured to reflect an OITS record.

When the *Define a Record* entry is selected from the *Process a Regulatory Record* cascading menu, the `_intern_DefineDefine` event is recognized by the main event loop. This causes the `loadDefine` function to be executed. The `loadDefine` function calls the `loadDocTypes` function, which then calls the `RPCOpenDocTypeCursor` and `RPCFetchDocTypeRow` functions to obtain a list of valid document types and load them into the *Document Type* drop-down list. The `loadDefine` function then calls the `openDialog` function to open the Define dialog and display the *Define a Regulatory Program Record* entry screen. The OITS record definition and header preloading process requires an entry screen through which the document type, OITS identifier, and topic are entered. When the define functionality is selected, the record definition screen for regulatory program records appears (Figure 2-25). When the OITS option is selected

for *Document Type*, the appearance of the screen changes to accommodate entry of OITS information (Figure 2-26).

2.2.10.2 Enter New and Updated Records

Once a record has been defined in the database, the textual information associated with that record may be entered. The enter a record process is used: (i) whenever the text of a new record is loaded for the first time or (ii) whenever a record's text is changed and reloaded. The information to be entered is submitted as a WordPerfect file on a diskette or on the network hard disk. This data file must be correctly identified and validated through the appropriate entry screen before the data is accepted and updated in CDOCS. The entry screens for regulatory program records (CDS, CDM, and RPS) differ from those for OITS. Therefore, the entry processes for regulatory program and OITS records are treated separately.

When the *Enter a Regulatory Record* entry is selected from the *Process a Regulatory Record* cascading menu, the *Enter a Regulatory Program Record* entry screen appears and the *Document Type* pull-down indicator is set to the default value of CDM (Figure 2-27).

If a different document type is required, the *Document Type* pull-down indicator is selected to display the pull-down list of document types, and the desired document type is selected (e.g., CDS, CDM, or RPS) (Figure 2-28). All of the active and defined review plan numbers and titles for the selected document type are displayed in the *Review Plan Number and Title* list view. The line in the *Review Plan Number and Title* list view that contains the desired review plan number and title is selected. A radio button under the *Entry Type:* heading is selected to indicate a *Major Revision* or *Minor Revision*. The appropriate number is entered in the *New Version Number* entry field. The system computes a new version number based on the current version number and the type of change (major or minor), and the new computed version number is compared to the entered version number. If the version numbers do not match, the system displays an error message. The *Filename for Entry Document* is keyed into the space provided. The *File Chooser* push-button next to the *Filename for Entry Document* entry field may be selected to cause a list of files to be displayed. The entered information is reviewed visually and corrections are made as required. The *Enter* push-button at the bottom of the *Enter a Regulatory Program Record* entry screen is selected to accept the entry information and update the database. A message is displayed indicating whether or not the entry was successful.

If the document type, review plan number, or title in the submitted document file does not match the corresponding information on the *Enter a Regulatory Program Record* screen, the system assumes that a wrong file has been submitted. This causes an error message to be displayed, showing the expected versus the submitted document type, review plan number, and title. In some cases, such as when a change occurs in the title of a document, the user may want to verify that the correct document has been selected by checking the review plan number, and then accept the input document file and override the title in the *Enter a Regulatory Program Record* screen information by selecting the *Force Entry* push-button at the bottom of the error message screen. The *Close* push-button at the bottom of the error message screen may be selected to exit from the *Enter a Regulatory Program Record* screen.

OTTS

Define an OTTS Record

Document Type: OTTS

OTTS ID: OREGUNC0912251051

Topic: USE OF THE PHRASE "QUATERNARY PERIOD"

Define Close Help

Figure 2-26. Define an OTTS record

Enter a Regulatory Record—Processing

When the *Enter a Record* entry is selected from the *Process a Regulatory Record* cascading menu, the `_intern_CheckinCheckin` event is recognized by the main event loop. This causes the `loadCheckin` function to be executed to initialize and display the *Enter a Regulatory Program Record* entry screen. The `loadDocTypes` function is executed to obtain a list of valid document types, and the *Document Type* pull-down indicator is set to the default value of CDM.

Enter a Regulatory Program Record

Document Type: **CDM** ▾

Review Plan Number and Title:

1.1	General Description of the Facility
1.2	Basis for Licensing Authority
1.3	Schedules
1.4	Certification of Safeguards
1.5	Physical Security Plan
1.6.1	Site Characterization Work Conducted
1.6.2	Status of DOE Resolution of NRC Objection
1.7	Statement of Compliance with the Performance
10.0	QUALITY ASSURANCE

Checkin Type: ☒ Major Revision ☐ Minor Revision

New Version Number:

Filename for Checkin Document: **File Chooser**

Checkin **Close** **Help**

Figure 2-27. Enter a regulatory program record input screen

Enter a Regulatory Record—Document Type Drop-down List Processing

When the *Document Type* drop down indicator is selected from the *Enter a Regulatory Program Record* screen, the `_intern_CheckinDocType` event is recognized by the main event loop. This causes code embedded within the main event loop to be executed. The `loadDocs` function is called with a parameter of `VACANT_ACTIVE` and the selected document type to obtain a list of document titles and numbers for vacant and active records. If the document type is `OITS`, the screen title is changed to *Enter an OITS Record* and the label of the list view is changed to *OITS ID and Topic*. If the document type is not `OITS`, the screen title is changed to *Enter a Regulatory Program Record* and the label of the list view is changed to *Review Plan Number and Title*.

Enter a Regulatory Record—Enter Push-button Processing

When the *Enter* push-button is selected from the *Enter a Regulatory Program Record* screen, the `_intern_CheckinOkayButton` event is recognized by the main event loop. This causes the `checkin` function to be called. The screen title is checked for the character string “OITS” to determine whether the record entry transaction is for a regulatory program record or an OITS record.

Enter a Regulatory Program Record

Document Type: CDM

Review Plan Number: CDS

1.1	General Description of the Facility
1.2	Basis for Licensing Authority
1.3	Schedules
1.4	Certification of Safeguards
1.5	Physical Security Plan
1.6.1	Site Characterization Work Conducted
1.6.2	Status of DOE Resolution of NRC Objection
1.7	Statement of Compliance with the Performance
10.0	QUALITY ASSURANCE

Checkin Type: ☒ Major Revision ☐ Minor Revision

New Version Number:

Filename for Checkin Document: File Chooser

Checkin Close Help

Figure 2-28. Enter a regulatory program record document type selection

If “OITS” is not present in the title, the function performs validity checks for regulatory program records. If the version number is not present, an error message is displayed. The radio button value for major or minor change is obtained, and the list view is examined. If no document is selected, an error message is displayed.

If no errors are detected, the *RPCCheckinDocument* function is called to pass the information to the server and enter the record. If the return value from the *RPCCheckinDocument* function indicates a successful record entry, a confirmation message is displayed.

Enter a Regulatory Record—File Chooser Push-button Processing

When the *File Chooser* push-button is selected from the *Enter a Regulatory Program Record* screen, the *_intern_CheckInFileChooserButton* event is recognized by the main event loop. This causes the *checkinfilechooser* function to be called. If the file chooser dialog has not been previously opened, the *checkinfilechooser* function calls the *vfilechrCreate* function to load it. Then the *vfilechrOpen* function is called to open the file chooser and obtain the file name. When the file chooser is closed, the *vfilechrSetApply* function is called to retrieve and store the file name.

Enter a Regulatory Record—Close Push-button Processing

When the *Close* push-button is selected from the *Enter a Regulatory Program Record* screen, the *_intern_CheckinCancelButton* event is recognized by the main event loop. This causes the *closeCheckin* function to be called and the dialog is closed.

Enter a Regulatory Record—Error Message Processing

If the document type, review plan number, or title in the submitted document file does not match the corresponding information on the *Enter a Regulatory Program Record* screen, the system assumes that a wrong file has been submitted. This causes an error message to be displayed, showing the expected versus the submitted document type, review plan number, and title.

The error checking for critical errors occurs in the checkin function following the call to the *RPCCheckinDocument* function. If the return value from the *RPCCheckinDocument* function indicates a *SVC_VERSION_ERROR* error code, the *RPCGetMismatches* function is called to obtain the review plan and title of the existing document in the database and an error message is prepared indicating the mismatch between the expected and entered version number. If the return value indicates a *SVC_MISMATCH_ERROR* error code, the *RPCGetMismatches* function is called to obtain the review plan and title of the existing document in the database and an error message is prepared indicating the mismatch between the expected and entered review plan number. If the entered document type does not match the document type in the submitted WordPerfect file, an error message is displayed indicating the mismatch. Finally, function checks for critical SQL errors. If any of the critical errors are found, the *checkinabortdialog* is loaded to inform the user that the record entry transaction has been aborted.

If the review plan title does not match the title of the submitted file, the error is considered non-critical, but the *checkinforcedialog* is loaded to display the error messages and permit the record entry to be forced.

Enter a Regulatory Record Error Message —Force Push-button Processing

In some cases, such as when a change occurs in the title of a document, the user may want to verify that the correct document has been selected by checking the review plan number. Then the document may be accepted and the title may be overridden in the *Enter a Regulatory Program Record* screen. Finally, the *Force Entry* push-button may be selected at the bottom of the error message screen.

When the *Force Entry* push-button is selected, the *_intern_CheckinForceForceButton* event is recognized by the main event loop and the *forcecheckin* function is called. The information is retrieved from the selected entry in the list view, and the *RPCCheckinDocument* function is called with a *FORCE_CHECKIN* parameter to accept the document and suppress mis-match errors. If any error is detected, an error message is displayed and the dialog is closed. If no errors are detected, a confirmation message is displayed before closing the dialog.

Enter a Regulatory Record Error Message —Close Push-button Processing

The *Close* push-button at the bottom of the error message screen may be selected to exit from the *Enter a Regulatory Program Record* screen. When the *Close* push-button is selected from the *Enter*

a *Regulatory Program Record* screen, the *_intern_CheckinforceCancelButton* event is recognized by the main event loop. This causes the *abortcheckin* function to be called, and the dialog is closed.

2.2.10.3 Enter Open Item Tracking System Records

OITS records are entered using the same screens as other regulatory records, but the screens are highly modified to accommodate the information requirements of OITS records when the OITS entry is selected from the *Document Type* drop-down list. When the *Document Type* drop-down indicator is selected to display the drop-down list of document types and the OITS entry is selected, the appearance of the screen changes and all of the active and defined OITS identifiers and topics are displayed in the *OITS ID and Topic* list view. The line in the OITS ID and Topic list view that contains the desired OITS identifier and topic is selected, and the *Enter an OITS Record* screen is displayed (Figure 2-29).

The *Filename for Entry Document* is entered in the space provided, and the *File Chooser* push-button next to the *Filename for Entry Document* entry field may be selected to cause a list of files to be displayed (Figure 2-28). When all of the information is correct, the *Enter* push-button at the bottom of the *Enter an OITS Record* entry screen is selected to accept the entered OITS information and update the database. When the record entry process completes, the *Enter* push-button returns to its normal appearance, and a message is displayed indicating whether or not the record entry was successful.

If the document type, OITS identifier, or topic in the submitted document file does not match the corresponding information on the *Enter an OITS Record* screen, the system assumes that the wrong file has been submitted. This causes the system to pause and display an error message, showing the expected versus the submitted document type, OITS identifier, and topic. The user may verify that the correct document has been selected, accept the input document file, and override the topic in the *Enter an OITS Record* screen information by selecting the *Force Entry* push-button at the bottom of the error message screen. The *Close* push-button at the bottom of the screen may be selected to exit from the *Enter an OITS Record* screen.

Enter an OITS Record—Processing

When the *Enter a Regulatory Record* entry is selected from the *Process a Regulatory Record* cascading menu, the *_intern_CheckinCheckin* event is recognized by the main event loop. This causes the *loadCheckin* function to be executed to initialize and display the *Enter a Regulatory Program Record* entry screen. The *loadDocTypes* function is executed to obtain a list of valid document types, and the *Document Type* pull-down indicator is set to the default value of CDM.

Enter an OITS Record—Document Type Drop-down List Processing

When the *Document Type* drop down indicator is selected from the *Enter a Regulatory Program Record* screen, the *_intern_CheckinDocType* event is recognized by the main event loop. This causes code embedded within the main event loop to be executed. The *loadDocs* function is called with a parameter of *VACANT_ACTIVE* and the selected document type to obtain a list of document titles and numbers for vacant and active records. If the document type is OITS, the screen title is changed to *Enter an OITS Record* and the label of the list view is changed to *OITS ID and Topic*. If the document type is not OITS, the screen title is changed to *Enter a Regulatory Program Record* and the label of the list view is changed to *Review Plan Number and Title*.

Enter an OITS Record

Document Type: **OITS**

OITS ID and Topic:

0A0017APR1992C001	Disposal of waste forms other than SF and
0A0017APR1992C002	Misplacement of potential impacts to the a
0A0017APR1992C003	Misplacement of discussion on performanc
0A0028MAY1993C001	PAC's may not be appropriately considere
0A0028MAY1993C002	Consideration of present PAC/FACs may be
0A0028MAY1993C003	Lack of clear relationship between PAC/FA
0A0028MAY1993C004	Selsmic network monitoring and geodetic t
0A0030NOV1993C001	Repository systems approach of FCRG not
0A0030SEP1992C001	Possible occurrences of potential disrupti

Checkin Type: ☒ Major Revision ☐ Minor Revision

New Version Number:

Filename for Checkin Document: **File Chooser**

Checkin **Close** **Help**

Figure 2-29. Enter an OITS record

Enter an OITS Record—File Chooser Processing

When the *File Chooser* push-button is selected from the *Enter an OITS Record* screen, the *_intern_CheckInFileChooserButton* event is recognized by the main event loop. This causes the *checkinfilechooser* function to be called. If the file chooser dialog does not exist, the *checkinfilechooser* function calls the *vfilechsrCreate* function to create it. Then the *vfilechsrOpen* function is called to open the file chooser and obtain the file name. When the file chooser is closed, the *vfilechsrSetApply* function is called to retrieve the file name.

Enter an OITS Record—Enter Push-button Processing

When the *Enter* push-button is selected from the *Enter an OITS Record* screen, the *_intern_CheckinOkayButton* event is recognized by the main event loop. This causes the *checkin* function to be called. The screen title is checked for the character string “OITS” to determine whether the record entry transaction is for a regulatory program record or an OITS record. If “OITS” is present in the title, the function sets the change type to *MAJOR_VERSION_CHANGE*. If no document is selected, an error message is displayed.

If no errors are detected, the *RPCCheckinDocument* function is called to pass the information to the server and enter the record. If the return value from the *RPCCheckinDocument* function indicates a successful record entry, a confirmation message is displayed.

Enter an OITS Record—Close Push-button Processing

When the *Close* push-button is selected from the *Enter an OITS Record* screen, the *_intern_CheckinCancelButton* event is recognized by the main event loop. This causes the *closeCheckin* function to be called and the dialog is closed.

Enter an OITS Record—Error Message Processing

If the document type, OITS ID, or title in the submitted document file does not match the corresponding information on the *Enter an OITS Record* screen, the system assumes that the wrong file has been submitted. This causes an error message to be displayed, showing the expected versus the submitted document type, OITS ID, and title.

The error checking for critical errors occurs in the *checkin* function following the call to the *RPCCheckinDocument* function. If the return value from the *RPCCheckinDocument* function indicates a *SVC_VERSION_ERROR* error code, the *RPCGetMismatches* function is called to obtain the OITS ID and title of the existing document in the database and an error message is prepared indicating the mismatch between the expected and entered version number. If the return value indicates a *SVC_MISMATCH_ERROR* error code, the *RPCGetMismatches* function is called to obtain the OITS ID and title of the existing document in the database and an error message is prepared indicating the mismatch between the expected and entered OITS ID. If the entered document type does not match the document type in the submitted WordPerfect file, an error message is displayed indicating the mismatch. Finally, function checks for critical SQL errors. If any of the critical errors are found, the *checkinabortdialog* is loaded to inform the user that the record entry transaction has been aborted.

If the OITS title does not match the title of the submitted file, the error is considered non-critical, but the *checkinforcedialog* is loaded to display the error messages and permit the record entry to be forced.

Enter an OITS Record—Force Push-button Processing

In some cases, such as when a change occurs in the title of a document, the user may want to verify that the correct document has been selected by checking the OITS ID, and then accepting the input document file and overriding the title in the *Enter an OITS Record* screen information by selecting the *Force Entry* push-button at the bottom of the error message screen.

When the *Force Entry* push-button is selected, the *_intern_CheckinForceForceButton* event is recognized by the main event loop and the *forcecheckin* function is called. The information is retrieved from the selected entry in the list view, and the *RPCCheckinDocument* function is called with a *FORCE_CHECKIN* parameter to check in the document and suppress mis-match errors. If any error is detected, an error message is displayed and the dialog is closed. If no errors are detected, a confirmation message is displayed before closing the dialog.

2.2.10.4 Format Check for New and Updated Regulatory Records

The CDOCS regulatory record processing function has a facility for checking the format of input records prior to submitting them. The format check facility is nearly identical to the record entry process described in Section 2.2.10.2, except that the database is not updated. All input edits are performed and any error conditions are included in a format check error report. The *Format Check* entry screens for regulatory program and OITS records require the record to be selected from a list view of available records. The input file must be identified either through entry of the full path and file name or through selection with the file chooser facility. The *Format Check a Regulatory Record* entry is selected from the *Enter a Regulatory Record* cascading menu, and the *Format Check a Regulatory Program Record* entry screen will appear (Figure 2-30).

The *Document Type* pull-down indicator is selected to display the drop-down list of document types, and the desired regulatory program document type (e.g., CDS, CDM, or RPS) is selected. All of the active and defined review plan numbers and titles for the selected document type appear in the *Review Plan Number and Title* list view. The line in the *Review Plan Number and Title* list view that contains the desired review plan number and title is selected, and either the *Major Revision* or *Minor Revision* radio button under the *Format Check Type* heading is selected. The appropriate number is entered in the *New Version Number* entry field for the document being format checked, and the system computes a new version number based on the current version number and the type of change (major or minor). The computed version number is compared to the entered version number, and if they do not match, the system displays an error message.

The *Filename for Format Check Document* is entered in the space provided, and the *File Chooser* push-button next to the *Filename for Format Check Document* entry field may be selected to display a list of files for selection. When all of the information is correct, the *Format Check* push-button at the bottom of the *Format Check a Regulatory Program Record* entry screen is selected to accept and validate the format check input information. When the format check process completes, the *Format Check* push-button returns to its normal appearance, and a message displays indicating whether or not the format check was successful. If the document type, review plan number, or title in the submitted document file does not match the corresponding information on the *Format Check a Regulatory Program Record* input screen, the system assumes that the wrong file has been submitted and an error message is displayed, showing the expected versus the submitted document type, review plan number, and title. The *Close* push-button at the bottom of the screen is selected to exit from the *Format Check a Regulatory Program Record* screen.

Format Check a Regulatory Program Record —Processing

When the *Format Check a Regulatory Record* entry is selected from the *Process a Regulatory Record* cascading menu, the *_intern_CheckinFormat* event is recognized by the main event loop. This causes the *loadFormatCheck* function to be executed to initialize and display the *Format Check a Regulatory Program Record* entry screen. The callback for the *Help* push-button is set, the *loadDocTypes* function is executed to obtain a list of valid document types, and the *Document Type* pull-down indicator is set to the default value of CDM.

Format Check a Regulatory Program Record

Document Type: **CDM** ▾

Review Plan Number and Title:

2.6	License Specifications
2.7	Nuclear Material Control
3.1.1	Description of Individual Systems and Characteristics
3.1.2	DESCRIPTION OF INDIVIDUAL SYSTEMS AND CHARACTERISTICS
3.1.3	DESCRIPTION OF INDIVIDUAL SYSTEMS AND CHARACTERISTICS
3.1.4	DESCRIPTION OF INDIVIDUAL SYSTEMS AND CHARACTERISTICS
3.1.5	Description of the Description of the Integrated Natural
3.2.1.1	Nature and Rates of Physical Processes
3.2.1.10	POTENTIALLY ADVERSE CONDITION: EVIDENCE OF

Format Check Type: ☒ Major Revision ☐ Minor Revision

New Version Number:

Filename for Format Check Document: **File Chooser**

Format Check **Close** **Help**

Figure 2-30. Format check a regulatory program record screen

Format Check a Regulatory Program Record—Document Type Drop-down List Processing

When the *Document Type* drop down indicator is selected from the *Format Check a Regulatory Program Record* screen, the `_intern_FormatCheckDocType` event is recognized by the main event loop. This causes code embedded within the main event loop to be executed. The `loadDocs` function is called with a parameter of `VACANT_ACTIVE` and the selected document type to obtain a list of list of document titles and numbers for vacant and active records. If the document type is `OITS`, the screen title is changed to *Format Check an OITS Record* and the label of the list view is changed to *OITS ID and Topic*. If the document type is not `OITS`, the screen title is changed to *Format Check a Regulatory Program Record* and the label of the list view is changed to *Review Plan Number and Title*.

Format Check a Regulatory Program Record —File Chooser Processing

When the *File Chooser* push-button is selected from the *Format Check a Regulatory Program Record* screen, the `_intern_FormatCheckFileChooserButton` event is recognized by the main event loop. This causes the `formatcheckfilechooser` function to be called. If the file chooser dialog does not exist, the `checkinfilechooser` function calls the `vfilechsrCreate` function to create it. Then the `vfilechsrOpen` function

is called to open the file chooser and obtain the file name. When the file chooser is closed, the *vfilechsrSetApply* function is called to retrieve and store the file name.

Format Check a Regulatory Program Record—Format Check Processing

When the *Format Check* push-button is selected from the *Format Check a Regulatory Program Record* screen, the *_intern_FormatCheckOkayButton* event is recognized by the main event loop. This causes the *formatCheck* function to be called. The *formatCheck* function calls the *vRPCCheckFormat* function to perform validity checking. If no errors are detected, an advisory message is displayed and the *formatCheck* function terminates. If errors are detected, the critical error flag is set, a callback is set for help on the format check abort screen, and the *formatcheckabortdialog* is loaded to display the results of the format check processing.

Format Check a Regulatory Program Record—Close Push-button Processing

When the *Close* push-button is selected from the *Format Check a Regulatory Program Record* screen, the *_intern_FormatCheckCancelButton* event is recognized by the main event loop. This causes the *closeFormatCheck* function to be called and the dialog is closed.

Format Check a Regulatory Program Record—Error Message Processing

If the document type, review plan number, or title in the submitted document file does not match the corresponding information on the *Format Check a Regulatory Program Record* screen, the system assumes that the wrong file has been submitted. This causes an error message to be displayed, showing the expected versus the submitted document type, review plan number, and title.

The error checking for critical errors occurs in the *formatCheck* function following the call to the *vRPCCheckFormat* function. If the return value from the *vRPCCheckFormat* function indicates a *SVC_VERSION_ERROR* error code, the *RPCGetMismatch* function is called to obtain the review plan and title of the existing document in the database and an error message is prepared indicating the mismatch between the expected and entered version number. If the return value indicates a *SVC_MISMATCH_ERROR* error code, the *RPCGetMismatch* function is called to obtain the review plan and title of the existing document in the database and an error message is prepared indicating the mismatch between the expected and entered review plan number. If the entered document type does not match the document type in the submitted WordPerfect file, an error message is displayed indicating the mismatch. The function checks for critical SQL errors but sets the critical error flag in all cases, and the *formatcheckabortdialog* is loaded to inform the user that the format check transaction has been aborted.

Format Check a Regulatory Program Record Error Message—Close Push-button Processing

The *Close* push-button at the bottom of the format check error message screen may be selected to exit from the *Format Check a Regulatory Program Record* screen. When the *Close* push-button is selected from the *Format Check a Regulatory Program Record* error message screen, the *_intern_FormatCheckAbortCloseButton* event is recognized, and the dialog is closed.

2.2.10.5 Format Check for Open Item Tracking System Records

The *Format Check a Regulatory Program Record* screen is used for all regulatory record types (CDS, CDM, RPS, and OITS). The OITS records differ from the other types of regulatory records and require different information. Therefore, when the *Document Type* is changed to OITS, the *Format Check a Regulatory Program Record* screen is reconfigured to reflect an OITS record.

The *Format Check a Regulatory Program Record* entry is selected from the *Enter a Regulatory Program Record* cascading menu. The *Document Type* drop-down indicator is selected to display the drop-down list of document types, and the OITS entry is selected. The appearance of the input screen changes to reflect OITS records and all of the active and defined OITS identifiers and topics are displayed in the *OITS Identifier and Topic* list view. The line in the *OITS ID and Topic* list view that contains the desired OITS identifier and topic is selected, and the *Filename for Format Check Document* is entered in the space provided. The *File Chooser* push-button next to the *Filename for Format Check Document* entry field may be selected to display a list of files.

When all of the information is correct, the *Format Check* push-button at the bottom of the *Format Check an OITS Record* entry screen is selected to accept and validate the format check input information for the submitted OITS record. When the format check process completes, the *Format Check* push-button returns to its normal appearance, and a message is displayed indicating whether or not the format check was successful. The *Close* push-button at the bottom of the screen may be selected to exit from the *Format Check an OITS Record* screen.

2.2.10.6 Retire Regulatory Program Records

A regulatory program record may become obsolete and may need to be removed from the active database. The record retirement process is used to archive an outdated regulatory program record and remove it from the active database. The information in the retired record is physically retained, but is no longer available to users through the normal access methods. The record to be retired must be positively identified through the *Retire a Regulatory Program Record* screen before it can be archived and removed from the active database.

To retire a regulatory program record the *Retire a Regulatory Record* entry is selected from the *Process a Regulatory Record* cascading menu. The *Retire a Regulatory Program Record* entry screen appears (Figure 2-31). The *Document Type* drop-down indicator is selected to display the drop-down list of document types, and the desired regulatory program document type (e.g., CDS, CDM, or RPS) is selected. All of the active and defined review plan numbers and titles for the selected document type appear in the *Review Plan Number and Title* list view. The line in the *Review Plan Number and Title* list view that contains the desired review plan number and title is selected.

When the correct record has been selected, the *Retire* push-button at the bottom of the *Retire a Regulatory Program Record* screen is selected. The system displays a confirmation message and requests the user to verify that the correct record has been selected. The *Retire* push-button in the message window is selected to continue the record retire process. If the wrong record has been selected, the *Close* push-button at the bottom of the *Retire a Regulatory Program Record* screen may be selected to exit from the screen. When a record has been retired, a message appears at the bottom of the screen to confirm that the record retirement transaction completed successfully. The *Close* push-button at the bottom of the screen may be selected to exit from the *Retire a Regulatory Program Record* screen.

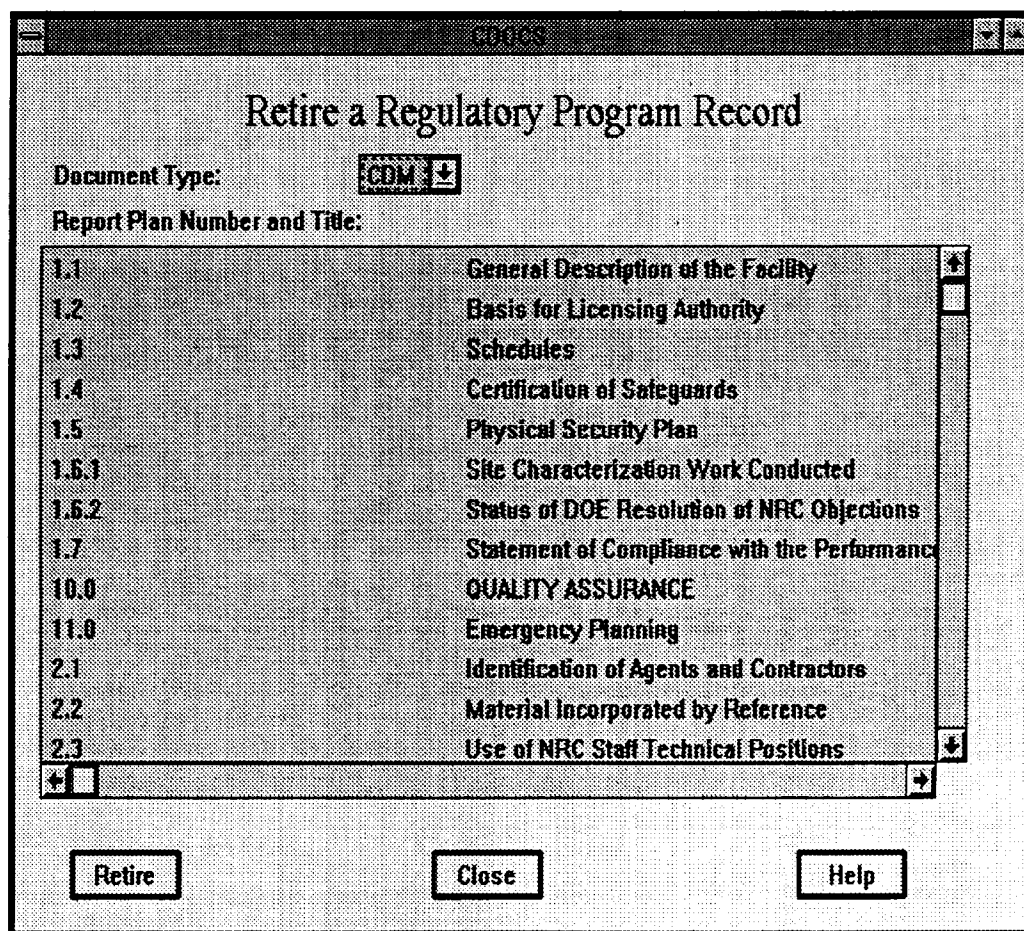


Figure 2-31. Retire a regulatory program record screen

Retire a Regulatory Program Record—Processing

When the *Retire a Regulatory Record* entry is selected from the *Process a Regulatory Record* cascading menu, the `_intern_MaintRetirement` event is recognized by the main event loop. This causes the `loadRetirement` function to be executed to initialize and display the *Retire a Regulatory Program Record* screen.

The callback for the *Help* push-button is set, the `loadDocTypes` function is executed to obtain a list of valid document types, and the *Document Type* pull-down indicator is set to the default value of CDM. The `loadDocs` function is then called to retrieve the review plan number and title of all active and vacant records. Then the `retirementdialog` is opened to display the *Retire a Regulatory Program Record* screen.

Retire a Regulatory Program Record—Document Type Drop-down List Processing

When the *Document Type* drop down indicator is selected from the *Retire a Regulatory Program Record* screen, the `_intern_RetirementDocType` event is recognized by the main event loop. This causes

code embedded within the main event loop to be executed. The *loadDocs* function is called with a parameter of VACANT_ACTIVE and the selected document type to obtain a list of list of document titles and numbers for vacant and active records.

Retire a Regulatory Program Record—Retire Push-button Processing

When the *Retire* push-button is selected from the *Retire a Regulatory Program Record* screen, the *_intern_RetirementOkayButton* event is recognized by the main event loop. This causes the retire function to be called. The retire function retrieves the document type and review plan number and title. If no review plan has been selected, an error message is displayed. The retirement confirmation dialog is initialized, a callback is set for the *Help* push-button, and the retirement confirmation dialog is opened.

Retire a Regulatory Program Record—Close Push-Button Processing

When the *Close* push-button is selected from the *Retire a Regulatory Program Record* screen, the *_intern_RetirementCancelButton* event is recognized by the main event loop. This causes the *closeRetirement* function to be called and the dialog is closed

Retire a Regulatory Program Record Confirmation Screen—Retire Push-button Processing

When the *Retire* push-button is selected from the *Retire a Regulatory Program Record* screen, the *_intern_ConfirmRetireRetire* event is recognized by the main event loop. This causes the *reallyretire* function to be called. The *reallyretire* function retrieves the review plan number and document type and calls the *RPCRetireDocument* function to retire the document. If an error is detected in the retirement process on the server, an error message is displayed. If no errors are detected, an advisory message is displayed indicating that the record has been retired, the entry for the document is deleted from the list view of review plan numbers and titles, and the dialog is closed.

Retire a Regulatory Program Record Confirmation Screen—Close Push-button Processing

When the *Close* push-button is selected from the *Retire a Regulatory Program Record* confirmation screen, the *_intern_ConfirmRetireCancel* event is recognized by the main event loop. This causes the *dontretire* function to be called. The *dontretire* function displays a message that the retirement for the specified record has been abandoned, and the dialog is closed.

2.2.11 Custodian Functions For Library Control

The CNWRA maintains a library of selected documents. CDOCS performs additional functions at the CNWRA associated with circulation control of these documents. The CDOCS circulation control facilities are not needed at the NRC, therefore, the circulation control options do not appear on the NRC Custodian pull-down menu.

2.2.11.1 Checking Out a Consolidated Document System Record

CDOCS provides facilities for circulation control of hard-copy documents associated with certain document sets. The circulation control facilities permit a document to be checked out from the CNWRA library.

Checkout Functionality

The *Checkout Document* entry is selected from the CNWRA *Custodian* pull-down menu in the *CDOCS Main Menu*, and the *CDOCS Checkout* entry screen appears (Figure 2-32) with the *Document Set* drop-down list is initialized to the default document set.

The number of the document to be checked out is entered in the *Document Number* entry field, and the staff member's name is selected from the *Selection List*. If the name is not in the *Selection List*, it may be entered in the *Other Names* field. The *OK* push-button at the right of the screen is selected to check out the record.

The check out process validates the document number that was entered. If an invalid document number was entered, an error message appears prompting reentry of the document number. If the document is already checked out to someone else, an error message is displayed, and prompts for reentry of the document number. If the document number is valid and the document is not already checked out, the document check out process displays an advisory message, including the title of the record being checked out, to indicate that the checkout process was completed successfully. The *Close* push-button at the right of the screen may be selected to exit from the *CDOCS Checkout* entry screen.

Document Checkout Screen—Processing

When the *Checkout Document* entry is selected from the *Custodian* pull-down menu, the *_intern_CustodianCheckout* event is recognized by the main event loop. This causes the *Checkout* function to be executed. The *Checkout* function calls *BuildDocSet* to obtain a list of available document sets, and then opens the *checkoutDialog* dialog.

Document Checkout Screen—OK Push-button Processing

When the *OK* push-button is selected from the *CDOCS Checkout* screen, the *Checkout OK Tag* callback is activated and the *CheckoutProc* function is called. When the callbacks for the *CDOCS Checkout* screen are initialized in the *LoadCheckout* function, the *OK* and *Close* push-buttons are both set to call the *CheckoutProc* function, which examines the callback tag. If the callback tag is *Checkout OK Tag* the *NamesSelectProc* function is called to obtain the name of the person checking out the document. The document number is retrieved from the dialog and extraneous white space is stripped. The *CheckTDOCSDocNum* function is called to validate the format of the document number. If the document number is invalid, an error message is displayed. The *CheckTDOCSUserName* function is called to validate the format of the user name. If the user name is invalid, an error message is displayed.

The *Confirm* function is called to check the document out. The *Confirm* function calls the *RPCGetRec* function to retrieve the record. If the record is not found, an error message is displayed. If the record is successfully retrieved, the document number and the title of the document are retrieved from the record, the *currentTDOCSOperation* variable is set to *TDOCSCHECKOUT* and the *TDOCSConfirmDialog* dialog is opened to accept or cancel the document checkout transaction.

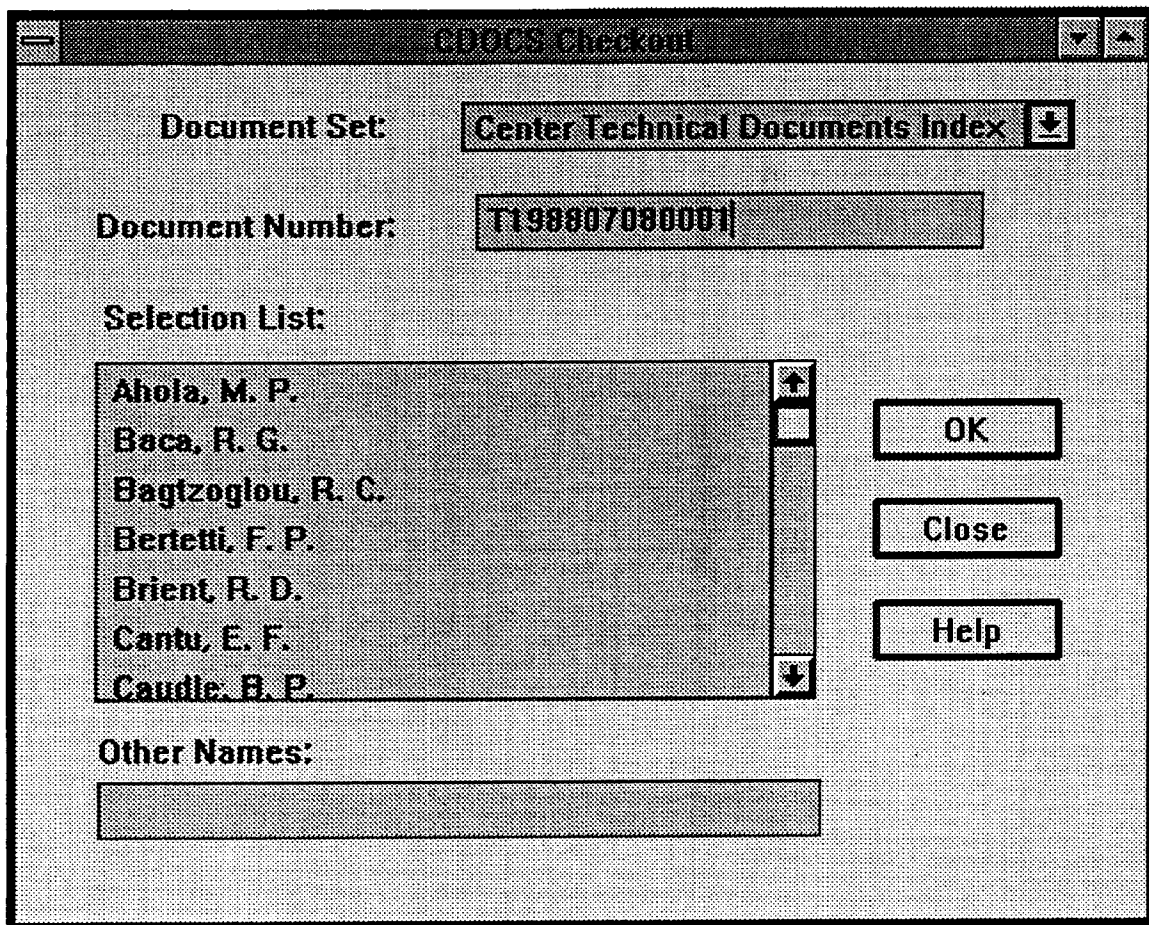


Figure 2-32. CDOCS checkout entry screen

Document Checkout Screen—Close Push-button Processing

When the *Close* push-button is selected from the *CDOCS Checkout* screen, the *Checkout Cancel Tag* callback is activated and the *CheckoutProc* function is called. When the callbacks for the *CDOCS Checkout* screen are initialized in the *LoadCheckout* function, the *OK* and *Close* push-buttons are both set to call the *CheckoutProc* function, which examines the callback tag. If the callback tag is *Checkout Cancel Tag* the dialog is closed.

CDOCS Confirm Screen—Functionality

A common function is used to confirm circulation control transactions. The *CDOCS Confirm* screen can be opened from several functions. When the callbacks for the *CDOCS Confirm* screen are initialized in the *LoadConfirm* function, the *OK* and *Close* push-buttons are both set to call the *ConfirmProc* function, which examines the callback tag.

CDOCS Confirm Screen—OK Push-button Processing

When the *OK* push-button is selected from the *CDOCS Confirm* screen, the *TDOCSConfirmOK* callback is activated and the *ConfirmProc* function is called. If the callback tag is *TDOCSConfirmOK* and the *currentTDOCSOperation* variable is set to *TDOCSCHECKOUT*, the *RPCCheckout* function is called to check the document out.

The *RPCCheckout* function calls the *vRPCCheckoutRecord* function with a parameter of *TDOCS_CHECKOUT_REQUEST*. If the return code from the *vRPCCheckoutRecord* function is *SVC_CHECKED_OUT_ERROR*, the *ForceCheckout* function is called to give the user the option of forcing the check out transaction.

If the check out transaction is successful, a confirmation message is displayed and the *checkoutDialog* and *Confirm* dialogs are closed.

CDOCS Confirm Screen—Cancel Push-button Processing

When the *Cancel* push-button is selected from the *CDOCS Confirm* screen, the *TDOCSConfirmCancel* callback is activated and the *ConfirmProc* function is called. The *ConfirmProc* function examines the callback tag, and the *CDOCS Confirm* dialog is closed.

CDOCS Force Checkout Screen—Processing

When a document checkout is attempted for a document that is already checked out, the *RPCCheckout* function calls the *ForceCheckout* function is called to give the user the option of forcing the check out transaction. The *TDOCSForceCheckoutDialog* is opened and a message is displayed indicating that the requested document is already checked out.

CDOCS Force Checkout Screen—OK Push-button

When the *OK* push-button is selected from the *CDOCS Checkout* error message screen, the *TDOCSForceOK* callback is activated and the *ForceCheckoutProc* function is called. If the callback tag is *TDOCSForceOK*, the *vRPCCheckoutRecord* function is called with a parameter of *TDOCS_FORCE_CHECKOUT_REQUEST* to suppress some of the error checking and force the document to be checked out. If the document check out is successful, an advisory message is displayed and the dialog is closed. If errors are detected, an error message is displayed.

CDOCS Force Checkout Screen—Close Push-button

When the *Cancel* push-button is selected from the *CDOCS Checkout* error message screen, the *TDOCSForceCancel* callback is activated and the *ForceCheckoutProc* function is called. The *ForceCheckoutProc* function examines the callback tag, and the *CDOCS Checkout* error message dialog is closed.

2.2.11.2 Checking In a CDOCS Record

CDOCS provides facilities for circulation control of hard-copy documents associated with certain document sets. The circulation control facilities permit the document to be checked in when the document is returned to the CNWRA library.

The *Checkin Document* entry is selected from the *CNWRA Custodian* pull-down menu in the *CDOCS Main Menu*, and the *CDOCS Checkin* entry screen appears with the *Document Set* drop-down list initialized to the default document set. The document number is entered in the *Document Number* entry field and the *OK* push-button at the right of the screen is selected to check in the record (Figure 2-33).

The checkin process validates the document number that was entered. If an invalid document number was entered, an error message appears prompting reentry of the document number. If the document is not checked out to someone, an error message is displayed and prompts for reentry of the document number. If the document number is valid and the document is currently checked out, the document check-in process displays a confirmation screen including the title of the record being checked in. The *OK* push-button is selected to check in the record, and the system displays an advisory message to indicate that the check-in process has completed successfully. The *Close* push-button at the right of the screen is selected to exit the *Document Checkin* entry screen.

Document Checkin Screen—Processing

When the *Checkin Document* entry is selected from the *Custodian* pull-down menu, the *_intern_CustodianCheckin* event is recognized by the main event loop. This causes the *TDOCSCheckin* function to be executed. The *TDOCSCheckin* function calls *BuildDocSet* to obtain a list of available document sets, and then opens the *checkinTDOCSDialog* dialog.

Document Checkin Screen—OK Push-button Processing

When the *OK* push-button is selected from the *CDOCS Checkin* screen, the *Checkin OK Tag* callback is activated and the *CheckinProc* function is called. When the callbacks for the *CDOCS Checkin* screen are initialized in the *LoadTDOCSCheckin* function, the *OK* and *Close* push-buttons are both set to call the *CheckinProc* function, which examines the callback tag.

If the callback tag is *Checkin OK Tag*, the *NamesSelectProc* function is called to obtain the name of the person checking out the document. The document number is retrieved from the dialog and extraneous white space is stripped. The *CheckTDOCSDocNum* function is called to validate the format of the document number. If the document number is invalid, an error message is displayed. Then *currentTDOCSOperation* variable is set to *TDOCSCHECKIN*, and the *Confirm* function is called to check the document in.

The *Confirm* function calls the *RPCGetRec* function to retrieve the record. If the record is not found, an error message is displayed. If the record is successfully retrieved, the document number and the title of the document are retrieved from the record, and the *TDOCSConfirmDialog* dialog is opened to accept or cancel the document checkin transaction.

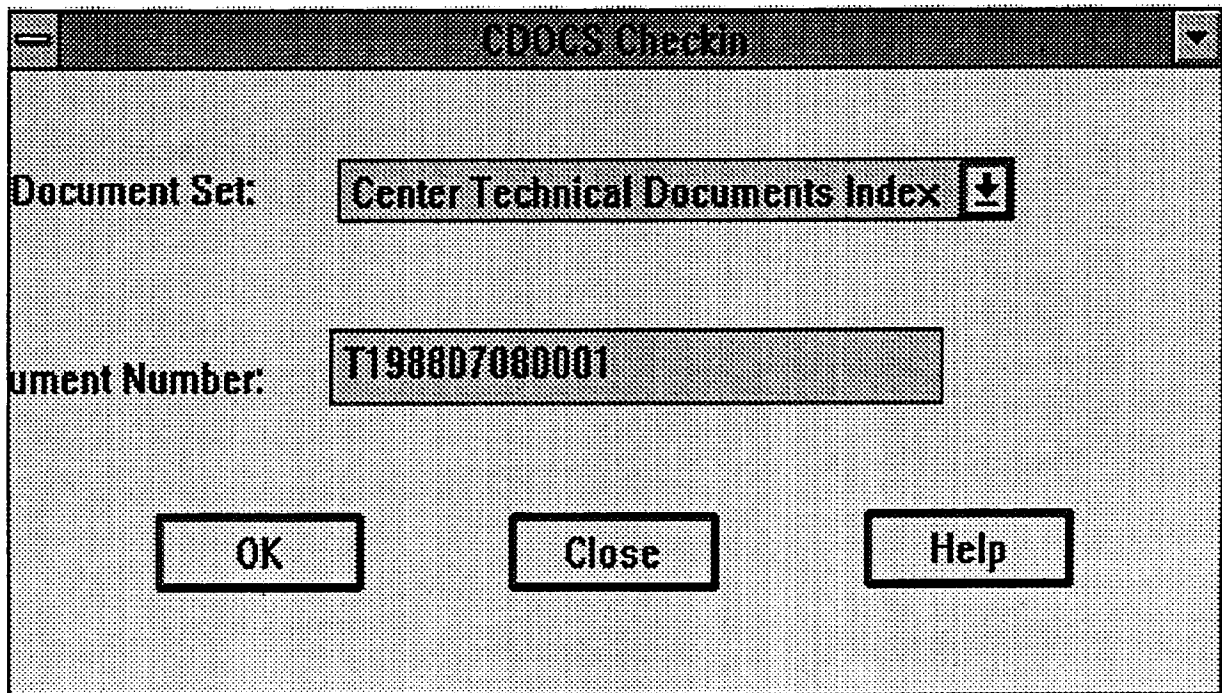


Figure 2-33. CDOCS checkin entry screen

CDOCS Confirm Screen—OK Push-button Processing

When the *OK* push-button is selected from the *CDOCS Confirm* screen, the *TDOCSConfirmOK* callback is activated and the *ConfirmProc* function is called. If the callback tag is *TDOCSConfirmOK* and the *currentTDOCSEOperation* variable is set to *TDOCSCHECKIN*, the *RPCCheckin* function is called to check the document in. If the check in transaction is successful, the *checkinTDOCSDialog* dialog is closed and the *Confirm* dialog is also closed.

CDOCS Confirm Screen—Cancel Push-button Processing

When the *Close* push-button is selected from the *CDOCS Confirm* screen, the *TDOCSConfirmCancel* callback is activated and the *ConfirmProc* function is called. The *ConfirmProc* function examines the callback tag, and the *CDOCS Confirm* dialog is closed.

Document Checkin Screen — Close Push-button Processing

When the *Close* push-button is selected from the *CDOCS Checkin* screen, the *Checkin Cancel Tag* callback is activated and the *CheckinProc* function is called. When the callbacks for the *CDOCS Checkin* screen are initialized in the *LoadTDOCSCheckin* function, the *OK* and *Close* push-buttons are both set to call the *CheckinProc* function, which examines the callback tag. If the callback tag is *Checkin Cancel Tag* the dialog is closed.

2.2.12 Main Menu—Report Entry

The availability of reports depends on the users authorities and permissions. Reports that are authorized for viewing appear on the *Report* pull-down menu.

2.2.12.1 Report Pull-down Menu

The *Report* entry is selected from the menu bar of the *CDOCS Main Menu* and the *Report* pull-down menu appears (Figure 2-34).

Database Content (CDS, CDM, Open Item) Cascading Menu—Functionality

When the *Database Content (CDS, CDM, Open Item)* entry is selected from the *Report* pull-down menu, the *Database Content* cascading menu is displayed. For further information on the functionality and processing of entries in the *Database Content* cascading menu, see Section 2.2.12.3.

Database Reports—Functionality

Production of database reports is supported by the generalized report writer. This facility permits selection of a wide variety of information and report formatting. Once the information has been extracted from the database, it is formatted according to the selected report specifications and presented as a WordPerfect document. The WordPerfect document may be printed, edited, saved, or incorporated into other work products.

The *Database Reports* entry is selected from the *Report* pull-down menu, and the *Database Reports Selection List* screen appears, including a list view with the names of all available database reports. For further information on the selection and display of database reports, see section 2.2.12.5.

Database Reports—Processing

When the *Database Reports* entry is selected from the *Report* pull-down menu, the *_intern_MenuReportWriter* event is recognized by the main event loop. This causes the *MenuReportWriter* function to be executed.

Reference Reports—Processing

The *Reference Reports* entry is selected from the *CDOCS Report* pull-down menu, and a list of available reference reports is displayed. The *Reference Report* contains listings of two NRC bibliographic databases: the NIST Citations and Hydrologic Database References. The desired report is selected from the list of reference reports, and the selected report is displayed. The *Print* push-button is selected to print the report.

When the *Reference Reports* entry is selected from the *Report* pull-down menu, the *_intern_MenuReportReference* event is recognized by the main event loop. This causes the *RefReport* function to be executed. The *RefReport* function opens the *refReportDialog* to display the list of available reference reports. The *RPCOpenRefRptListCursor* function is called to open the cursor to obtain a list of available reports. If errors are detected the function terminates. If a current list of reports exists, the

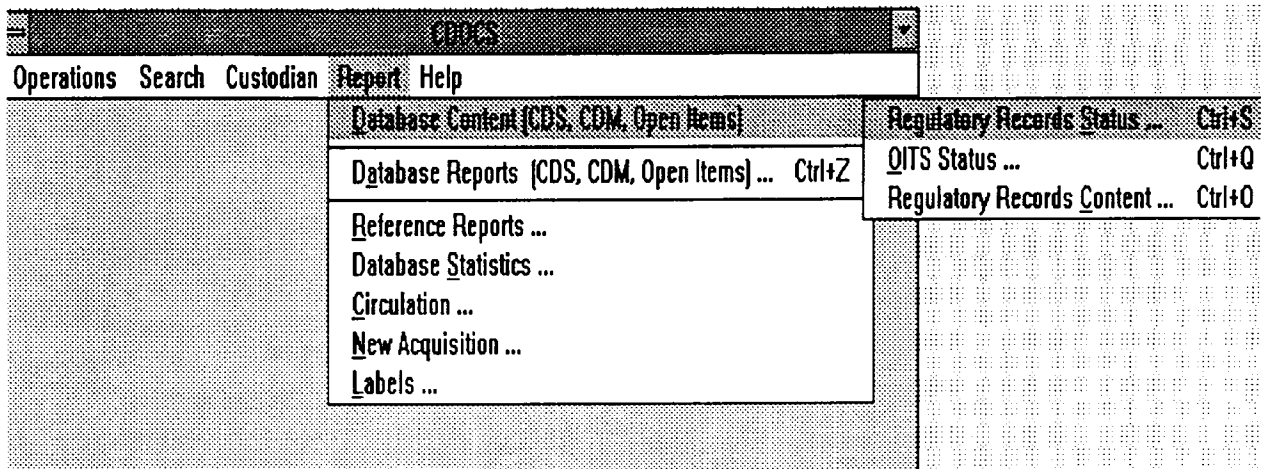


Figure 2-34. Report pull-down menu

RPCOpenRefRptListCursor function is called to delete the entries in the list. The *RPCFetchRefRptListRow* function is called to retrieve each report name. When all report names have been retrieved, the *RPCCloseRefRptListCursor* function is called to close the cursor and the function terminates.

Reference Reports Selection Screen—OK Push-button Processing

When the *OK* push-button is selected from the *Reference Reports* selection screen, the *ReferenceReportOKButton* callback is activated and the *RefReportProc* function is called. If the callback tag is *ReferenceReportOKButton* the list view of reports is examined. If there are no entries, an error message, “No Reports Available”, is displayed. If no entries in the list view are selected, and error message, “You MUST Select a Report to View”, is displayed. The title of the selected report is retrieved from the list view and passed to the *RPCGetReferenceReport* function to retrieve the report, and the *Reference Report* selection screen dialog is closed. If no errors are detected by the *RPCGetReferenceReport* function, the *loadreport* function is called with a parameter of *TDOCS_REF_RPT_REQUEST* to display the report.

Reference Reports Selection Screen—Close Push-button Processing

When the *Close* push-button is selected from the *Reference Reports* selection screen, the *ReferenceReportCloseButton* callback is activated and the *RefReportProc* function is called. The

RefReportProc function examines the callback tag, and the *Reference Reports* selection list dialog is closed.

Database Statistics—Functionality

The Database Statistics Report contains summary listings of the content of CDOCS by numbers of type of document and month/year of submittal. The *Database Statistics* entry is selected from the CDOCS *Report* pull-down menu, and the database statistics report is displayed. The *Print* push-button is selected to print the report.

Database Statistics—Processing

When the *Database Statistics* entry is selected from the *Report* pull-down menu, the *_intern_MenuReportDBStats* event is recognized by the main event loop. This causes the *loadreport* function to be executed with a parameter of *TDOCS_STATS_RPT_REQUEST*. For a further discussion of the *loadreport* function and the processing for report display and printing, see Section 2.2.12.4.

Circulation Report—Functionality

The CDOCS circulation reports comprise a suite of related reports that assist in locating physical documents. The specific circulation reports are accessed through the *CDOCS Circulation Reports* screen (Figure 2-35). The *Circulation* entry is selected from the *Report* pull-down menu, and the *CDOCS Circulation Reports* selection screen is displayed. The radio button for the desired type of report is selected. If the radio button for *Who has a certain document* was selected, the document number is entered in the corresponding entry field. If the radio button for *Retrieve all documents checked out by a certain person* was selected, the person's name is clicked in the selection list, or entered in the *Other Name* field. The *OK* push-button is selected to execute the report. The *Print* push-button is selected to print the report. The *Close* push-button may be selected to exit from the *CDOCS Circulation Reports* selection screen.

Circulation Report — Processing

When the *Circulation* entry is selected from the *Report* pull-down menu, the *_intern_tdocsCirculation* event is recognized by the main event loop. This causes the *GetReportFileName* function to be executed with a parameter of 2. The *GetReportFileName* function examines the passed parameter and if the *currentReportNumber* variable is 2 the *Circulation* function is called. The *BuildDocSet* function is called to create a drop-down list of document sets. Then the *circulationDialog* dialog is opened.

CDOCS Circulation Reports — OK Push-button Processing

When the *OK* push-button is selected from the *CDOCS Circulation Reports* selection screen, the *CirculationOKButton* callback is activated and the *CirculationProc* function is called. The *CirculationProc* function examines the callback tag, and retrieves the document set from the dialog. The *GetWhichCirculationReport* function is called to retrieve the type of report requested.

If the *CirculationCurrentBtn* is selected, *whichReport* is set to 1. If the *CirculationWhoBtn* is selected, *whichReport* is set to 2 and the name entered in the associated field is stripped of white space

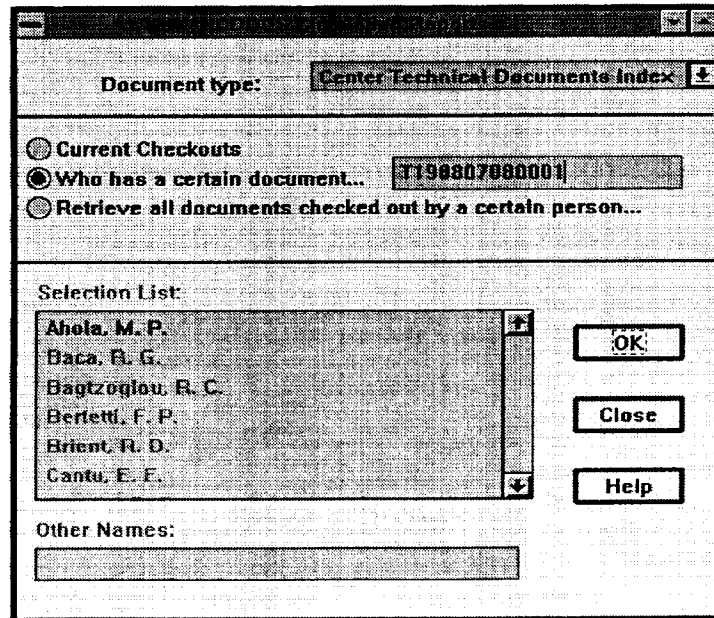


Figure 2-35. Circulation reports selection screen

and stored in *whoWhat*. If the *CirculationWhatBtn* is selected, *whichReport* is set to 3 and the associated selected or entered name is stripped of white space and stored in *whoWhat*, and if no name was selected or entered, an error code is returned.

If the value of *whoWhat* is 2, the *CirculationProc* function calls the *CheckTDOCSDocNum* function to validate the document number. If the value of *whoWhat* is 3, the *CirculationProc* function calls the *CheckTDOCSUserName* function to validate the entered or selected name. If no errors are detected, the *RPCGetCirculationReport* function is called to generate the requested report. The *circulationDialog* dialog is closed, and the title of the *Save As* screen is set to “CDOCS Circulation Report.”

CDOCS Circulation Reports—Close Push-button Processing

When the *Close* push-button is selected from the *CDOCS Circulation Reports* selection screen, the *CirculationCancelButton* callback is activated and the *CirculationProc* function is called. The *CirculationProc* function examines the callback tag, and the *CDOCS Circulation Reports* selection list dialog is closed.

2.2.12.2 New Acquisitions—Functionality

The CDOCS New Acquisitions Report is designed to assist in tracking new acquisitions in the CDOCS system. The new acquisitions report is accessed through the *CDOCS New Acquisitions Report* selection screen. The *New Acquisitions* entry is selected from the *Report* pull-down menu, and the *CDOCS New Acquisitions Report* selection screen is displayed (Figure 2-36).

The *Document Set* drop-down indicator is selected to display a list of available document sets and the desired entry is selected. The beginning date for the reporting period is entered in the *From Date* entry field (Figure 2-37). The ending date for the reporting period is entered in the *To Date* entry field. The *OK* push-button is selected to execute the report, and the *New Acquisitions Report* screen is displayed

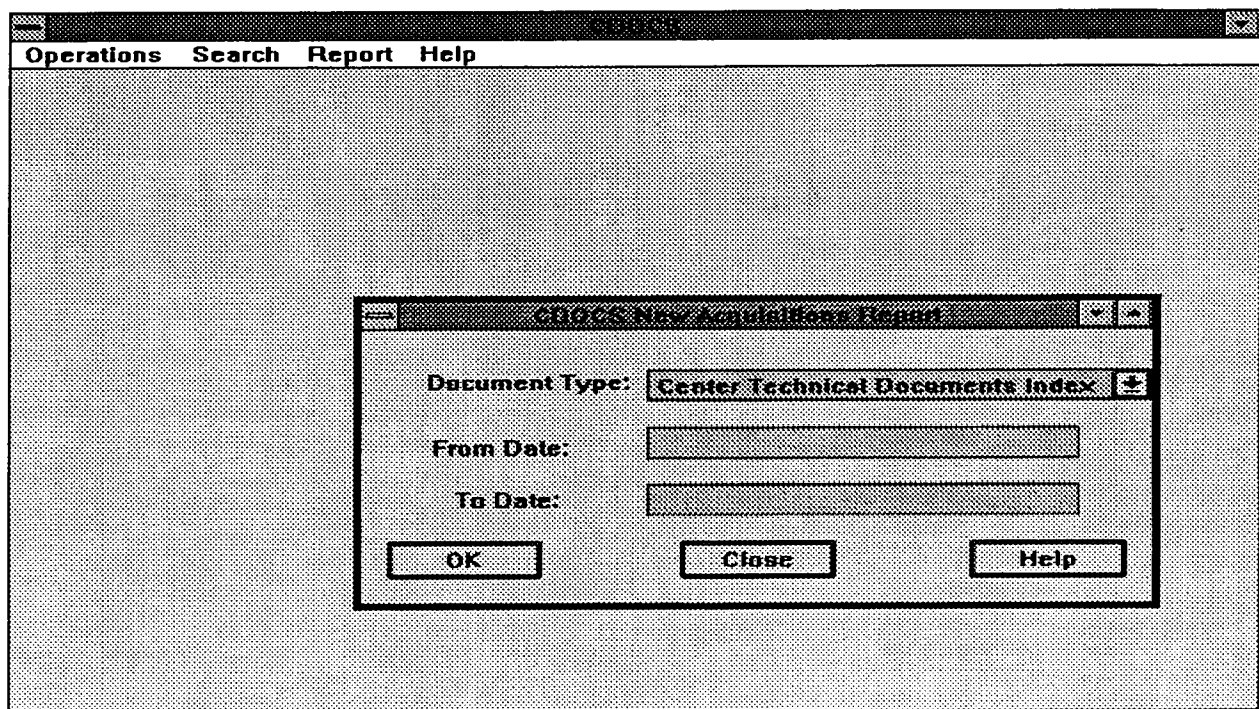


Figure 2-36. New Acquisitions Report selection screen

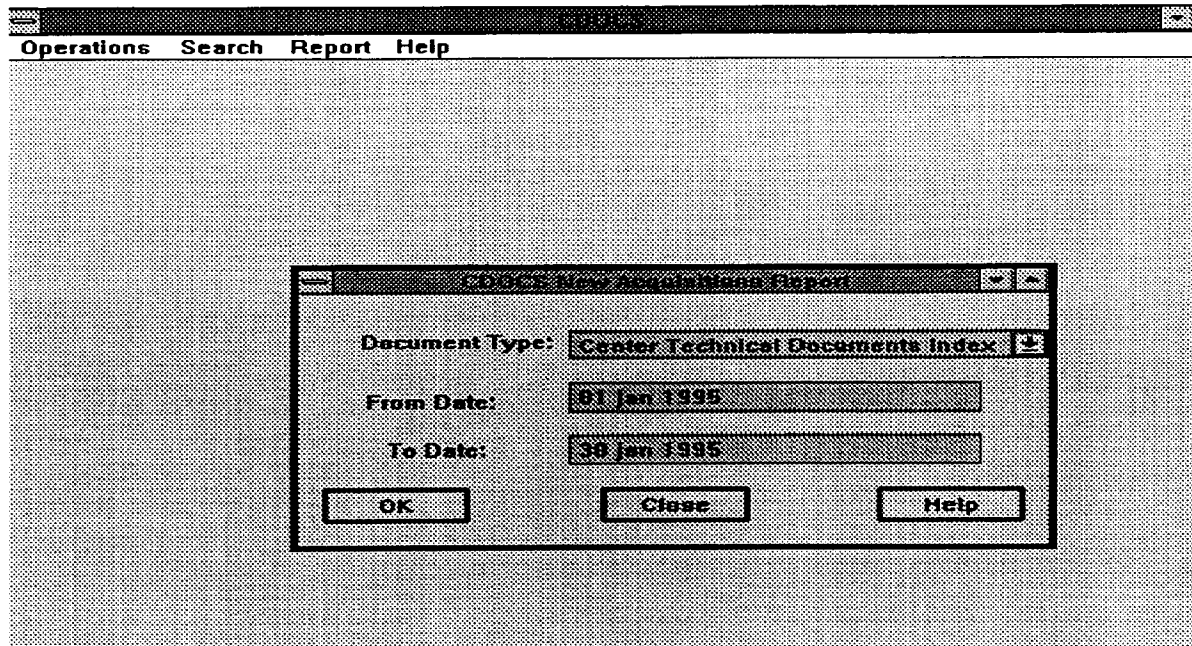


Figure 2-37. New Acquisitions Report selection example

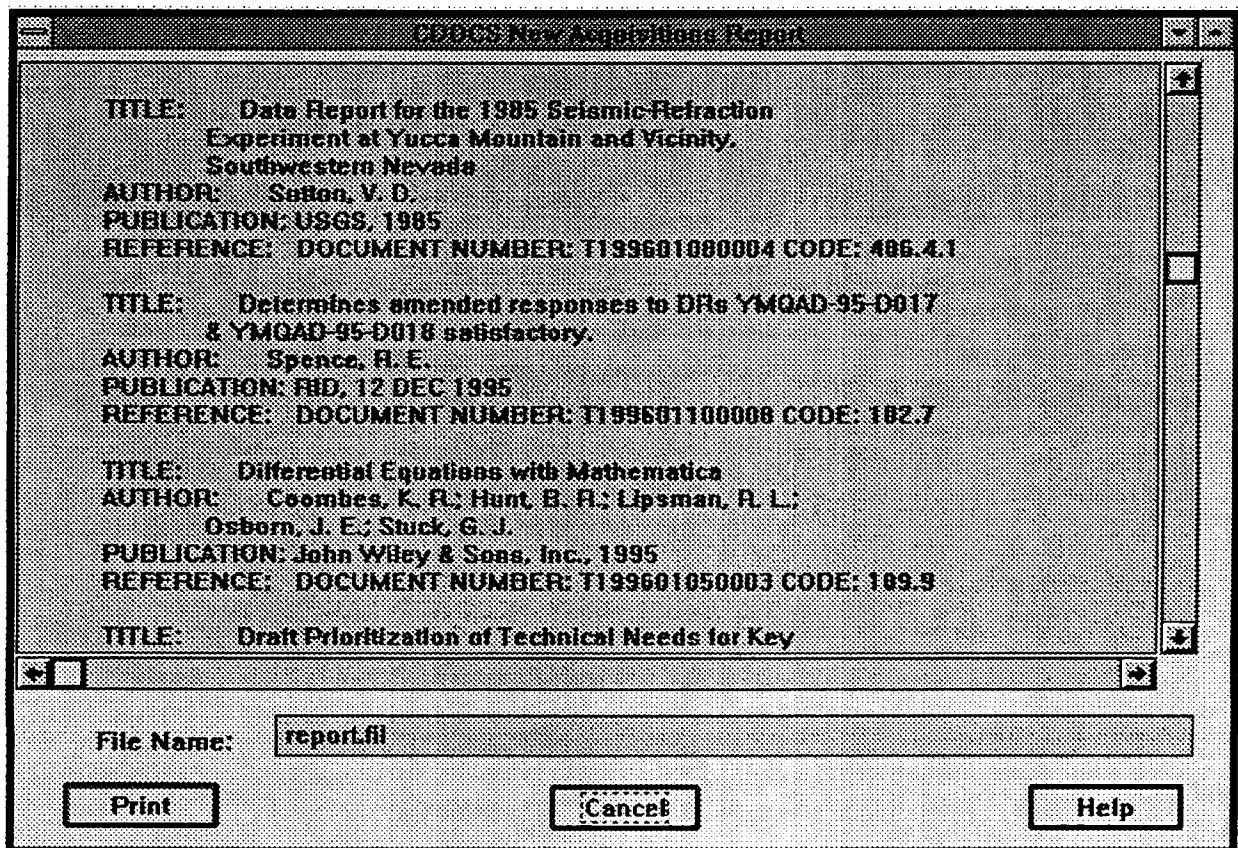


Figure 2-38. New acquisitions report screen

(Figure 2-38). The *Print* push-button is selected to print the report. The *Close* push-button may be to exit from the *CDOCS New Acquisition Report* selection screen.

New Acquisitions—Processing

When the *New Acquisitions* entry is selected from the *Report* pull-down menu, the `_intern_tdocsNewDocs` event is recognized by the main event loop. This causes the `GetReportFileName` function to be executed with a parameter of 3. The `GetReportFileName` function examines the passed parameter and if the `currentReportNumber` variable is 3 the `NewDocs` function is called. The `BuildDocSet` function is called to create a drop-down list of document sets. Then the `newDocsDialog` dialog is opened.

New Acquisitions Reports—OK Push-button Processing

When the *OK* push-button is selected from the *New Acquisitions Reports* selection screen, the `NewDocs OK Button` callback is activated and the `NewDocsProc` function is called. The `NewDocsProc` function examines the callback tag, and retrieves the “from” and “to” dates from the dialog. The `CheckTDOCSDate` function is called to validate the “from” and “to” dates. If the dates are valid, the

RPCGetNewDocsReport function is called to generate the report. The *newDocsDialog* dialog is closed, and the title of the *Save As* screen is set to “CDOCS New Acquisitions Report.” The *showSaveAsText* function is called to display the report.

New Acquisitions Reports—Close Push-button Processing

When the *Close* push-button is selected from the *New Acquisitions Reports* selection screen, the *NewDocs Cancel Button* callback is activated and the *NewDocsProc* function is called. The *NewDocsProc* function examines the callback tag, and the *New Acquisitions Reports* selection list dialog is closed.

Labels—Functionality

The CDOCS system has the capability to produce labels that are affixed to hard copy documents (books, reports, etc.) to assist in document tracking. The *Labels* entry is selected from the *Report* pull-down menu, and the *Labels Report* selection screen is displayed. The *Print* push-button is selected to print the labels.

Labels — Processing

When the *Labels* entry is selected from the *Report* pull-down menu, the *_intern_tdocsLabels* event is recognized by the main event loop. This causes the *GetReportFileName* function to be executed with a parameter of 0. The *GetReportFileName* function examines the passed parameter and if the *currentReportNumber* variable is 0 the *GetLabel* function is called, and the *RPCGetLabels* function is called to generate the current labels. The *showSaveAsText* function is called to display and print the labels.

2.2.12.3 Database Content (CDS, CDM, Open Items) Cascading Menu

The *Database Content* entry is selected from the *Report* pull-down menu, and a cascading menu is displayed that includes the names of the available *Regulatory Records* reports in the *Database Content* group (Figure 2-39).

Regulatory Records Status Report—Processing

When the *Regulatory Records Status* entry is selected from the *Database Content (CDS, CDM, Open Items)* cascading menu, the *_intern_MenuReportStatus* event is recognized by the main event loop. This causes the *loadreport* function to be executed with a parameter of *RPD_RP_STATUS_RPT_REQUEST*. For a further discussion of the *loadreport* function and the processing for report display and printing, see Section 2.2.12.4.

Regulatory Records Content Report—Processing

When the *Regulatory Records Content* entry is selected from the *Database Content (CDS, CDM, Open Items)* cascading menu, the *_intern_MenuReportContent* event is recognized by the main event loop. This causes the *loadreport* function to be executed with a parameter of *RPD_RP_CONTENT_RPT_REQUEST*. For a further discussion of the *loadreport* function and the processing for report display and printing, see Section 2.2.12.4.

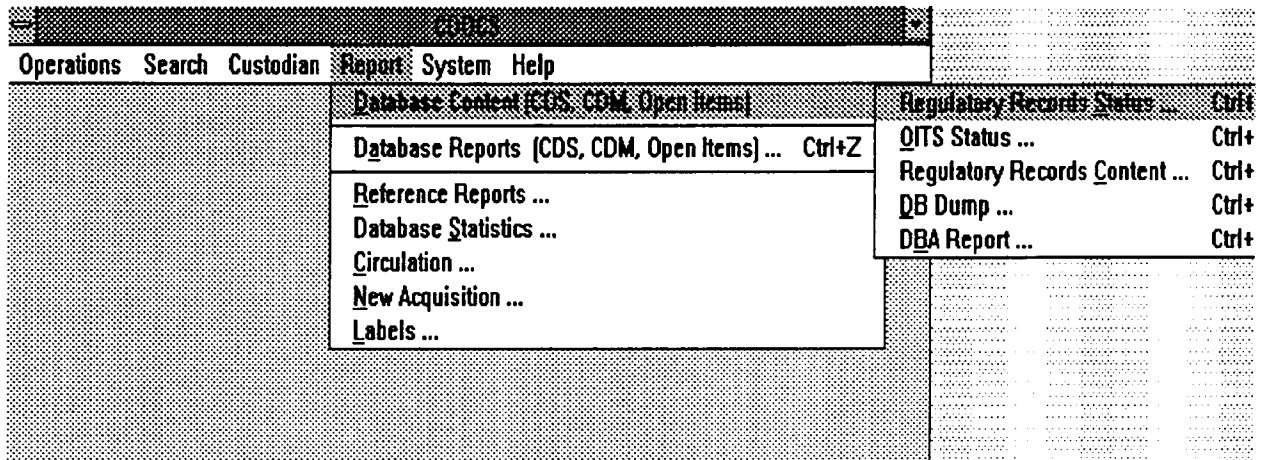


Figure 2-39. Database content cascading menu

OITS Status Report—Processing

When the *OITS Status* entry is selected from the *Database Content (CDS, CDM, Open Items)* cascading menu, the *_intern_ReportOITSSStatus* event is recognized by the main event loop. This causes the *loadreport* function to be executed with a parameter of *RPD_OI_STATUS_RPT_REQUEST*. For a further discussion of the *loadreport* function and the processing for report display and printing, see Section 2.2.12.4.

Database Dump—Processing

When the *DB Dump* entry is selected from the *Database Content (CDS, CDM, Open Items)* cascading menu, the *_intern_MenuReportDump* event is recognized by the main event loop. This causes the *loadreport* function to be executed with a parameter of *RPD_DBA_RPT_REQUEST*. For a further discussion of the *loadreport* function and the processing for report display and printing, see Section 2.2.12.4.

DBA Report—Processing

When the *DBA Report* entry is selected from the *Database Content (CDS, CDM, Open Items)* cascading menu, the *_intern_MenuReportDBA* event is recognized by the main event loop. This causes the *loadreport* function to be executed with a parameter of *RPD_SHORT_DBA_RPT_REQUEST*. For a further discussion of the *loadreport* function and the processing for report display and printing, see Section 2.2.12.4.

2.2.12.4 CDOCS Report Display

The desired report is selected from the *Database Content* reports cascading menu, and the report is prepared and displayed on the screen (Figure 2-40). If printed output is desired, the *Print* push-button at the bottom of the *Report Display* screen is selected, and the report display is formatted and routed to a printer. The *Report Display* screen may be closed by selecting the *Close* push-button at the bottom of the screen.

CDOCS Report Display—Processing

Many of the CDOCS reports utilize the *loadreport* function to retrieve and display the requested information. The *loadreport* function accepts a parameter that indicates the type of report. the *getRptFileName* function is called with a parameter of *NONLABELS* to obtain the standard file name for the report. The *intern_ReportDialog* is opened and a call back is established for the *Help* push-button. Platform dependent code is included to set the appropriate font for the display of the report.

If the report type parameter is *TDOCS_REF_RPT_REQUEST*, the *openDocument* function is called to open the report, and the *SetRefReportTitle* function is called to format the report title. Then the *reportdialog* is opened to display the report.

If the report type parameter is *TDOCS_STATS_RPT_REQUEST*, the *RPCGetStatisticsReport* function is called to open the database statistics report. If the report type parameter is not *TDOCS_STATS_RPT_REQUEST*, the *RPCReport* function is called to open the report. If no errors are detected in the report retrieval, the *openDocument* function is called to open the report, and the *SetRefReportTitle* function is called to format the report title. Then the *reportdialog* is opened to display the report.

CDOCS Report Display—Print Push-button Processing

When the *Print* push-button is selected from the *CDOCS Reports* display screen, the *_intern_TDOCSSaveAsPrint* callback is activated and the *printreport* function is called. The *printreport* function contains platform dependent code for each platform.

CDOCS Report Display—Close Push-button Processing

When the *Close* push-button is selected from the *Reference Reports* selection screen, the *_intern_ReportCloseButton* callback is activated and the *reportdialog* dialog is closed.

2.2.12.5 Selecting, Displaying, and Printing Database Reports

The *Database Reports* entry is selected from the *Report* pull-down menu, and the *Database Reports Selection List* screen appears, including a list view with the names of all available database reports. The desired report is selected from the list of available database reports, and the *View* push-button at the bottom of the screen is clicked. The system formats the requested report as a WordPerfect document and start the WordPerfect software. The *Close* push-button at the bottom of the screen may be selected to exit from the *Database Reports Selection List* screen.

Regulatory Records Status Report					
REGULATORY PROGRAM RECORD STATUS REPORT AS OF 09 JAN 1995					
REV PLAN	TITLE OF DOCUMENT	TYPE	VERSION	DATE	
1.1	General Description of the Facility	CDM	0.0	01 JUL 94	V2
1.1	GENERAL DESCRIPTION OF THE FACILITY	CDS	0.2	01 JUL 94	AC
1.1	General Description of the Facility	RPS	0.0	06 JUL 94	AC
1.2	Basis for Licensing Authority	CDM	0.0	01 JUL 94	V2
1.2	BASIS FOR LICENSING AUTHORITY	CDS	0.2	01 JUL 94	AC
1.2	Basis for Licensing Authority	RPS	0.0	06 JUL 94	AC
1.3	Schedules	CDM	0.0	01 JUL 94	V2
1.3	SCHEDULES	CDS	0.2	01 JUL 94	AC
1.3	Schedules	RPS	0.0	06 JUL 94	AC

Figure 2-40. Regulatory records status report

Database Reports—Processing

When the *Database Reports* entry is selected from the *Report* pull-down menu, the *_intern_MenuReportWriter* event is recognized by the main event loop. This causes the *MenuReportWriter* function to be executed. The *MenuReportWriter* function opens the *menuReportWriterDialog* dialog and calls the *RPCOpenRptListCursor* function to open the cursor for retrieving the available reports. If errors are detected when opening the cursor, the function terminates. If entries exist in the database reports list view, they are deleted. Then the function calls *RPCFetchRptListRow* for each available report and copies the entry to the list view. When all rows have been copied the cursor is closed.

Database Reports Selection List—View Push-button Processing

When the *View* push-button is selected from the *Database Reports Selection List* screen, the *DatabaseReportViewButton* callback is activated and the *MenuReportWriterProc* function is called. The tag is examined, and the list view is examined to obtain the name of the selected report. If there are no entries in the list view, a message is displayed, "No Reports Available." If no entries are selected from

the list view, a message is displayed, “You MUST Select a Report to View.” The *getRptFileName* function is called to obtain the standard file name for the report, and the *RPCReportWriter* function is called to retrieve the report. The *menuReportWriterDialog* dialog is closed and the *showDatabaseReport* function is called to display the report.

Database Reports Selection List—Subset Push-button Processing

When the *Subset* push-button is selected from the *Database Reports Selection List* screen, the *DatabaseReportsSubsetButton* callback is activated and the *MenuReportWriterProc* function is called. The tag is examined, and the list view is examined to obtain the name of the selected report. If no entries are selected from the list view, a message is displayed, “You MUST Select a Report to View.” If the title of the selected report contains the string “Item” a parameter is stored with the value “OITS”. Otherwise, the parameter is stored with the value “CDS.” The *loadSubsetReport* function is called to start the reportwriter and generate the requested report.

Database Reports Selection List—Close Push-button Processing

When the *Close* push-button is selected from the *Database Reports Selection List* screen, the *DatabaseReportCloseButton* callback is activated and the *MenuReportWriterProc* function is called. The tag is examined, and the *menuReportWriterDialog* dialog is closed.

2.2.13 Main Menu—System Entry

The *System* main menu entry permits the database administrator to perform maintenance on the list of authorized users of the system. This facility permits users to be added, deleted, or changed.

2.2.13.1 System Menu Processing

When *User IDs* entry is selected from the *System* pull-down menu, the *_intern_MenuUserIDs* event is recognized by the main event loop. This causes the *loadUserIDs* function to be executed. A callback is established for the *Help* push-button. The *loadPrivilegeLevel* function is called to retrieve entries the privilege level drop down list. If the *useridlistview* is present the *DeleteListEntries* function is called to clear the list view. The *RPCOpenUserListCursor* function is called to open the cursor for retrieval of the user information, and the *RPCFetchUserListRow* function is called to retrieve each row. When all rows have been loaded into the list view, the *RPCCloseUserListCursor* function is called to close the cursor. The *systemuseridsdiallog* dialog is opened and the *SetUpUserIDPassword* function is called to set up the edit filter for the password field.

User ID Maintenance Screen—User ID List View Processing

When the *User ID* list view is selected from the *User Id Maintenance* screen, the *intern_SystemUserIDsListView* callback is activated and the *selectIDChange* function is called. The selected entry in the list view is retrieved and the *Username*, *User ID*, and *Privilege* fields are refreshed.

User ID Maintenance Screen—Close Push-button Processing

When the *Close* push-button is selected from the *User Id Maintenance* screen, the *_intern_SystemUserIDsCloseButton* event is recognized by the main event loop, and the *systemuserid dialog* is closed.

User ID Maintenance Screen—Add Push-button Functionality

The *System* entry is selected from the menu bar of the *CDOCS Main Menu*. The system displays a pull-down menu that contains the *User IDs* entry (Figure 2-41). The *User IDs* entry in the *System* pull-down menu is selected, and the *CDOCS User ID Maintenance* screen (Figure 2-42) appears. The user identification number is entered in the *User ID* entry field. The password is entered in the *Password* entry field. The password that is entered in this field is the initial password for the user.

The name of the user is entered in the *Username* entry field, and the desired privilege from the *Privilege* drop-down list. When all of the information is correct, the *Add* push-button at the bottom of the *User ID Maintenance* screen is selected to add the new *User ID*. The *Close* push-button at the bottom of the screen is selected to exit from the *CDOCS User ID Maintenance* screen.

User ID Maintenance Screen—Add Push-Button Processing

When the *Add* push-button is selected from the *User Id Maintenance* screen, the *_intern_SystemUserIDsAddButton* event is recognized by the main event loop, and the *addUserID* function is called. The *User ID* is retrieved from the dialog and the extraneous white space is stripped. The *Password* is retrieved from the dialog. The *Username* is retrieved from the dialog and the extraneous white space is stripped. If the *User ID*, *Password*, or *Username* is missing, an appropriate error message is displayed. If no errors are detected, the *vRPCAddUser* function is called to add the user. If an *SVC_SQL_ERROR* error is detected an error message, "Error: User-ID already exists", is displayed, and the add transaction is terminated. If a different error code is returned, the *checkForSQLErrors* function is called to display the appropriate error message. If no errors are detected, the *User ID* list view is refreshed and an advisory message is displayed indicating that the user has been added.

User ID Maintenance Screen—Delete Push-Button Functionality

The *User IDs* entry is selected the *System* pull-down menu. The *CDOCS User ID Maintenance* screen (Figure 2-39) appears. The *User ID* to be deleted is selected from the scrollable list view of *User IDs and privileges*. When the correct *User ID* has been selected, the *Delete* push-button at the bottom of the *User ID Maintenance* screen is selected to delete the *User ID*. The *Close* push-button at the bottom of the screen is selected to exit from the *CDOCS User ID Maintenance* screen.

User ID Maintenance Screen—Delete Push-button Processing

When the *Delete* push-button is selected from the *User Id Maintenance* screen, the *_intern_SystemUserIDsDeleteButton* event is recognized by the main event loop, and the *deleteUserIDs* function is called. The *User ID* is retrieved from the list view in the dialog and a confirmation message is displayed. The *vRPCDropUser* function is called to delete the user, and the *User ID* list view is refreshed. An advisory message is displayed indicating that the user has been deleted.

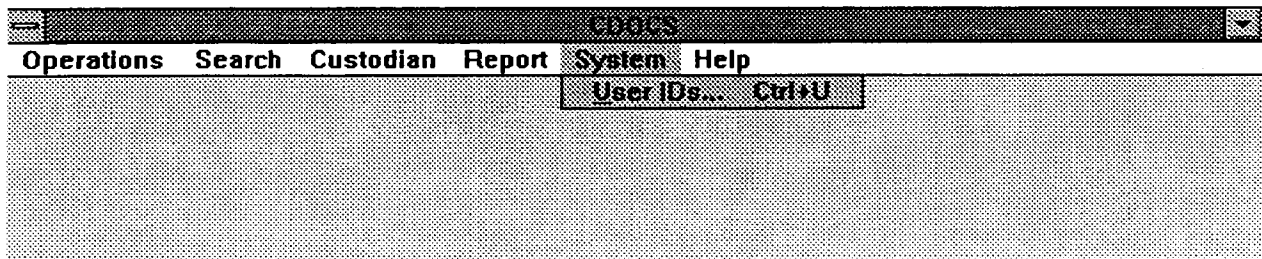


Figure 2-41. CDOCS system pull-down menu

User ID Maintenance Screen—Change Push-button Functionality

The *User IDs* entry is selected from the *System* pull-down menu and the *CDOCS User ID Maintenance* screen (Figure 2-42) appears. The *User ID* of the user record to be changed is selected from the scrollable list of *User IDs*. The *Privilege* push-button is selected to view the *Privilege* drop-down list. The *Change* push-button at the bottom of the *User ID Maintenance* screen is selected to change the user's privileges. The *Close* push-button at the bottom of the screen may be selected to exit from the *CDOCS User ID Maintenance* screen.

User ID Maintenance Screen—Change Push-button Processing

When the *Change* push-button is selected from the *User Id Maintenance* screen, the `_intern_SystemUserIDsChangeButton` event is recognized by the main event loop, and the `changeUserIDs` function is called. The *User ID*, *Password*, and *Username* are retrieved from the dialog. If the *User ID*, *Password*, or *Username* is missing, an appropriate error message is displayed. The `vRPCDropUser` function is called to delete the user, and then the `vRPCAddUser` function is called to add the revised user record. An advisory message is displayed indicating that the user has been changed and the list view of users is updated.

2.2.14 Main Menu—Help Entry

CDOCS provides on-line help that is configured to reflect the individual user's authorities and permissions. The *Help* entry is selected from the menu bar of the CDOCS Main Menu, and the *Help* pull-down menu appears with help entries for the user (Figure 2-43).

2.2.14.1 Help Screen Functionality

When the *Help* is selected and a screen appears that provides information about how to use the Help facility.

User ID	PrivilegeUsername
abagtzo	10 Bagtzotou, A. C.
achowdh	10 Chowdhury, Asadul H.
acortin	3 Alice Cortinas
adewisp	30 DeWispelare, Aaron R.
aghoash	10 Ghosh, Amitava
alopez	10 Lopez, Anna G.
aramos	10 Ramos, Arturo
bcaudie	10 Caudie, Bonnie
bgarcia	10 Garcia, Bonnie L.

Username: User ID:
 Password: Privilege:

Figure 2-42. CDOCS User ID maintenance screen

Help About Screen—Functionality

If the *About* entry is selected, a description of the CDOCS is displayed.

Help About Screen—Processing

When the *About* entry in the *Help* pull-down menu is selected, the `_intern_MenuHelpAbout` event is recognized by the main event loop, and the *aboutdialog* dialog is opened.

Help About Screen—Close Push-button Processing

When the *Close* push-button is selected from the *Help About* screen, the `_intern_AboutCloseButton` event is recognized by the main event loop, and the *aboutdialog* dialog is closed.

Help Index—Functionality

If the *Help* entry is selected from the *Help* pull-down menu, the *Help Index* window is displayed (Figure 2-44).

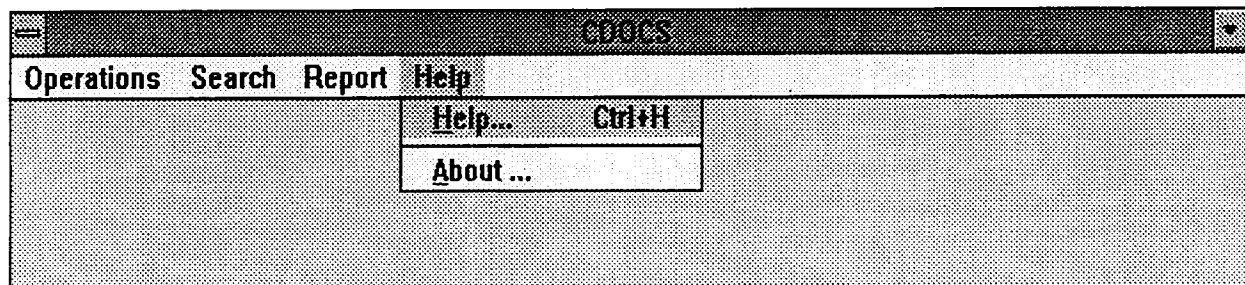


Figure 2-43. CDOCS help pull-down menu

Help Index—Processing

When the *Help* entry in the *Help* pull-down menu is selected, the *_intern_MenuHelpHelp* event is recognized by the main event loop, and the *BuildHelpIndex* function is executed. The *RPCOpenHelpListCursor* function is called to open the cursor of help topics. If errors are detected, the *BuildHelpIndex* terminates. If entries exist in the help index, they are deleted. The *RPCFetchHelpListRow* function is called for each row to retrieve the entry for the help topic, and the header, key, and subkey of each row is stored in the listview. When all rows have been retrieved, the *RPCCloseHelpListCursor* function is called to close the cursor and the listview is sorted.

Help Index Screen—OK Push-button Processing

When the *OK* push-button is selected from the *Help Index* screen, the *HelpIndexOK* callback is activated, and the *HelpIndexProc* function is executed. The tag is examined and the selected topic is retrieved from the list view in the dialog. If no entry is selected, an error message is displayed, "You MUST Select a Help topic to View." If no errors are detected, the *ShowHelp* function is called and the button tag is passed.

Help Index Screen—Close Push-button Processing

When the *Close* push-button is selected from the *Help Index* screen, the *HelpIndexCancel* callback is activated, and the *HelpIndexProc* function is executed. The tag is examined and the *helpIndexDialog* dialog is closed.

Help Screens—Functionality

The desired entry is selected from the *Help Index* window to display the help topics for that entry (Figure 2-45).

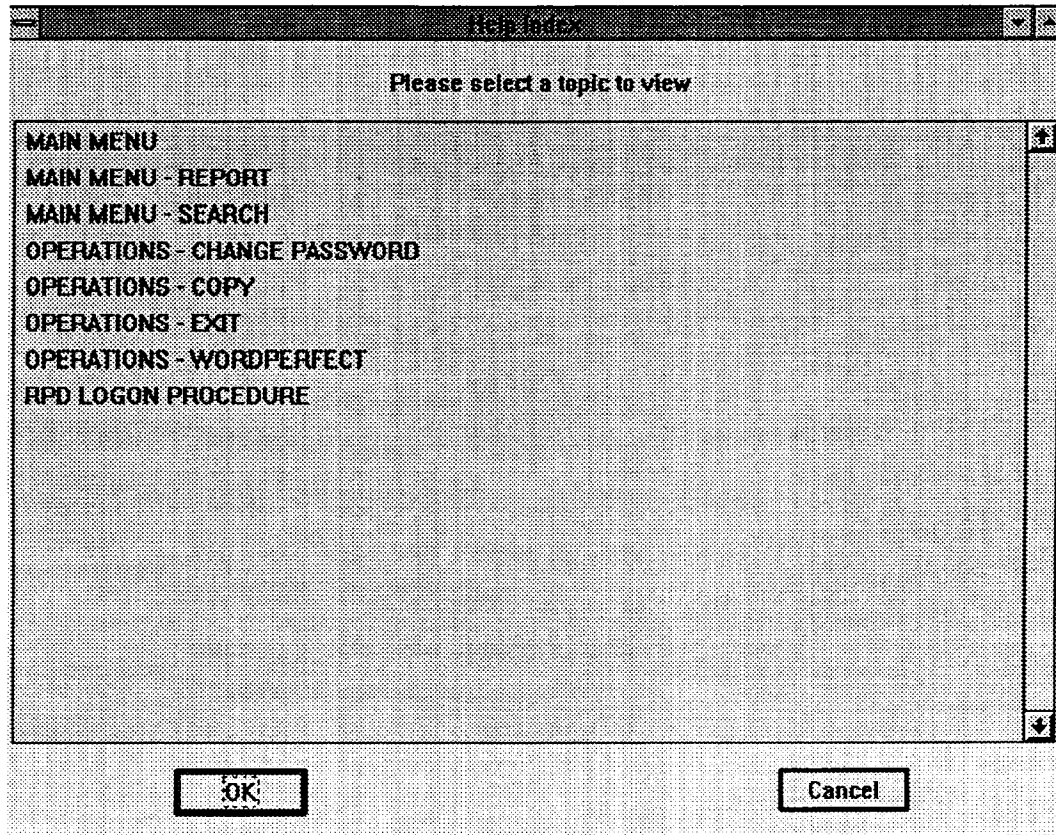


Figure 2-44. CDOCS help index

Help Screen—Processing

Numerous callbacks are set to invoke the help function. When these callbacks are activated the help function retrieves the button tag and uses it as a key to retrieve the help text. The button tag is examined and if it is *LogonHelp* the *ShowHelp* function is called with the button tag and a parameter of 0. Otherwise, the *ShowHelp* function is called with the button tag and a parameter of 1.

The *ShowHelp* function calls the *RPCGetHelp* function to retrieve the help information. If an error code is returned, the *ShowHelp* function calls the *RPCGetHelp* function with a parameter of "NoHelpAvail." The function then sets up the heading of the *helpDialog* dialog, inserts the help text, and opens the dialog.

Help Screen—Close Push-button Processing

When the *Close* push-button is selected from the *Help* screen, the *HelpCancel* callback is activated, and the *HelpProc* function is executed. The tag is examined and the *helpDialog* dialog is closed.

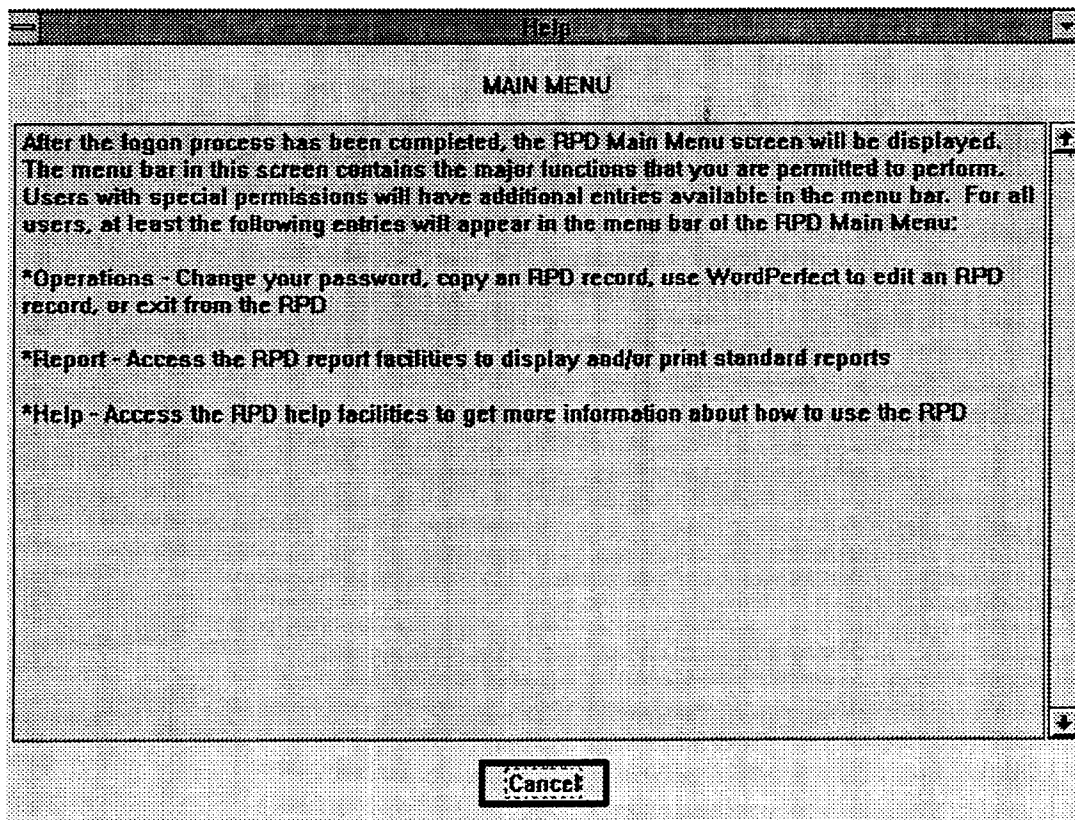


Figure 2-45. CDOCS help screen

2.3 SERVER SYSTEM TECHNICAL DOCUMENTATION

The server consists of two major components that are initialized by the *main* program.

- The Synchronization Process—Handles synchronization between the local and remote servers
- The Main Service Switch—Handles incoming RPC service requests from the client

Note: throughout the server section the expression `\n` is used to signify the hexadecimal digit `x00n` (e.g., `\1` is `x001`).

2.3.1 Main Server Program Initialization

The main server program initializes the server and logs into the ORACLE database. When the initialization is complete, the server waits for RPC calls that are handled by the *rpc_N* function. When a *RPDOCS_EXIT_REQUEST* service request is received by the *rpc_N* function, the server terminates.

The *main* server program calls *SVCSetResLim* to set the file descriptor resource limit to the maximum, and then prints a server startup message. The *SVCCheckArgs* function is called to check the

command line arguments. The *SVCSetSvcVer* function is called to set the server version information, and an e-mail message is formatted and sent if the function fails. The *SVCSetSvcEnv* function is called to set the server environment information, and an e-mail message is formatted and sent if the function fails. The *SVCCheckUID* function is called to verify that the userid is TOPIC, and an e-mail message is formatted and sent if the function fails. The *TDOCSCleanSubmit* is called to clean out the submit directory on start up. The *SVCInitLog* function is called to initialize request and synchronization logging, and an e-mail message is formatted and sent if the function fails. The *SVCRegister* function is called to register the server with the RPC daemon, and an e-mail message is formatted and sent if the function fails. The *SVCLogDBAIn* function is called to log the server into the Oracle database as DBA. If the synchronization mode is *RPDOCS_SYNC_IN*, the *SVCRunSyncIn* function is called to run the TDOCS synchronization. If the synchronization mode is *RPDOCS_SYNC_SYNC*, the *SVCInitSyncIn* function is called to initialize TDOCS synchronization, and an e-mail message is formatted and sent if the function fails. When initialization is complete, a start services message is printed and the program waits for an RPC request. As each RPC request is received, the *rpc_N* function is called to service the request. When a *RPDOCS_EXIT_REQUEST* service request is received by *rpc_N*, the function terminates and returns control to the main server program, which terminates the server.

2.3.2 Initialization of the Synchronization Process

The main server program initialization starts the synchronization process by calling the *SVCInitSyncIn* function. The *SVCInitSyncIn* function starts synchronization by calling the *TDOCSSyncInSetAlarm* function.

The *TDOCSSyncInSetAlarm* function sets a callback to the *TDOCSSyncInAlarm* function, calculates the remaining seconds until the next hour, and activates the alarm. This causes synchronization to wait until the next hour when it is activated by calling the *TDOCSSyncInAlarm* function.

2.3.3 Synchronization Processing

The synchronization process is started by a callback from an alarm that is initially set by the *TDOCSSyncInSetAlarm* function. When the callback is activated each hour, the *TDOCSSyncInAlarm* function is called. Figure 2-46 illustrates the overall synchronization process.

The *TDOCSSyncInAlarm* function is the synchronization alarm handler. It sets up signal handler and forks a process for synchronization of TDOCS and RPD documents in the test or production databases, as appropriate. The *SIGUSR1* signal is set for *TDOCSSyncInSetAlarm*. Then the *TDOCSSyncIn* function is called to perform synchronization input processing for TDOCS documents, and the *RPDSyncIn* function is called to perform synchronization input processing for RPD documents. When the synchronization process completes, the forked process is killed and the *SIGUSR1* signal is activated. This causes the *TDOCSSyncInSetAlarm* function to be executed, which sets a new alarm to call *TDOCSSyncInAlarm* at the end of the next hour.

2.3.3.1 Synchronization of TDOCS Documents

The *TDOCSSyncIn* function performs synchronization input processing for TDOCS documents on a document set by document set basis. The *TDOCSSyncInLogIn* function is called to login to the local database for synchronization, and the *ChangeWorkingDir* function is called to change to the local synchronization directory. The *TDOCSSyncInGetDocSets* function is called to fetch a list of document sets,

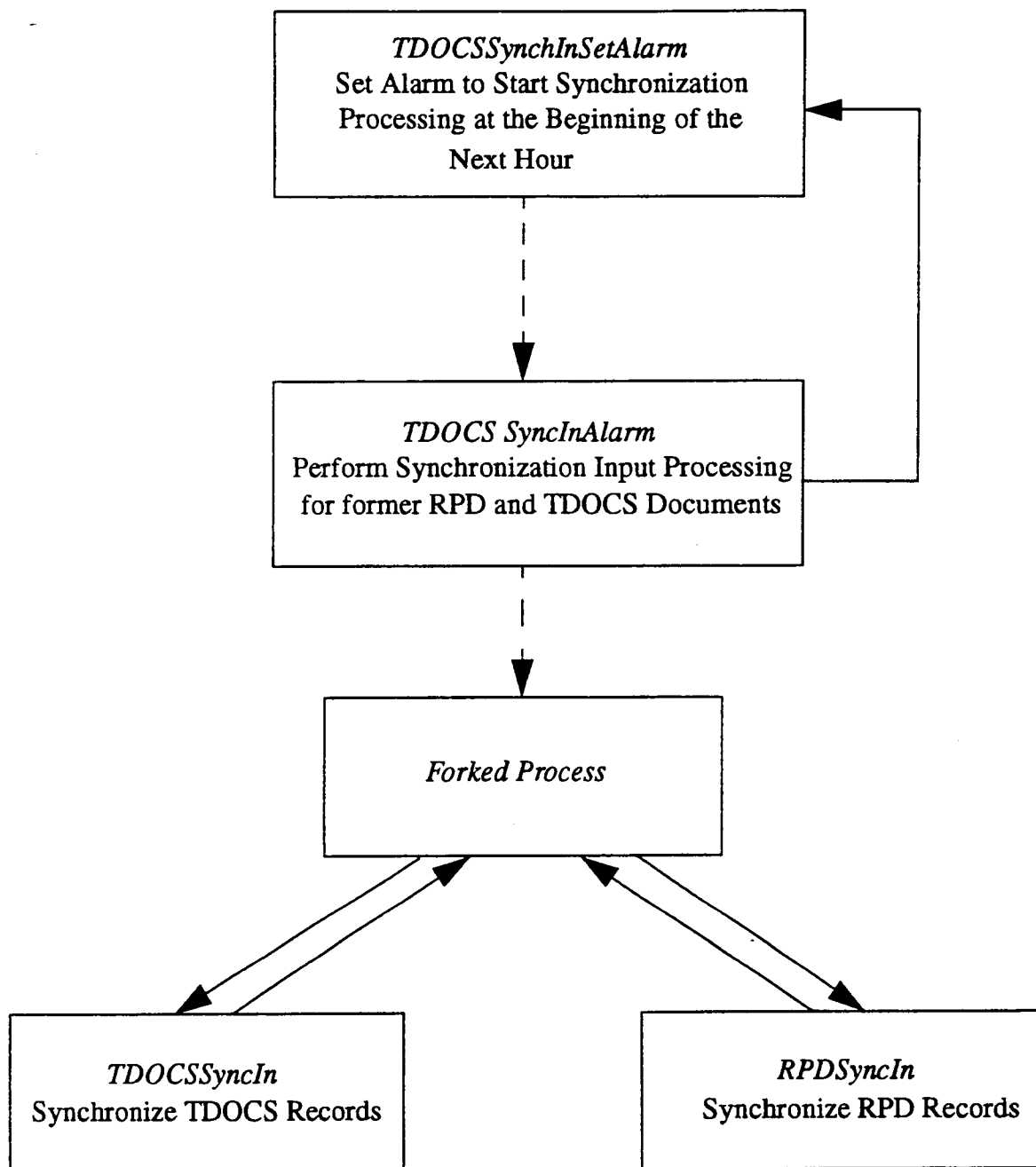


Figure 4-46. Synchronization

converted to lower case, that might require synchronization. The document sets are retrieved from the *TDOCS_DOCUMENT_SETS* table and are stored in a singly linked list. If the *TDOCSSyncInGetDocSets* function fails, *TDOCSSyncInLogOut* is called to log out and the function terminates. The *TDOCSSyncInDocSet* function is called to process successive entries in the document set link list. When all document sets have been processed, the *TDOCSFreeDataList* function is called to free memory and the *TDOCSSyncInLogOut* function is called to log out.

The *TDOCSSyncInDocSet* function processes synchronization input on a document by document basis. For each header file found in the remote out-queue, (i) the header is read, (ii) the document id is modified, (iii) the record is inserted, (iv) the files are renamed to local incoming, and (v) the transaction is committed or rolled back.

2.3.3.2 Synchronization of RPD Documents

The *RPDSyncIn* function performs synchronization input processing for RPD documents on a document set by document set basis. The *TDOCSSyncInLogIn* function is called to login to the local database for synchronization, and the *ChangeWorkingDir* function is called to change to the local synchronization directory. The *TDOCSSyncInGetDocSets* function is called to fetch a list of document sets, converted to lower case, that might require synchronization. The document sets are retrieved from the *TDOCS_DOCUMENT_SETS* table and are stored in a singly linked list. If the *TDOCSSyncInGetDocSets* function fails, the *TDOCSSyncInLogOut* function is called to log out and the function terminates. The *RPDSyncInDocSet* function is called to process successive entries in the document set link list. When all document sets have been processed, the *TDOCSFreeDataList* function is called to free memory and the *TDOCSSyncInLogOut* function is called to log out.

The *TDOCSSyncInDocSet* function processes synchronization input on a document by document basis. For each header file found in the remote out-queue, (i) the header is read, (ii) the document id is modified, (iii) the record is inserted, (iv) the files are renamed to local incoming, and (v) the transaction is committed or rolled back.

2.3.4 Switch Function Call/Data Return Mechanism

In general, function calls receive a request packet of data containing input parameters in the structure named *data*. (That structure is dynamically allocated, and the function called actually receives a pointer “*pPacket*” that points to “*data*”, i.e. *pPacket*→*data*). Error flags are returned as *SVC_CLNT_XXX* errors when client requested data is in error and *SVC_SVC_XXX* when the server cannot complete the request. When a called function successfully completes a list of data needs to be returned to the client, data is returned by the *RPC* Return structure, pointed to by the *pReturn* pointer, (i.e. *pReturn*→*RPC* Return). Lists of data are stored in linked lists at *pReturn*→*RPC* Return.list with the name of the list stored in *pReturn.data* and status and or error codes stored in the *pReturn*→*RPC* Return structure.

2.3.5 Main Service Switch Function

The *rpc_N* function is the main service request switch that maps service request identifiers to service functions. It also frees previous results, logs the user in (where the request is validated and logged), rolls back on failure, logs the user out, and returns the *pRPCReturn* pointer to service results packet. Table 2-5 documents the functions called in response to service requests.

Table 2-5. Service requests and functions.

Service Request	Function	Description
<i>USER_INFO_REQUEST</i>	<i>GetUserInfo</i>	Get User Information The <i>GetUserInfo</i> function gets the privilege and name for a specified userid from the <i>RPDOCS_USERS</i> and <i>RPDOCS_NAMES</i> tables.
<i>USER_NAMES_REQUEST</i>	<i>GetUserNames</i>	Get User List The <i>GetUserNames</i> function gets list of user names, sorted by name, from the <i>RPDOCS_NAMES</i> table.
<i>USER_LIST_REQUEST</i>	<i>GetUserList</i>	Get User List The <i>GetUserList</i> function gets a list of userids, privileges, and names from the <i>RPDOCS_USERS</i> and <i>RPDOCS_NAMES</i> tables.
<i>CHANGE_PW_REQUEST</i>	<i>ChangePassword</i>	Change Password The <i>ChangePassword</i> function changes the password for a specified user by granting connect privileges for that userid and password.
<i>ADD_USER_REQUEST</i>	<i>AddUser</i>	Add a User The <i>AddUser</i> function adds the user as defined by the specified userid, password, privilege, and name.
<i>DROP_USER_REQUEST</i>	<i>DropUser</i>	Drop a User The <i>DropUser</i> function drops the user specified by the userid parameter, revokes the connect for the userid, and removes the user id.
<i>GET_NAMES_REQUEST</i>	<i>GetNames</i>	Get Names The <i>GetNames</i> function retrieves a list of names from the <i>PEOPLE</i> table.
<i>RPD_GET_DOC_SETS_REQUEST</i>	<i>RPDGetDocSets</i>	Get RPD Document Sets The <i>RPDGetDocSets</i> function gets a list of document sets, sorted by document set name, from the <i>RPD_DOCUMENT_SETS</i> table.
<i>RPD_ACTIVE_DOCLIST_REQUEST</i>	<i>RPDGetActiveDocList</i>	Get a List of Active RPD Documents The <i>RPDGetActiveDocList</i> function gets a list of active documents for a specified document set from the RPD table, sorted by document number. Only the most recent instance of each document is retrieved.

Table 2-5. Service requests and functions. (cont'd)

Service Request	Function	Description
<i>RPD_NONRET_DOCLIST_REQUEST</i>	<i>RPDGetNonRetDocList</i>	Get a List of Non-Retired Documents The <i>RPDGetNonRetDocList</i> function gets a list of documents for a specified document set with status of "VACANT" or "ACTIVE" from the RPD table.
<i>RPD_SUBSET_DOCLIST_REQUEST</i>	<i>RPDGetDocSubsetList</i>	Get a List of Documents for a Subset Report The <i>RPDGetDocSubsetList</i> function gets a list of document numbers and titles for whose pieces are consistent with the intended report query.
<i>RPD_BIN_DOC_GET_REQUEST</i>	<i>RPDGetCDSCDMDoc</i>	Retrieve a CDS, CDM, or RPS Document The <i>RPDGetCDSCDMDoc</i> function retrieves the binary file for the active CDS, CDM, or RPS file specified by the submitted document set and document number.
<i>RPD_DB_STATUS_RPT_REQUEST</i> <i>RPD_DB_CONTENT_RPT_REQUEST</i> <i>RPD_RP_STATUS_RPT_REQUEST</i> <i>RPD_RP_CONTENT_RPT_REQUEST</i> <i>RPD_OI_CONTENT_RPT_REQUEST</i>	<i>RPDDBReport</i>	Print a Status or Content Report The <i>RPDDBReport</i> function generates and returns a status or content report. The report request is examined and used to condition the report headings and the SQL query that retrieves the document records.
<i>RPD_OI_STATUS_RPT_REQUEST</i>	<i>RPDOITSStatusReport</i>	Print the Open Item Status Report The <i>RPDOITSStatusReport</i> function generates an OITS status report and returns it to the client. The report includes OITS ID, topic, last update date, and resolution status.
<i>RPD_SHORT_DBA_RPT_REQUEST</i>	<i>RPDShortDBAReport</i>	Short DBA Report The <i>RPDShortDBAReport</i> function produces a short form DBA report that includes the document type, document id, instance id, document number, and status record data for a specified document type.
<i>RPD_DBA_RPT_REQUEST</i>	<i>RPDDBAReport</i>	DBA Report The <i>RPDDBAReport</i> function produces a DBA report that includes dumps of all record data for all document types.
<i>RPD_DEFINE_REQUEST</i>	<i>RPDDefine</i>	Define an RPD Record The <i>RPDDefine</i> function defines CDS, CDM, RPS, and OITS records.

Table 2-5. Service requests and functions. (cont'd)

Service Request	Function	Description
<i>RPD_FORMAT_CHECK_REQUEST</i>	<i>RPDCheckFormat</i>	Format Check an RPD Record The <i>RPDCheckFormat</i> function checks the format and validates CDS, CDM, or RPS records.
<i>RPD_CHECKIN_REQUEST</i> <i>RPD_FORCE_CHECKIN_REQUEST</i>	<i>RPDCheckin</i>	Enter an RPD Record The <i>RPDCheckin</i> function checks the format, validates, and stores CDS, CDM, or RPS records.
<i>RPD_RETIRE_REQUEST</i>	<i>RPDRetire</i>	Retire an RPD Record The <i>RPDRetire</i> function retires a CDS, CDM, or RPS record.
<i>RPD_REPORT_LIST_REQUEST</i>	<i>RPDGetReportList</i>	Get a List of Report Writer Reports The <i>RPDGetReportList</i> function retrieves a list of report writer reports, titles, types, and helps.
<i>RPD_REPORT_WRITER_REQUEST</i>	<i>RPDReportWriter</i>	Retrieve a Report Writer Report The <i>RPDReportWriter</i> function gets report name from report title, retrieves the report, and returns it.
<i>TDOCS_GET_DOC_SETS_REQUEST</i>	<i>TDOCSGetDocSets</i>	Get a List of Loadable Document Sets The <i>TDOCSGetDocSets</i> function retrieves and returns a list of document sets that are loadable.
<i>TDOCS_DOC_SET_SPEC_REQUEST</i>	<i>TDOCSGetDocSetSpec</i>	Get the Sharable Flag and Specifications for a Document Set The <i>TDOCSGetDocSetSpec</i> function retrieves the specification and sharable flag for a specified document set
<i>TDOCS_SUBMIT_DOC_REQUEST</i>	<i>TDOCSSubmitDocument</i>	Submit a CDOCS Document The <i>TDOCSSubmitDocument</i> function submits a CDOCS document, inserts the document in the database, and stores the files.
<i>TDOCS_GET_RECORD_REQUEST</i>	<i>TDOCSGetRecord</i>	Get a CDOCS Record The <i>TDOCSGetRecord</i> function gets a CDOCS record by document number and formats it for return as a delimited string.
<i>TDOCS_UPDATE_DOC_REQUEST</i>	<i>TDOCSUpdateDocument</i>	Update a CDOCS Record The <i>TDOCSUpdateDocument</i> function updates a CDOCS document by replacement.

Table 2-5. Service requests and functions. (cont'd)

Service Request	Function	Description
<i>TDOCS_DELETE_DOC_REQUEST</i>	<i>TDOCSDeleteDocument</i>	Delete a CDOCS Document The <i>TDOCSDeleteDocument</i> function marks a CDOCS document for deletion. The record is marked but the actual deletion done during Batch processing.
<i>TDOCS_UNDELETE_DOC_REQUEST</i>	<i>TDOCSUndeleteDocument</i>	Undelete a CDOCS Record The <i>TDOCSUndeleteDocument</i> function undeletes a document that has been marked for deletion, but has not been deleted by the Batch process.
<i>TDOCS_LABEL_DOCS_REQUEST</i>	<i>TDOCSLabelDocuments</i>	Print CDOCS Labels The <i>TDOCSLabelDocuments</i> function generates labels for any TDI, QA, and/or CSP document that has not yet been labeled.
<i>TDOCS_RELABEL_DOCS_REQUEST</i>	<i>TDOCSRelabelDocuments</i>	Relabel CDOCS Documents The <i>TDOCSRelabelDocuments</i> function regenerates labels for documents specified in the data list.
<i>TDOCS_CHECKOUT_REQUEST</i> <i>TDOCS_FORCE_CHECKOUT_REQUEST</i>	<i>TDOCSCheckoutDocument</i>	Checkout a CDOCS Document The <i>TDOCSCheckoutDocument</i> function checks out a document, identified by the submitted document number, to a name or location by adding a record to the <i>TDOCS_CIRCULATION</i> table.
<i>TDOCS_CHECKIN_REQUEST</i>	<i>TDOCSCheckinDocument</i>	Check In a CDOCS Document The <i>TDOCSCheckinDocument</i> function checks in a CDOCS document identified by document number in by deleting it from the <i>TDOCS_CIRCULATION</i> table.
<i>TDOCS_CIRC_RPT_REQUEST</i>	<i>TDOCSCirculationReport</i>	CDOCS Circulation Report The <i>TDOCSCirculationReport</i> function generates and returns a circulation report for the specified document set, and optionally an individual checkout name.
<i>TDOCS_NEWDOCS_RPT_REQUEST</i>	<i>TDOCSNewDocsReport</i>	Print the New Acquisitions Report The <i>TDOCSNewDocsReport</i> function generates a report on new CDOCS documents entered from a given date.
<i>TDOCS_REF_RPT_LIST_REQUEST</i>	<i>TDOCSGetRefRptList</i>	Get a List of Reference Reports The <i>TDOCSGetRefRptList</i> function builds and returns a list of CDOCS reference reports.

Table 2-5. Service requests and functions. (cont'd)

Service Request	Function	Description
<i>TDOCS_REF_RPT_REQUEST</i>	<i>TDOCSReferenceReport</i>	Print a CDOCS Reference Report The <i>TDOCSReferenceReport</i> function generates and returns a CDOCS reference report.
<i>TDOCS_STATS_RPT_REQUEST</i>	<i>TDOCSStatisticsReport</i>	Print the CDOCS Statistics Report The <i>TDOCSStatisticsReport</i> function calls the <i>ReadAsciiFile</i> function to retrieve and return the CDOCS statistics report.
<i>SQLDTD_FETCH_RPTS_REQUEST</i>	<i>SQLDTDFetchReports</i>	Get a list of Report Writer Reports The <i>SQLDTDFetchReports</i> function fetches a list of report writer reports including the report id, title, and help.
<i>SQLDTD_SELECT_RPT_REQUEST</i>	<i>SQLDTDSelectReport</i>	Select Specifications for a Specified Report Writer Report The <i>SQLDTDSelectReport</i> function selects and returns the report specifications for a particular report id.
<i>SQLDTD_INSERT_RPT_REQUEST</i>	<i>SQLDTDInsertReport</i>	Insert Specifications for a Report Writer Report The <i>SQLDTDInsertReport</i> function inserts a report specification record for a report writer report.
<i>SQLDTD_UPDATE_RPT_REQUEST</i>	<i>SQLDTDUpdateReport</i>	Update Specifications for a Report Writer Report The <i>SQLDTDUpdateReport</i> function updates a report specification record for a specified report writer report.
<i>SQLDTD_DELETE_RPT_REQUEST</i>	<i>SQLDTDDeleteReport</i>	Delete Specifications for a Report Writer Report The <i>SQLDTDDeleteReport</i> function deletes a report specification record for a specified report writer report.
<i>SQLDTD_QUERY_RPT_REQUEST</i>	<i>SQLDTDQueryReport</i>	Run a Query for a Report Writer Report The <i>SQLDTDQueryReport</i> function runs a SQL query for a report and returns the tag and search text results.
<i>SQLDTD_GET_FILE_REQUEST</i>	<i>SQLDTDGetFile</i>	Retrieve the WordPerfect Macro or Intellitag DTD File for the Report Writer The <i>SQLDTDGetFile</i> function retrieves a WordPerfect macro or an Intellitag DTD file for the Report Writer.

Table 2-5. Service requests and functions. (cont'd)

Service Request	Function	Description
<i>SQLDTD_PUT_FILE_REQUEST</i>	<i>SQLDTPutFile</i>	Store the WordPerfect Macro or Intellitag DTD File for the Report Writer The <i>SQLDTPutFile</i> function stores WP macro (<report>.wpm) or Intellitag DTD (<report>.dtd) file.
<i>RPDOCS_EXIT_REQUEST</i>	<i>rpcData.pReturn->status=SVC_SUCCESS</i>	Terminate the Server
<i>RPDOCS_HELP_REQUEST</i>	<i>GetHelp</i>	Get Help Information The <i>GetHelp</i> function gets help header and text for a specified privilege, button, and subkey.
<i>RPDOCS_HELP_LIST_REQUEST</i>	<i>GetHelpList</i>	Get a List of Help Topics The <i>GetHelpList</i> function gets a list of help button, subkeys, and headers for specified privilege, and writes the output to <i>pReturn</i> as a delimited string.
<i>default</i>	<i>rpcData.pReturn->status=SVC_GEN_ERROR</i>	The <i>SVC_GEN_ERROR</i> message is returned when an unrecognized service request is received.

Get User Information

The *GetUserInfo* function gets the privileges and the name for the userid found in *pPacket->data*, and checks the client version number. If the client version number is not provided, *SVC_CLNT_SYNC_ERROR* is returned. The *GetUserInfo* function expects the input parameters as a labeled delimited string that includes userid and client. The privileges are retrieved from the *RPDOCS_USERS* and *RPDOCS_NAMES* tables, the results are written to *pReturn* data, and *pReturn* status and error are set.

Get User Names

The *GetUserNames* function gets a list of user names, sorted by name, from the *RPDOCS_NAMES* table. The *PutListData* function is called to assemble the list of names. The function writes a label to *pReturn* data, writes the name values to *list*, and sets the *pReturn* status and error.

Get User List

The *GetUserList* function gets a list of userids, privileges, and names from the *RPDOCS_USERS* and *RPDOCS_NAMES* tables. Users with a privilege level of 2 are excluded and the results are sorted by userid. The *PutListData* function is called to assemble the list of userids, privileges, and names. The function writes label to *pReturn* data, the name values to *list*, and sets *pReturn* status and error.

Change Password

The *ChangePassword* function changes the password for a specified user. The *ChangePassword* function expects the input parameters as a labeled delimited string that includes `USER_ID \2 userid \1` `PASSWD \2 passwd`. If the `userid` or `password` is not found in the parameters, a `SVC_DATA_SET_ERROR` is returned. The *ChangePassword* function builds a `GRANT CONNECT` command for the specified `userid` and `password`, executes the command, and returns `SVC_SUCCESS`.

Add a User

The *AddUser* function adds the user, as defined by the specified `userid`, `password`, `privilege`, and `name`. The *AddUser* function expects the input parameters as a labeled delimited string that includes `USER_ID \2 userid \1` `PASSWD \2 passwd \1` `PRIVILEGE \2 privilege \1` `NAME \2 name \1`. The `RPDOCS_USERS` table is checked for the specified `userid`. If the `userid` is already in the table or if the `userid`, `password`, `privilege` or `name` is missing, `SVC_DATA_SET_ERROR` is returned. A `GRANT CONNECT` command is prepared and executed for the specified `userid` and `password`. If the specified `name` exists in the `RPDOCS_NAMES` table, the `name ID` is retrieved. Otherwise the next `name ID` is retrieved. The `name` is inserted in the `RPDOCS_NAMES` table, the user `privilege` is inserted in the `RPDOCS_USERS` table, and `SVC_SUCCESS` is returned.

Drop a User

The *DropUser* function drops the user specified by the `userid` parameter. The *DropUser* function expects the input parameters as a labeled delimited string that includes `USER_ID \2 userid \1`. If the `userid` parameter is not present, `SVC_DATA_SET_ERROR` error is returned. A `REVOKE CONNECT` command is executed for the specified `userid`, the `userid` is deleted from the `RPDOCS_USERS` table, and `SVC_SUCCESS` is returned.

Get Names

The *GetNames* function retrieves a list of CNWRA names from the `PEOPLE` table. The *PutListData* function is called to assemble the list of names. The function writes the label to `pReturn` data, the `name` values to `list`, and sets `pReturn` status and error.

Get RPD Document Sets

The *RPDGetDocSets* function gets a list of document sets, sorted by document set name, from the `RPD_DOCUMENT_SETS` table. The *PutListData* function is called to assemble the list of document set names. The function writes the label to `pReturn` data, the document set `name` values to `list`, and sets `pReturn` status and error.

Get a List of Active RPD Documents

The *RPDGetActiveDocList* function gets a list of active documents for a specified document set from the `RPD` table, sorted by document number. Only the most recent instance of each document is retrieved. The *PutListData* function is called to assemble the list of document numbers, titles and version numbers for each document. The function writes the label to `pReturn` data, the data values to `list`, and sets `pReturn` status and error.

Get a List of Non-Retired Documents

The *RPDGetNonRetDocList* function gets a list of documents for a specified document set with status of "VACANT" or "ACTIVE" from the *RPD* table. The document number, title and version of each selected document are returned, sorted by *SORT_ID*. The *PutListData* function is called to assemble the list of document numbers, titles and version numbers. The function writes the label to *pReturn* data, the data values to *list*, and sets *pReturn* status and error.

Get a List of Documents for a Subset Report

The *RPDGetDocSubsetList* function gets a list of document numbers and titles for documents with pieces that are consistent with the intended report query. The query statements for the specified report are retrieved from the *RPD_REPORTS* table and the predicates are assembled to form a query that is executed to retrieve the document numbers and titles of the documents required for the subset report. The *PutListData* function is called to assemble the list of document numbers and titles. The function writes the label to *pReturn* data, the document number and title values to *list*, and sets *pReturn* status and error.

Retrieve a CDS, CDM, or RPS Document

The *RPDGetCDSCDMDoc* function retrieves the binary file for the active CDS, CDM, or RPS file specified by the submitted document set and document number. The *RPDGetCDSCDMDoc* function gets the *WORKINGDIR* directory, document set, and document number from the *pPacket* access instance. The *PARTITION* and *FILENAME* for the specified document are retrieved from the *RPD* table. The archive directory is assumed to be (*db_install_directory*) and archive. From that path the function constructs path to the file as (*db_install_directory*)/archive/<*PARTITION*>/<*FILENAME*>. Then the function calls the *ReadBinaryFile* function to read the binary file. If the file read is not successful, an error code is returned. Otherwise, *SVC_SUCCESS* is returned.

Print a Status or Content Report

The *RPDDBReport* function generates and returns a status or content report. The report request is examined and used to condition the report headings and the SQL query that retrieves the document records. The report format includes the document number, title, document type, version, date, and status. The content reports include all records, but the status reports include only the most recent version of each document.

The *RPDDBReport* function obtains the current system date and then calls the *RPDOpenDBCursor* function. Table 2-6 relates the report request and the defines that are combined in the *RPDOpenDBCursor* function to form the cursor used to retrieve the documents. The generated cursor is opened and the function terminates and returns *SVC_SUCCESS*.

After the cursor has been opened by *RPDOpenDBCursor*, the *RPDDBReport* function examines the report request and builds appropriate page headings for the report. The *RPTFileOpen* function is called to open the report file. If an error occurs, the error message is returned and the *RPDDBReport* function terminates. The *RPDFetchDBCursor* function is called to retrieve each row, and the counter for the appropriate type of record is incremented. The record is formatted as an array of pointers and the *RPTSetRecord* function is called to assemble the record and write it to the report structure. When all records have been processed, the *RPDDBCcounts* function is called to print record counts. The

RPTFileClose function is called to close the report file, and the *ReadAsciiFile* function is called to read the file and return the report to the client. Then the report file is removed.

Table 2-6. Content and Status Report Requests.

Report Request	Defines combined to form the SQL query
<i>RPD_DB_STATUS_RPT_REQUEST</i>	<i>RPD_RPT_CURSOR RPD_RPT_VACANT_ACTIVE RPD_RPT_DB_DOC_SET RPD_RPT_STATUS_INST</i>
<i>RPD_DB_CONTENT_RPT_REQUEST</i>	<i>RPD_RPT_CURSOR RPD_RPT_ACTIVE_ARCHIVE RPD_RPT_DB_DOC_SET RPD_RPT_CONTENT_INST</i>
<i>RPD_RP_STATUS_RPT_REQUEST</i>	<i>RPD_RPT_CURSOR RPD_RPT_VACANT_ACTIVE RPD_RPT_RP_DOC_SET RPD_RPT_STATUS_INST</i>
<i>RPD_RP_CONTENT_RPT_REQUEST</i>	<i>RPD_RPT_CURSOR RPD_RPT_ACTIVE_ARCHIVE RPD_RPT_RP_DOC_SET RPD_RPT_CONTENT_INST</i>
<i>RPD_OI_CONTENT_RPT_REQUEST</i>	<i>RPD_RPT_CURSOR RPD_RPT_ACTIVE_ARCHIVE RPD_RPT_OI_DOC_SET RPD_RPT_CONTENT_INST</i>

Print the Open Item Status Report

The *RPDOITSStatusReport* function generates an OITS status report and returns it to the client. The report includes OITS id, topic, last update date, and resolution status.

The *RPDOITSStatusReport* function obtains the current system date and then calls the *OpenOITSStatusCursor* function to open a cursor to retrieve the most recent instance of all active and vacant OITS records, sorted by document number, from the *RPD* and *RPD_PIECES* tables. After the cursor has been opened by *OpenOITSStatusCursor*, the *RPDOITSStatusReport* function builds page headings for the report. The *RPTFileOpen* function is called to open the report file. If an error occurs, the error message is returned and the *RPDOITSStatusReport* function terminates. The *FetchOITSStatusCursor* function is called to retrieve each row and the record counter is incremented. The record is formatted as an array of pointers and the *RPTSetRecord* function is called to assemble the record and write it to the report structure. When all records have been processed, the *RPDDBCounts* function is called to print record counts. The *RPTFileClose* function is called to close the report file, and the *ReadAsciiFile* function is called to read the file and return the report to the client. Then the report file is removed.

Short DBA Report

The *RPDShortDBAReport* function produces a short form DBA report that includes the document type, document id, instance id, document number, and status record data for a specified document type. The report may be printed by document set or by date. If the document set is specified, the page headings are built for the document set format of the report. Otherwise, the page headings are built for the date format of the report. The *RPTFileOpen* function is called to open the report file. If an error occurs, the error message is returned and the *RPDShortDBAReport* function terminates. A cursor is opened to retrieve

DOCUMENT_SET, *DOCUMENT_ID*, *INSTANCE_ID*, *DOCUMENT_NUMBER*, and *DOCUMENT_STATUS* from the *RPD* table. As each record is retrieved the information is formatted and the *RPTPrtLine* function is called to print the record to the report file.

DBA Report

The *RPDDBAReport* function produces a DBA report that dumps all record data for all document types. The function retrieves the document set and current system date and then formats the page headings. The *RPTFileOpen* function is called to open the report file. If an error occurs, the error message is returned and the *RPDDBAReport* function terminates. A cursor is prepared and opened to retrieve the *DOCUMENT_ID*, *TITLE*, *DOCUMENT_NUMBER*, *DOCUMENT_SET*, *DOCUMENT_STATUS*, *MAJOR_VERSION*, *MINOR_VERSION*, *MAJOR_VERSION_NAME_ID*, *MAJOR_VERSION_DATE*, *MINOR_VERSION_NAME_ID*, *MINOR_VERSION_DATE*, *CHECKOUT_DATE*, *PARTITION*, *FILE_NAME*, and *SORT_ID* from the *RPD* table. As each record is retrieved, the major revision name, minor revision name and the checkout name are retrieved from the *RPDOCS_NAMES* table. The retrieved information is formatted and the *RPTPrtLine* function is called to print the record to the report file.

Define an RPD Record

The *RPDDefine* function defines CDS, CDM, RPS, and OITS records. The *RPDDefine* function expects a *pPacket->docset* value of "CDS", "CDM", "RPS", or "OITS". The *pPacket->data* value for RPD records is a labeled delimited string that includes *DOC_NUM \2 ~~~~ \1 TITLE \2 ~~~~ \1 MAJOR_VERSION_NAME \2 ~~~~ \1*. The *pPacket->data* value for OITS records is a labeled delimited string that includes the *OITS_ID* rather than the *REVIEW_PLAN_NUMBER*. The *RPDSetDocSet* function is called to obtain and validate the document set. The *RPDDefineGetData* and *RPDDefineCheckData* functions are called to get and validate the define data, including checking the lengths of data and checking for duplication of the record definition. The *RPDSetDocIDInstID* and *RPDSetStatusAndID* functions are called to complete definition of the record by obtaining and storing the document id, instance id, date, partition, and file name, document ID, instance ID, and document status. The *RPDSetMajorMinorVersion* function is called to set the version numbers and the *RPDSetVersionNameIDAndDate* function is called to set the version and checkout dates. The *RPDSetDocumentSortID* function is called to build the sort id and the *RPDDefineInsert* is called to insert the record and load a vacant document into Topic.

Format Check an RPD Record

The *RPDCheckFormat* function checks the format and validates CDS, CDM, or RPS records. The function calls the *RPDSetDocSet* function to retrieve and set the document set. The *RPDCheckinGetData* function is called to get the data from *pPacket*. If the version name is "MAJOR", *h_majorversionname* is set and *h_minorversionname* is set to *null*. If the version name is "MINOR", *h_minorversionname* is set and *h_majorversionname* is set to *null*. For OITS records all record entry transactions are major version changes. Therefore, for OITS records, *h_majorversionname* is set and *h_minorversionname* is set to *null*. The *RPDSetDocIDInstID* function is called to get the document id and instance id from the *RPD* table. The *RPDCheckStatus* function is called to check that status is not *DEFINE* or *CHECKIN*, because only one define or entry transaction per day is allowed for each document. If the status is *DEFINE* or *CHECKIN* the function returns *SVC_CHECKED_IN_ERROR*. The *RPDSetPartitionAndFilename* function is called to set the current and new partition based on document set id. The file name is set based on the document id and instance id.

If no errors are detected, the *RPDCheckFormat* function runs the macrostyler and Intellitag against the document. The *RPDWriteConvertAndParse* function is called to parse and insert tags into the input file. The *WriteBinaryFile* function is called to copy the submitted WordPerfect file. The *WP_TO_ASCII* script is modified to point to the active partition and file name, and then the script is executed to convert the WordPerfect file to ASCII. If the ASCII conversion fails, *SVC_CONVERT_ERROR* is returned. The *GET_DOC_INFO* awk script is modified to point to the active partition and file name, and then the script is executed to parse the ASCII file. If the parsing fails, *SVC_PARSE_ERROR* is returned.

The *RPDValidateDocument* function is called to validate the parsed document set, document number, and title against expected input values. If the transaction request is *RPD_FORCE_CHECKIN_REQUEST*, the title is updated. Otherwise, if mismatches are found the *pReturn* data is set to the identify the mismatch(es) as follows: < a label to indicate the type of mismatch(e.g., *DOCUMENT_SET*, *DOC_NUM*, or)> \2 <the expected value>\3 <the received value> \1.

The *RPDMacroStyler* function is called to execute the macro styler and create a styled file with the same input file name and an extension of “wp”. When the input file is copied, the time stamp is obtained. Then the macro styler is executed and the time stamp of the output file is obtained. If the time stamps are different, it indicates that the macro styler wrote the output file, and success is assumed. If the time stamps are the same, failure is assumed and *SVC_MACRO_FAIL_ERROR* is returned.

The *RPDIntellitag* function is called to invoke the Intellitag application and tag the styled file. Intellitag generates an output file (fn.wp.sgm) for each WordPerfect input file (fn.wp). The files are not significant, because only the Intellitag log file is examined. When the Intellitag application terminates, the *RPDIntellitagLog* function is called to check the intellitag log file for errors and report them.

Enter an RPD Record

The *RPDCheckin* function checks the format, validates, and stores CDS, CDM, or RPS records. The function calls the *RPDSetDocSet* function to retrieve and set the document set. The *RPDCheckinGetData* function is called to get the data from *pPacket*. If the version name is “MAJOR”, *h_majorversionname* is set and *h_minorversionname* is set to *null*. If the version name is “MINOR”, *h_minorversionname* is set and *h_majorversionname* is set to *null*. For OITS records, all record entry transactions are major version changes. Therefore, for OITS records, *h_majorversionname* is set and *h_minorversionname* is set to *null*. The *RPDCheckinCheckData* function is called to validate the lengths of the fields.

The *RPDSetDocIDInstID* function is called to get the document id and instance id from the *RPD* table. The *RPDCheckStatus* function is called to check that status is not *DEFINE* or *CHECKIN*, because only one define or entry transaction per day is allowed for each document. If the status is *DEFINE* or *CHECKIN* the function returns *SVC_CHECKED_IN_ERROR*.

The *RPDSetAndCheckVersion* function is called to check the version number. The version number entered by the user is checked against the current version plus major or minor increment. If the versions match, the new version is set. If the versions do not match, *SVC_OP_ERROR* is returned and the mismatch is written to *pReturn->data*. When the version number is null, a major version change is assumed and the version number is incremented. The *RPDSetStatusAndID* function is called to set the current status to *ARCHIVED*, and the new status to *CHECKIN*. The *RPDSetPartitionAndFilename* function

is called to set the current and new partition based on document set id. The file name is set based on the document id and instance id.

The *RPDCheckinInsert* function is called to insert the record. The last archived record is selected and modified. The *RPDCreateNewInstance* function is called to create a new instance of the record, and the current instance is updated in the database.

If no errors are detected, the *RPDWriteConvertAndParse* function is called to parse and insert tags into the input file. The *WriteBinaryFile* function is called to copy the submitted WordPerfect file. The *WP_TO_ASCII* script is modified to point to the active partition and file name, and then the script is executed to convert the WordPerfect file to ASCII. If the ASCII conversion fails, *SVC_CONVERT_ERROR* is returned. The *GET_DOC_INFO* awk script is modified to point to the active partition and file name, and then the script is executed to parse the ASCII file. If the parsing fails, *SVC_PARSE_ERROR* is returned.

The *RPDValidateDocument* function is called to validate the parsed document set, document number, and title against expected input values. If the transaction request is *RPD_FORCE_CHECKIN_REQUEST*, the title is updated. Otherwise, if mismatches are found the *pReturn* data is set to the identify the mismatch(es) as follows: < a label to indicate the type of mismatch(e.g., *DOCUMENT_SET*, *DOC_NUM* , or)> \2 <the expected value>\3 <the received value> \1.

If synchronization is required, the *RPDSyncOut* function is called. The *RPDSyncOut* function checks to see if the remote production or remote test database and Topic location is specified. The document set is converted to lower case and stored. The nature of the request from the client is examined and the *RPDSyncOutCheckin* function is called to format and store the synchronization transaction.

Retire an RPD Record

The *RPDRetire* function retires a CDS, CDM, or RPS record. The *RPDSetDocSet* function is called to retrieve and set the document set. The *RPDRetireGetData* function is called to get the data from *pPacket*. The document number and name are retrieved, and the *RPDSetVersionNameIDAndDate* function is called to retrieve the version name and date from the *VALID_NAME* and *DUAL* tables.

The *RPDRetire* function calls the *RPDRetireCheckData* function to validate the retirement data. If the document number or major version name is not present, *SVC_DATA_SET_ERROR* is returned. If the document number or major version number is too long, *SVC_DATA_LEN_ERROR* is returned. The *RPDRetireCheckData* function calls *RPDDocumentIsDefined* to verify that the document exists. If the record does not exist, *SVC_DATA_FIND_ERROR* is returned.

The *RPDRetire* function calls the *RPDSetDocIDInstID* function is called to retrieve the document number and instance id from the *PADB* table. The *RPDSetStatusAndID* function is called and the current status is set to "ARCHIVED" and the new status is set to "RETIRE". The *RPDSetPartitionAndFilename* function is called to retrieve the partition from the *DOCUMENT_TYPE_PARTITION* table and build the file name.

The *RPDRetireInsert* function is called to create a new instance and update it with the current retirement transaction information. Finally, the *RPDSyncOut* function is called to synchronize the retirement transaction if the record is shared.

Get a List of Report Writer Reports

The *RPDGetReportList* function retrieves a list of report writer reports, titles, types, and helps. The title, help, and privilege for each report are retrieved from the *RPD_REPORTS*, and the *PutListData* function is called to build a null-terminated, delimited list. When all reports have been retrieved, the list is returned.

Retrieve a Report Writer Report

The *RPDReportWriter* function gets the report name from report title, reads the report and returns it. If a document number is not provided, the report identifier is selected from the *RPD_REPORTS* table and the file name is constructed.

If a document number is provided, the *RPD_GENERATE_RUN* script is modified and executed. If the report name is not valid, *SVC_RPT_UNKNOWN_ERROR* is returned. If the report name is valid, the *RPD_GENERATE_FILE* is modified with the document number and stored.

If a subset report is requested, and no document number is supplied, the report identifier is selected from the *RPD_REPORTS* table and the file name is constructed from *REPORT_WRITER_TEMP*. The *RPDGenerateReport* function is called to generate the report. If the report generation is successful, the file name is built from *REPORT_WRITER_SGMU*.

The *ReadBinaryFile* function is called to read the report file and return the report.

Get a List of Loadable Document Sets

The *TDOCSGetDocSets* function retrieves and returns a list of document sets that are loadable. The names of all “local” and loadable document sets are retrieved from the *TDOCS_DOCUMENT_SETS* table. As each document set is retrieved, the *PutListData* function is called to build a null-terminated, delimited list. When all document sets have been retrieved, the list is returned.

Get the Sharable Flag and Specifications for a Document Set

The *TDOCSGetDocSetSpec* function retrieves the specification and sharable flag for a specified document set. The *TDOCSDocSetFromTitle* function is called to retrieve the *DOCUMENT_SET* and *DOCUMENT_SET_ID* from the *TDOCS_DOCUMENT_SETS* table. Then the *SPECIFICATION* and *SHARABLE* information is retrieved from the *TDOCS_DOCUMENT_SETS* table and returned.

Submit a CDOCS Document

The *TDOCSSubmitDocument* function submits a CDOCS document, inserts the document in the database, and stores the files.

The *TDOCSSubmitDocument* function calls the *TDOCSGetSubmitData* function to retrieve and check the document submission data. The *TDOCSGetGeneralData* is called to get the document set name, sharable set, host name, and source type. The document id, submission date, document number and current date are retrieved from the *TDOCS* and *DUAL* tables. If it is the first document for the day for the specified document set the document number sequence is set to “0001”. Otherwise, the document number

sequence is incremented and the document number is formatted. The *TDOCSSGetDocumentData* function is called to get the document data from *pPacket*. The *TDOCSCheckData* function is called to check for required, optional and not applicable fields. The *SPECIFICATION* data is retrieved from the *TDPCS_DOCUMENT_SETS* table and each required field is checked. If required fields are missing, *TDPCS_SET_ERR*, *NULL* is returned.

The *TDOCSSGetIncomingFiles* function is called to get the submitted files from the local directory created for submit or update, rename them by document id, and move them to the appropriate incoming document set directory. The ASCII document file must have a ".txt" extension and the image and figure file(s) must be ordered with three-digit indexes (plus a character for figures) and must have a ".tif" or ".pcx" extension. No assumptions are made about the file names of binary files. The ASCII file is moved to the *../incoming/<docset>* directory, with a file name of *<docid>.txt*. The image and figure files are moved to the *../incoming/<docset>* directory, with a file name of *<docid>.<index>.<extension>*. The binary file is moved to the *../incoming/<docset>* directory with a file name of *<docid>.bin*.

The *TDOCSSubmitDocument* function then calls the *TDOCSSInsertData* function to insert the submitted document data into the *TDPCS* table. If synchronization is required, the *TDOCSSyncOut* function is called to write synchronization transactions. The *TDOCSSyncOut* function calls *TDOCSSyncOutSubmit*, *TDOCSSyncOutUpdate*, and *TDOCSSyncOutDelete* as required. When the document submission process is completed, the document number and status is returned.

Get a CDOCS Record

The *TDOCSSGetRecord* function gets a CDOCS record by document number and formats it for return as a delimited string. The *TDOCSDocSetFromTitle* function is called to retrieve the document set and document set id from the *TDPCS_DOCUMENT_SETS* table. The *TDOCSSRetrieveData* function is called to get the document record from the *TDPCS* and associated tables. The *TDPCSFormatData* function is called to format the retrieved record, and the *TDPCSFreeAssociatedData* function is called to free memory. The retrieved record is returned as a labeled and delimited string.

Update a CDOCS Record

The *TDPCSUpdateDocument* function updates a CDOCS document by replacement.

The *TDOCSSGetUpdateData* function is called to set, get, and check the CDOCS update data. The *TDOCSSGetGeneralData* function is called to get the document set name, sharable set, host name, and source type. The *TDOCSSGetDocumentData* function is called to get the document data from *pPacket*. The document record is retrieved from the *TDPCS* and *DUAL* tables, and the *TDOCSCheckData* function is called to check for required, optional and not applicable fields. The *TDOCSSGetIncomingFiles* function is called to get the submitted files from the local submit directory that client created for submit or update, rename them by document id, and move them to the appropriate incoming document set directory. The ASCII document file must have a ".txt" extension and the image and figure file(s) must be ordered with three-digit indexes (plus a character for figures) and must have a ".tif" or ".pcx" extension. No assumptions are made about the file names of binary files. The ASCII file is moved to the *../incoming/<docset>* directory, with a file name of *<docid>.txt*. The image and figure files are moved to the *../incoming/<docset>* directory, with a file name of *<docid>.<index>.<extension>*. The binary file is moved to the *../incoming/<docset>* directory with a file name of *<docid>.bin*.

The *TDOCSDeleteAssocData* function is called to delete the associated CDOCS data for the update from the *TDOCS_AUTHORS*, *TDOCS_ADDRESSEES*, *TDOCS_ASSIGNED_NUMBERS*, and *TDOCS_ASSIGNED_CODES* tables, and the *TDOCSUpdateData* function is called to update the record in the *TDOCS* table. The *TDOCSInsertAssocData* function is called to insert the associated data in the *TDOCS_AUTHORS*, *TDOCS_ADDRESSEES*, *TDOCS_ASSIGNED_NUMBERS*, and *TDOCS_ASSIGNED_CODES* tables.

If synchronization is required, the *TDOCSSyncOut* function is called to write synchronization transactions. The *TDOCSSyncOut* function calls *TDOCSSyncOutSubmit*, *TDOCSSyncOutUpdate*, and *TDOCSSyncOutDelete* as required.

When all processing for the update transaction completes, the *TDOCSFreeAssociatedData* function is called to free memory, and status is returned in *pReturn*.

Delete a CDOCS Document

The *TDOCSDeleteDocument* function marks a CDOCS document for deletion. The record is marked but the actual deletion done during Batch processing.

The *TDOCSGetDeleteData* function is called to set, get, and check the CDOCS delete data. The *TDOCSGetGeneralData* function is called to get the document set name, sharable set, host name, and source type. The *TDOCSGetDocumentData* function is called to get the document data from *pPacket*. The document record is retrieved from the *TDOCS* and *DUAL* tables, and the *TDOCSCheckData* function is called to check for required, optional and not applicable fields. If the current record status is input, submit, or update and there are old files in incoming, the old files are deleted.

The *TDOCSDeleteData* function is called to “delete” the record by marking it for batch deletion. The *TDOCS_CIRCULATION* table is examined and if the document is checked out, *SVC_CHECKED_OUT_ERROR* is returned. The document record is updated in the *TDOCS* table and *DOCUMENT_STATUS* is set to “D”. The *TDOCSDeleteAssocData* function is called to delete the associated CDOCS data for the update from the *TDOCS_AUTHORS*, *TDOCS_ADDRESSEES*, *TDOCS_ASSIGNED_NUMBERS*, and *TDOCS_ASSIGNED_CODES* tables.

Undelete a CDOCS Record

The *TDOCSUndeleteDocument* function undeletes a document that has been marked for deletion, but has not been deleted by the Batch process. The *TDOCSDocSetFromTitle* function is called to retrieve the document set and document set id from the *TDOCS_DOCUMENT_SETS* table. If the document set is not found, *SVC_DATA_SET_ERROR* is returned. The *DOCUMENT_STATUS* for the document is changed from “D” to “L” in the *TDOCS* table.

Print CDOCS Labels

The *TDOCSLabelDocuments* function generates labels for any TDI, QA, and/or CSP document that has not yet been labeled. For each record in the TDI, QA, and CSP document sets with a status of defined, but not labeled or loaded, the subject code, document number, and title are written to a file in label format. The *DOCUMENT_NUMBER*, *TITLE*, and *ASSIGNED_CODE* information for the selected

records are retrieved from the *TDOCS* and *TDOCS_ASSIGNED_CODES* tables, formatted in label format, and returned in *pReturn* as an ASCII linked list.

Relabel CDOCS Documents

The *TDOCSRelabelDocuments* function regenerates labels for documents in specified in the data list. If the data list contains a range, of document numbers, all of the document records in the range are updated in the *TDOCS* table and the *DOCUMENT_STATUS* is set to "I". If the data list is a sequence of document numbers, each specified record is updated in the *TDOCS* table and the *DOCUMENT_STATUS* is set to "I". The *TDOCSFreeAssociatedData* function is called to free memory and the *TDOCSLabelDocuments* function is called to generate the labels.

Check Out a CDOCS Document

The *TDOCSCheckoutDocument* function checks a document, identified by the submitted document number, out to a name or location by adding a record to the *TDOCS_CIRCULATION* table. If the request is for a check out transaction, and the document is already checked out, an error is returned. However, if the request is to force a check out, then the new record is created.

The *TDOCSDocSetFromTitle* function is called to obtain the document set and the existence of the specified record in the *TDOCS* table with a status of "L" is tested. If the record is not found, *SVC_DATA_FIND_ERROR* is returned. If the request is for a normal checkout transaction (*TDOCS_CHECKOUT_REQUEST*), the existence of a record for the document in the *TDOCS_CIRCULATION* table is tested. If the record is found, *SVC_CHECKED_OUT_ERROR* is returned.

If no errors are detected, the system date is retrieved, a new record for the document is inserted in the *TDOCS_CIRCULATION* table, and *SVC_SUCCESS* is returned.

Check In a CDOCS Document

The *TDOCSCheckinDocument* function checks in a CDOCS document, identified by document number, by deleting it from the *TDOCS_CIRCULATION* table. The document number and check out name are retrieved from the input. If either the document number or check out name is in error, *SVC_DATA_SET_ERROR* is returned. If the check out name was not provided, it is retrieved from the document record in the *TDOCS_CIRCULATION* table. All records for the specified document number are retrieved and counted and the check out names are stored. If the record count is zero, *SVC_CHECKED_IN_ERROR* is returned. If the record count is greater than 1, *SVC_CHECKIN_ERROR* is returned. If the record count is 1, the record is deleted from the *TDOCS_CIRCULATION* table, based on the check out name and document number. If the deletion is not successful, *SVC_DATA_FIND_ERROR* is returned.

CDOCS Circulation Report

The *TDOCS_CirculationReport* function generates and returns a circulation report for the specified document set, and, optionally, individual checkout name. The circulation report includes the document number, checkout date, and checkout name. The current system date is retrieved and the *TDOCSDocSetFromTitle* function is called to obtain the document set. The report heading is formatted, the *RPTFileOpen* function is called to open the report file, and the *TDOCSOpenCirculationCursor* function

is called to open the cursor. The *TDOCSFetchCirculationCursor* function is called to fetch each record, and the information is formatted and written to the report file. When all records have been processed, the *RPTFileClose* and *ReadAsciiFile* functions are called to close the file and return the report information.

Print the New Acquisitions Report

The *TDOCSNewDocsReport* function generates a report on new CDOCS documents entered from given date. The *TDOCSDocSetFromTitle* function is called to obtain the document set and the *TDOCSInitNewDocsReport* function is called to initialize the report. The *TDOCSInitNewDocsReport* function retrieves the “from date” and “to date”. If either date is missing, *SVC_DATA_SET_ERROR* is returned. The current system date is retrieved and the report heading is formatted. The *TDOCSNewDocsReport* function calls *TDOCSOpenNewDocCursor* to initialize and open the cursor. The *TDOCSFetchNewDocCursor* function is called for each row and the *TDOCSWriteNewDocsReport* function is called to format the information for each record as a labeled ASCII string. When all records have been processed, the *TDOCSEndNewDocsReport* function is called close the report, the *ReadAsciiFile* function is called return the report information, and the *TDOCSFreeAssociatedData* function is called to free memory.

Get a List of Reference Reports

The *TDOCSGetRefRptList* function builds and returns list of CDOCS reference reports. A cursor is opened to retrieve the *TITLE* and *HELP* for each report from the *TDOCS_REPORTS* table. As each row is retrieved, the *PutListData* function is called to store the information. When all rows have been processed, the list of reports is returned.

Print a CDOCS Reference Report

The *TDOCSReferenceReport* function generates and returns a CDOCS reference report. A cursor is opened and the report title is used to retrieve the report identifier from the *TDOCS_REPORTS* table. If the report identifier is *HYD_REPORT*, the *ReadAsciiFile* function is called to read and return the Hydrology report. If the report identifier is *NIST_REPORT*, the *ReadAsciiFile* function is called to read and return the NIST report. If the report identifier is not recognized, *SVC_RPT_ERROR* is returned.

Print the CDOCS Statistics Report

The *TDOCSStatisticsReport* function calls the *ReadAsciiFile* function to retrieve and return the CDOCS statistics report.

Get a list of Report Writer Reports

The *SQLDTRDFetchReports* function fetches a list of report writer reports including the report id, title, help. A cursor is opened to retrieve the *REPORT*, *TITLE*, and *HELP* from the *RPD_REPORTS* table. As each row is retrieved, the *PutListData* function is called to write the information as a labeled, delimited string. When all rows have been processed, the cursor is closed and the list is returned.

Select Specifications for a Specified Report Writer Report

The *SQLDTDSelectReport* function selects and returns the report specifications for a particular report id. The report id and type are retrieved and stored, and if either is missing *SVC_DATA_SET_ERROR* is returned. A cursor is prepared and opened to retrieve *TITLE*, *HELP*, *SORT_ID*, *PRIVILEGE*, *SGML_TAG*, and *QUERY* from *RPD_REPORTS* for the specified report id and type. If the record is not found, *SVC_DATA_FIND_ERROR* is returned. Otherwise the report specification is formatted and returned.

Insert Specifications for a Report Writer Report

The *SQLDTDInsertReport* function inserts a report specification record for a report writer report. The *SQLDTDGetData* function is called to retrieve the input information, and an SQL command is executed to insert the row into the *RPD_REPORTS* table.

Update Specifications for a Report Writer Report

The *SQLDTDUpdateReport* function updates a report specification record for a specified report writer report. The *SQLDTDGetData* function is called to retrieve the input information, and an SQL command is executed to update the row in the *RPD_REPORTS* table.

Delete Specifications for a Report Writer Report

The *SQLDTDDeleteReport* function deletes a report specification record for a specified report writer report. The report id and type are retrieved and stored, and if either is missing *SVC_DATA_SET_ERROR* is returned. An SQL command is executed to delete the row in the *RPD_REPORTS* table. If the deletion is not successful, *SVC_DATA_FIND_ERROR* is returned.

Run a Query for a Report Writer Report

The *SQLDTDQueryReport* function runs an SQL query for a report and returns the tag and search text results. The query string is retrieved from the input and modified to form a valid SQL statement that includes the appropriate document numbers. When the modification of the query is complete, the SQL statement is used to open a cursor. As each row is retrieved, the *PutListData* function is called to add the retrieved information to a delimited string that is returned when all rows have been processed.

Retrieve the WordPerfect Macro or Intellitag DTD File for the Report Writer

The *SQLDTDGetFile* function retrieves WordPerfect macro (<report>.wpm) or Intellitag DTD (<report>.dtd) file for the Report Writer. The *SQLDTDSetFilePath* function is called to set the path for the specified file, and the *ReadBinaryFile* function is called to read and return the file.

Store the WordPerfect Macro or Intellitag DTD File for the Report Writer

The *SQLDTDPutFile* function stores WordPerfect macro (<report>.wpm) or Intellitag DTD (<report>.dtd) file. The *SQLDTDSetFilePath* function is called to set the path for the specified file, and the *WriteBinaryFile* function is called to store the file.

Get Help Information

The *GetHelp* function gets help header and text for a specified privilege, button, and subkey. The privilege, button, and subkey are retrieved from the input, and if any are missing, *SVC_DATA_SET_ERROR* is returned. An SQL query is prepared and executed to retrieve the *HEADER* and *TEXT* from the *RPDOCS_HELP* table, and the retrieved information is returned as a delimited string.

Get a List of Help Topics

The *GetHelpList* function gets a list of help button, subkeys, and headers for specified privilege, and writes the output to *pReturn* as a delimited string. The privilege is retrieved from the input, and if it is missing, *SVC_DATA_SET_ERROR* is returned. An SQL query is prepared and executed to retrieve the button, subkeys, and headers for the specified privilege from the *RPDOCS_HELP* table, and the retrieved information is returned as a delimited string.

2.4 BATCH SYSTEM TECHNICAL DOCUMENTATION

The purpose of the Batch system is to permit efficient updating of the Topic database during non-prime hours. The Topic update process requires permissions for the Topic directories. Therefore, the Batch process must be run by the Topic user.

The Batch process is initiated from the command line with parameters that determine the type of processing to be performed.

2.4.1 The Batch Process

The *da_batch* script is executed to start the main Batch process. This process must be run by the Topic user, because one of the main purposes of processing is loading documents into Topic.

2.4.1.1 Batch Command Line Parameters

The *da_batch* process requires two command line parameters. The first *da_batch* command line argument identifies the userid, password, and database instance. The form of the first command line argument is

<uid>/<pwd>@<inst>

where

uid is the userid

pwd is the passwd of the database administrator

inst is the database instance (e.g., test1 or prod1).

The second *da_batch* command line argument identifies the type of processing required. The form of the second command line argument is

cdocs	TDOCS and RPD
tdocs	all of TDOCS
rpdc	all of RPD, Report Writer on change
rpt	only RPD reports
cds [docnum]	only CDS documents
cdm [docnum]	only CDM documents
oits [docnum]	only OITS documents
stat	only statistics report
hydro	only hydrology references
nist	only nist references

The Batch process produces two reports: a Topic report, and a RPD report. These reports are written to the directory

/<topicdb>/<inst>/reports/batch

where

topicdb is the drive where the Topic database is located

inst is instance (e.g., prod1 or test1)

A third report can be produced by redirecting standard output to a file.

2.4.2 Main Batch Program

The main batch program takes a database access string and data set as argument and sets pointers to access the data set. It then performs a login and gets current date and extension form of the date (YYYYMMDD). It changes to either the test or production Topic database, batches TDOCS documents or RPD documents, and performs a logout. A simple log file may be captured via redirection. Otherwise each Batch operation produces its own log.

2.4.2.1 Retrieve the Command Line Arguments

The *GetCommandLineArgs* function is called to retrieve the command line arguments. If the version string, version number, access, instance, or data set is missing, a usage error message is displayed and the process terminates.

The *GetCommandLineArgs* function captures command line arguments and sets the access, instance and data set variables. The command line arguments contain the following parameters:

int argc	number of command line arguments
char** argv	command line arguments
char** verStr	version string
int* verNum	version number
char** access	database access string
char** instance	database instance & directory path

char** dataSetArg	data set to batch
char** docNumArg	document number to batch
int* dataSet	data set flag

If the number of command line arguments is greater than three, the version string pointer is set, the version is changed to an integer, the database access pointer is set, and the instance pointer is set. Then the data set argument is examined and the *dataSet* flag is set as indicated in Table 2-7.

2.4.2.2 Initialize the Environment

The *RPDOCSInitEnv* function is called to initialize the environment variables from the ini file. If the initialization is not successful, the *ENVIRON_FAIL* message is displayed and the Batch process terminates. The *RPDOCSInitEnv* function calls *RPDOCSVoidEnv* to set the environment variables to null values. The path is built for the initialization file path and the file is opened. If the file open fails, the function terminates. The initialization file is read, and when a section heading is found the *RPDOCSSetEnv* function is called to initialize the environment setting for the section. When the entire initialization file has been read, the file is closed and the *RPDOCSInitEnv* function terminates.

Table 2-7. Initialization of the data set flag

Data Set Argument	<i>dataSet</i> Flag
TDOCS_ARG	BATCH_TDOCS
RPD_ARG	BATCH_RPD
RPT_ARG	BATCH_RPT
CDS_ARG	BATCH_CDS
CDM_ARG	BATCH_CDM
OITS_ARG	BATCH_OITS
STATS_ARG	BATCH_STATS
HYDRO_ARG	BATCH_HYDRO
NIST_ARG	BATCH_NIST

2.4.2.3 Log In

The *LogIn* function is called to perform the login. If the login is not successful, the Batch process terminates. The *LogIn* function logs in as *dba* using *uid/pwd@instance* for access. The userid, password, and instance are assembled into a character string and an SQL *CONNECT* command is executed. If the connect fails, an error message, "Unable to log in: check database access string ", is displayed and *FAILURE* is returned.

2.4.2.4 Get the Report Date

The *GetDateExtension* function is called to get the report date. If the function execution is not successful, the Batch process terminates. The *GetDateExtension* function gets the current date from the Oracle database and sets pointers to the date and extension. An SQL command is executed to retrieve the system date in two forms, “DD-MON-YYYY” and “YYYYMMDD”, from the *DUAL* table. The *curDate* pointer is set to point to the date with the “DD-MON-YYYY” format, and the extension pointer is set to point to the date with the “YYYYMMDD” format.

2.4.2.5 Set the Working Directory

The *ChangeWorkingDir* function is called to set the working directory. If working directory cannot be set, an error message, “xxx is not a valid topic directory”, and the Batch process terminates. The *ChangeWorkingDir* function modifies the *RPDOCS_WORKING_DIR* string to reflect the production or test database as appropriate, and then executes a *chdir* command to change the working directory. If the *chdir* comm is unsuccessful, *FAILURE* is returned.

If the initialization process completes without errors, a message is printed that includes the *RPDOCS_BATCH_RPT* message, the version string, the database instance, the data set arguments, and any document number arguments.

2.4.3 TDOCS Batch Execution

The data set argument is examined to determine the type of processing required. If the data set argument is *BATCH_TDOCS*, *BATCH_STATS*, *BATCH_HYDRO*, or *BATCH_NIST*, the *TDOCSBatch* function is called to run the TDOCS batch procedures. When the *TDOCSBatch* function completes, a message is printed that includes the *TDOCS_BATCH* message, the log files, the instance, the version string and the extension. Figure 2-47 illustrates the overall batch process.

The *TDOCSBatch* function processes CDOCS document sets, and performs database maintenance functions for each loadable document set in the following order:

- Fetch the document set name and specification
- Update all records marked for update
- Submit all records marked for submission
- Delete all records marked for deletion

The *OpenBatchReport* function is called to open the report file, and the *WriteBatchReport3* function is called to modify *TDOCS_BATCH_REPORT_HEADER* with the instance and current date and write the batch header to the report file. The *WriteBatchReportEnv* is called to write the environment variables to the batch report file, and the associated data and batch control variables are initialized to *null*.

If the data set is *BATCH_TDOCS*, the *TDOCSOpenBatchSets* function is called to obtain the names of all documents to be processed. The *TDOCSFetchBatchSet* function is called for each document

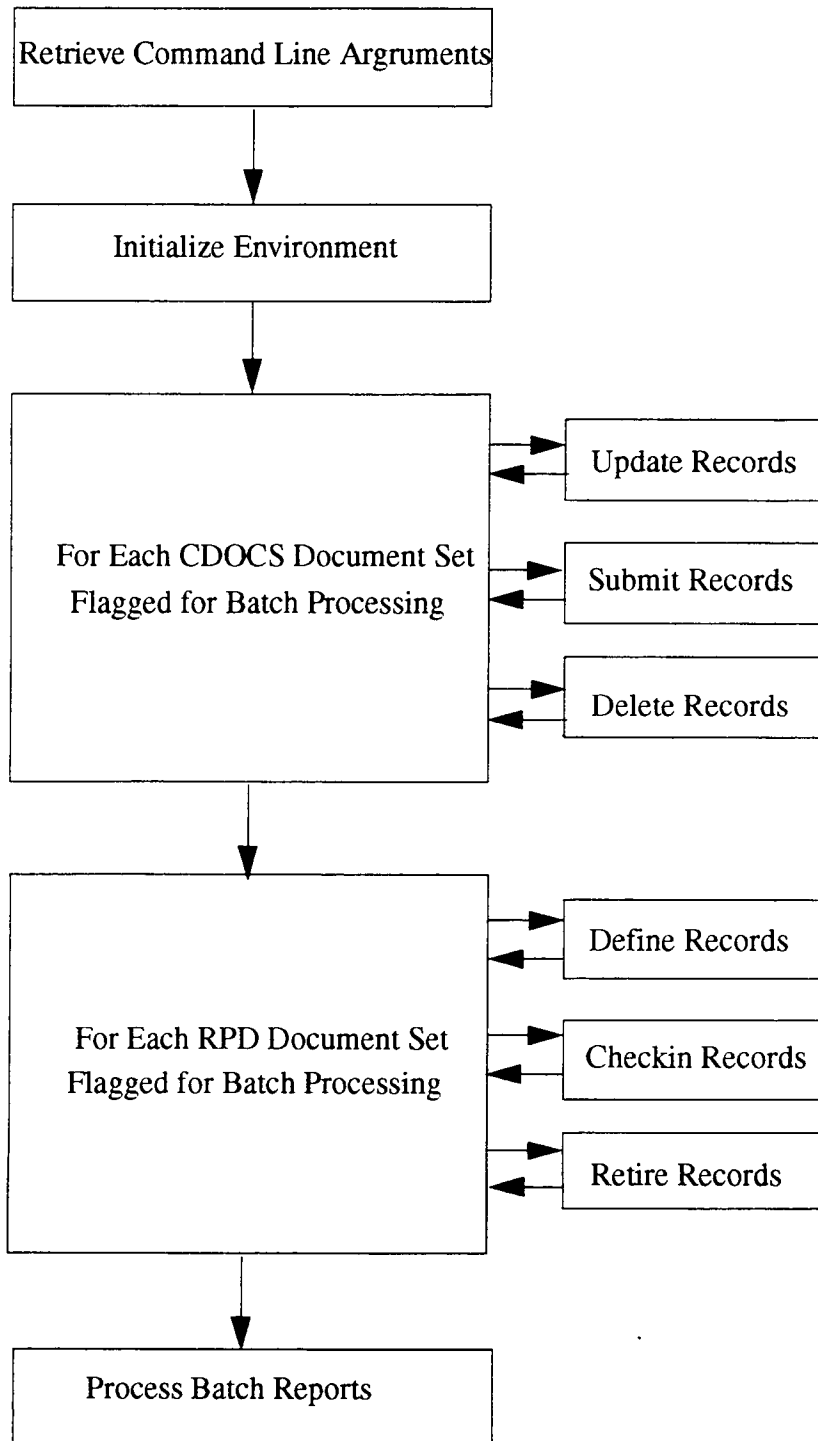


Figure 2-47. Batch

and the *TDOCSUpdateDocuments*, *TDOCSSubmitDocuments*, and *TDOCSDeleteDocuments* functions are called to update, submit, or delete the documents.

When all documents have been processed, if any documents have been loaded, the data set is set to *BATCH_STATS* to force the database statistics report to be generated. If the data set is *BATCH_STATS*, *BATCH_HYDRO*, or *BATCH_NIST* the *TDOCSBatchReports* function is called to generate the database statistics, hydrology, or NIST reports. The *CloseBatchReport* function is called to close the batch report and the *TDOCSBatch* function terminates.

2.4.3.1 TDOCS Batch Execution—Open Batch Sets

The *TDOCSOpenBatchSets* function opens a cursor on document sets to be batched and fetches them into a list the first time. Each time it sets *h_documentset*. The *SLLInitialize* function is called to allocate and initialize a pointer to a singly linked list, and a cursor is opened to select *DOCUMENT_SET*, *SPECIFICATION*, *LOCATION*, *SHARABLE* for all loadable document sets from the *TDOCS_DOCUMENT_SETS* table. As each row is retrieved, the information is formatted and appended to the singly linked list. When all rows have been retrieved, the cursor is closed.

2.4.3.2 TDOCS Batch Execution—Fetch Batch Sets

The *TDOCSFetchBatchSet* function gets the specification and other data for each document set. The first time the function is executed, the link list pointer is initialized to the first entry. Thereafter, the link list pointer is incremented to point to the next entry. When a null link list entry is found, the entire link list is deleted and the memory is freed. An SQL statement is executed for the current document set to select *SPECIFICATION*, *LOCATION*, and *SHARABLE* from the *TDOCS_DOCUMENT_SETS* table.

2.4.3.3 TDOCS Batch Execution—Update Documents

The *TDOCSUpdateDocuments* function updates document set records marked for update. The function opens a cursor on the document set records marked for update, and processes each fetched record:

- Unload the document from TOPIC
- Restore the document files to the *incoming* directory so that they can be submitted
- Remove the update bit from *source*
- Update the status of all updated records to *submit* status
- Write information to the batch report

The effect of this process is to delete the old record and then cause the new record to be submitted so that the updated record is inserted in Topic.

The function *TDOCSUpdateDocuments* function calls *WriteBatchReport2* to write an update header to the batch file. The *TDOCSOpenBatchCursor* function is called to open a cursor that retrieves *DOCUMENT_ID*, *SOURCE_TYPE*, *DOCUMENT_NUMBER*, *TITLE*, *PUBLICATION*,

DOCUMENT_DATE, *DURATION*, and *NOTE* from the *TDOCS* table for all documents in the specified document set that have a status of *TDOCS_UPDATE_STATUS*.

The *TDOCSFetchBatchRecord* function is called to retrieve the next document and write a record to the batch report file, and the *TDOCSSetBatchControl* function is called to set the logic for the update.

The *TDOCSSetBatchControl* function controls logic of batch update, submit, delete through the following process:

- Determine whether it is time to load a partition (when *DOCS_PER_PARTITION* documents have been loaded, *waitonload* is set to 0).
- Determine whether the load should insert documents (previous partition with less than *DOCS_PER_PARTITION* documents) or create a new partition (*newPartition* set to 1).
- Determine the name of the partition to insert into or create (in *subpartnumber*).
- Determine whether the document is the first of a load request (*tdocsFirstDocument* set to 1).
- Determine the document id of the first document of a partition (in *h_firstpartdoc*).
- Create partition as needed.

The *TDOCSSetBatchControl* function checks for the existence of the target partition for the document set. If the partition does not exist, *FAILURE* is returned. If the batch request is *TDOCS_BATCH_SUBMIT*, the function checks for a partition boundary, and calls the *DOCSCreatePartDirs* function to create a new partition if necessary. If the batch request is *TDOCS_BATCH_UPDATE* or *TDOCS_BATCH_DELETE* the current partition for the document is obtained and the function terminates.

The *TDOCSUnloadDocument* function is called to delete the document from Topic. The *TDOCS_DELETE_CONTROL* file is opened and the *TDOCS_DELETE_SCRIPT* is modified to reflect the target document and written to the file. This script includes the control statements for the *mkbuild* utility that is used to delete the document from TOPIC. The file is closed and *TDOCS_DELETE_DOCUMENT* is modified to point to the correct partition and executed to invoke the *rpdocs_delete_doc* script and delete the document.

The *TDOCSUpdateFiles* function is called to move the archive and image files back to the *incoming* directory when the update transaction is not accompanied by submitted files. This is done so that the submit processing will have files to submit. If files are submitted with the update transaction, the old files from the archive and image directories are not moved. The *TDOCSRemoveLoadLinkFiles* function is called to remove the document's load and link files. If either the load or link files cannot be removed, an error message is written to the batch report file. If the document does not have current files submitted, but has previously submitted text files, the *TDOCSRestoreTxtFile* function is called to restore the text file to the *incoming* directory. If the document does not have current files submitted, but has previously submitted binary files, the *TDOCSRestoreBinFile* function is called to restore the binary file to the *incoming* directory. If the document does not have current files submitted, but has previously submitted

image files, the *TDOCSRestoreImgFiles* function is called to restore the image files to the *incoming* directory. If the document has a current text file and a previous text file, the *TDOCSRemoveTxtFile* function is called to remove the text file. If the document has a current binary file and a previous binary file, the *TDOCSRemoveBinFile* function is called to remove the binary file. If the document has current image files and previous image files, the *TDOCSRemoveImgFiles* function is called to remove the image files.

The *TDOCSUpdateSource* function is called for update transactions to update the source and remove the update bits. An SQL statement is executed to update *SOURCE_TYPE* in the *TDOCS* table for the document record.

2.4.3.4 TDOCS Batch Execution—Submit Documents

The *TDOCSSubmitDocuments* function submits TDOCS records from Oracle to Topic. All records marked input are changed to submit status. A cursor is opened for all TDOCS records marked for submission. The following process is performed for each record:

- The record is fetched from the database.
- The partition and document information is set.
- A partition is created if needed.
- Files are copied from incoming to the partition subdirectory.
- The load and link files are written and the file name is appended to the file list.
- If the partition is full, the prepared documents are loaded into Topic and the status of those records is updated.
- When the load files and file lists have been generated, any remaining prepared documents are loaded into Topic and the status of those records is updated.
- If the submit processing is successful, the transactions are committed and the incoming files are cleaned up.

The submit processing starts by calling the *WriteBatchReport2* function. It writes a submit header to the report file, and initialize the *batchCtl* variables for first document and partition boundary. The *TDOCSBatchUpdateStatus* function is called to change documents with *TDOCS_INPUT_STATUS* to *TDOCS_SUBMIT_STATUS*.

The *TDOCSOpenBatchCursor* function is called to open a cursor that retrieves *DOCUMENT_ID*, *SOURCE_TYPE*, *DOCUMENT_NUMBER*, *TITLE*, *PUBLICATION*, *DOCUMENT_DATE*, *DURATION*, and *NOTE* from the *TDOCS* table for all documents in the specified document set that have a status of *TDOCS_SUBMIT_STATUS*.

The *TDOCSFetchBatchRecord* function is called to retrieve the next document and write a record to the batch report file. If no records are found, *BATCH_TDOCS_NONE* is returned. For each record, the *TDOCSSetBatchControl* function is called to control the logic of the submit processing:

- Determines whether it's time to load a partition (when *DOCS_PER_PARTITION* documents have been processed, *waitonload* is set to 0).
- Determines whether the load should insert documents (previous partition with less than *DOCS_PER_PARTITION* documents) or create a new partition (*newPartition* set to 1).
- Determines the name of the partition to insert into or create (in *subpartnumber*).
- Determines whether the document is the first of a load request (*tdocsFirstDocument* set to 1).
- Determines the document id of the first document of a partition (in *h_firstpartdoc*).
- Creates partition as needed.

The *TDOCSSetBatchControl* function checks for the existence of the target partition for the document set. If the partition does not exist, *FAILURE* is returned. If the batch request is *TDOCS_BATCH_SUBMIT*, the function checks for a partition boundary, and calls the *DOCSCreatePartDirs* function to create a new partition if necessary.

The *TDOCSSubmitFiles* function is called to store any files associated with the submitted record. If a text file has been submitted, the *TDOCSCopyTxtFile* function is called to store the text file. If a binary file has been submitted, the *TDOCSCopyBinFile* function is called to store the binary file. If image files have been submitted, the *TDOCSCopyImgFiles* function is called to store the image files.

The *TDOCSWriteLoadLinkFiles* function is called to write files to the load and link directories. If a text file is submitted, it is opened and the output load and link files are also opened. The *TDOCSWriteHeaderFields* function is called to write the header information to both the load and link files. The *TDOCSWriteDocumentLabel* function is called to write the document text launch pad label information to the load file, and the *TDOCSWriteDocumentPattern* function is called to write the document launch pad pattern to the link file so that the Topic utility will be able to parse the file and insert the link. The *TDOCSWriteHeaderFields* function is called to write the external header fields to both the load and link files.

If image files are submitted, the *TDOCSGetImageListFile* function is called to create and open the image list file. A directory list command is executed to retrieve the names of the image files, and the output is directed to the image list file. Then the *TDOCSWriteImageLabelsPatterns* function is called to insert image launch pad patterns for the images at the top of each text page.

The *TDOCSAppendAsciiText* function is called to append the ASCII text to the load and link files if a text file is submitted. The *TDOCSAppendCleanUp* function is called to add the document id to a singly linked list of document files requiring cleanup, and the *TDOCSFetchBatchRecord* function is called to see if there are more documents to be processed.

The *TDOCSCheckLoadBoundary* function is called to check whether the next document id falls within the current partition. If it is, the boundary flag is set to one and the wait on loading flag is set to zero. If a partition boundary has been reached, the boundary flag is set to zero.

If the partition boundary flag or the wait on loading flag is set, the *TDOCSLoadPartition* function is called to load the document into the Topic partition by creation or insertion. Scripts are modified and executed to move the load files down one level to the load directory and to move the link files up one level from the link directory. If a new partition is required, the *TDOCS_CREATE_PARTITION* script is modified and executed to run *rpdocs_load_part* to create the new partition. Otherwise, the *TDOCS_INSERT_PARTITION* script is modified and executed to run *rpdocs_load_part* to rebuild the partition. If the partition load fails, the *TDOCS_MV_LOAD_FILES_UP* script is modified and executed to restore the load files to their original directories. The *TDOCS_LINK_DOCS* script is modified and executed to build the links, and the *TDOCS_MV_LOAD_FILES_UP* script is modified and to move the load files up to their original directory.

When all submitted documents have processed, the *somethingLoaded* flag is set and the *TDOCSBatchUpdateStatus* is called to update the status of the document records. Then the *TDOCSCleanUpIncoming* function is called to clean out the incoming document set files after successful loading.

2.4.3.5 TDOCS Batch Execution—Delete Documents

The *TDOCSDeleteDocuments* function deletes records marked for deletion from CDOCS. Each record marked for deletion is unloaded from Topic, its files are deleted, and the database record status is changed to “deleted”.

The *WriteBatchReport2* function is called to write a deletion header to the batch report file. The *TDOCSOpenBatchCursor* function is called to open a cursor that retrieves *DOCUMENT_ID*, *SOURCE_TYPE*, *DOCUMENT_NUMBER*, *TITLE*, *PUBLICATION*, *DOCUMENT_DATE*, *DURATION*, and *NOTE* from the *TDOCS* table for all documents in the specified document set that have a status of *TDOCS_DELETE_STATUS*.

The *TDOCSFetchBatchRecord* function is called to retrieve the next document and write a record to the batch report file. If no records are found, *BATCH_TDOCS_NONE* is returned.

The *TDOCSSetBatchControl* function controls logic of batch update, submit, delete through the following process:

- Determine whether it's time to load a partition (when *DOCS_PER_PARTITION* documents have been processed, *waitonload* is set to 0).
- Determine whether the load should insert documents (previous partition with less than *DOCS_PER_PARTITION* documents) or create a new partition (*newPartition* set to 1).
- Determine the name of the partition to insert into or create (in *subpartnumber*).
- Determine whether the document is the first of a load request (*tdocsFirstDocument* set to 1).

- Determine the document id of the first document of a partition (in *h_firstpartdoc*).
- Create partitions as needed.

The *TDOCSSetBatchControl* function checks for the existence of the target partition for the document set. If the partition does not exist, *FAILURE* is returned. If the batch request is *TDOCS_BATCH_DELETE* the current partition for the document is obtained and the function terminates.

The *TDOCSUnloadDocument* function is called to delete the document from Topic. The *TDOCS_DELETE_CONTROL* file is opened and the *TDOCS_DELETE_SCRIPT* is modified to reflect the target document and written to the file. This script includes the control statements for the *mkbuild* utility that is used to delete the document from TOPIC. The file is closed and *TDOCS_DELETE_DOCUMENT* is modified to point to the correct partition and executed to invoke the *rpdocs_delete_doc* script and delete the document.

The *TDOCSDeleteFiles* function is called to delete the associated files for the document. The *TDOCSRemoveLoadLinkFiles* function is called to remove the document's load and link files. If either the load or link files cannot be removed, an error message is written to the batch report file. If the document has a text file, the *TDOCSRemoveTxtFile* function is called to remove the text file. If the document has a binary file, the *TDOCSRemoveBinFile* function is called to remove the binary file. If the document has image files, the *TDOCSRemoveImgFiles* function is called to remove the image files.

When all deleted documents have been processed, the *TDOCSBatchUpdateStatus* is called to update the status of the document records.

2.4.3.6 TDOCS Batch Execution—Batch Reports

The *TDOCSBatchReports* function examines the *dataset* parameter and runs the requested report. If the *dataset* parameter is *BATCH_STATS* the *TDOCSStatisticsReport* function is called to generate the database statistics report. If the *dataset* parameter is *BATCH_HYDRO* the *TDOCSHydroRefReport* function is called to generate the Hydrology reference report. If the *dataset* parameter is *BATCH_NIST* the *TDOCSNISTRefReport* function is called to generate the NIST reference report.

The *TDOCSStatisticsReport* function generates the database statistics report. The system date is retrieved from Oracle, and the *WriteBatchReport1* function is called to write a header record to the batch report file. The *RPTFileOpen* function is called to open the report file, and a cursor is opened to retrieve the *TABLE_NAME*, *DOCUMENT_SET*, and *TITLE* from the *TDOCS_DOCUMENT_SETS* table for all document sets. As each document set record is retrieved a header is written to the report file. A cursor is prepared to count the document set entries, and the count information is formatted and written to the report file. When all document sets have been processed, the report file is closed.

The *TDOCSHydroRefReport* function generates the hydrology reference report. The system date is retrieved from Oracle, and the report heading is formatted. The *RPTFileOpen* function is called to open the report file, and the *TDOCSOpenHydroRefCursor* function is called to open the cursor. The *TDOCSFetchHydroRefCursor* function is called to retrieve each record, *RPTFormatLine* function is called to format the line for each document, and the *RPTPrtLine* function is called to print the line. When all records have been processed, the *RPTFileClose* function is called to close the report file.

The *TDOCSNISTRefReport* function generates the NIST reference report. The system date is retrieved from Oracle, and the report heading is formatted. The *RPTFileOpen* function is called to open the report file, and the *TDOCSOpenNISTRefCursor* function is called to open the cursor. The *TDOCSFetchNISTRefCursor* function is called to retrieve each record, *RPTStuffLine* function is called to build the line for each document. The *RPTFormatLine* function is called format the line for each document and perform line wrap, and the *RPTPrtLine* function is called to print the line(s). When all records have been processed, the *RPTFileClose* function is called to close the report file.

2.4.4 RPD Batch Execution

The data set argument is examined to determine the type of processing required. If the data set argument is *BATCH_RPD*, *BATCH_RPT*, *BATCH_CDS*, *BATCH_CDS*, or *BATCH_OITS*, the *RPDBatch* function is called to run the RPD batch procedures. When the *RPDBatch* function completes, a message is printed that includes the *RPD_BATCH* message, the log files, the instance, the version string and the extension.

The *RPDBatch* function batch processes RPD document defines, checks-in, and generates a batch report. If any records are successfully processed, *doReports id* is returned non-zero. The *OpenBatchReport* function is called to open the batch report file, and *WriteBatchReport3* function is called to write a header to the report file. The *WriteBatchReportEnv* function is called to write the environment information to the report file.

2.4.4.1 RPD Batch Execution—Define

If the batch request is *BATCH_RPD*, the *RPDBatchDefineDocuments* function is called . The *RPDBatchDefineDocuments* function batch processes RPD document defines and generates batch report. The function fetches all records in the document set marked for define into a list. For each record the function performs the following processing:

- Writes vacant ASCII, WordPerfect, and SGML files to the incoming directory
- Parses the SGML file and stores it in the piece table
- Writes the header information to an information file
- Concatenates the information and ASCII files to form a load document
- Appends the file name to the load file list
- Stores the WordPerfect document in the archive
- Cleans up and fetches the next record
- Batch loads, links and secures files into Topic
- Updates the document status from *define* to *vacant*

The *WriteBatchReport2* function is called to write a header record to the report file. The *RPDGetBatchRecords* function is called to prepare a singly linked list of documents to be defined. If no records are flagged to be defined, *RPD_NOTHING_TO_DEFINE* is written to the report file and the function terminates. The *RPDInitLoad* function is called select the proper partition and delete the appropriate file list. The *SLLHead* function is called to initialize a singly linked list to point to the head of the list. The *RPDWriteVacant* function is called to create vacant ASCII, WordPerfect, and SGML documents with the document number and title. The *RPDPieceDocument* function is called to parse the document into pieces and store them in the *PIECES* table. The *RPD_PIECE_DOCUMENT* is modified and executed to run the *pieces* program. When the *pieces* program competes, any error message are written to the report file. The *RPDWriteHeader* function is called to write the document header with all control information to a file. The *RPDBuildLoadDoc* function is called to concatenate the header and ASCII files to form a load document. The *RPDWriteFileList* function is called to append the file name to the file list, and the *RPDStoreDocument* function is called to write the header for document with all control information.

When all records in the list have been processed, if the record count is greater than zero , the *RPDLoadDocuments* function is called to load the Topic partition by creation or insertion, adds links to launch Word Perfect, and secure the documents. The *RPDUpdateDefineRecords* function is called to update the status for the defined records in the RPD table. The *RPDSummaryReport* function is called to produce a summary report of good and bad records. Then the *RPDFreeBatchRecords* function is called to free memory and the function terminates.

2.4.4.2 RPD Batch Execution—Checkin

The *RPDBatch* function calls the *RPDBatchCheckinDocuments* function. The *RPDBatchCheckinDocuments* function batch processes RPD document checkin transactions and generates an updated batch report. The function fetches all records in the document set marked for checkin into a list. For each record the function performs the following processing:

- All records in document set marked for checkin are fetched and inserted into a linked list
- The report writer functions are run against each document
 - A copy of the file is processed by the macro styler to generate a styled WordPerfect document
 - Intellitag is executed to process the styled WordPerfect documents and generate SGML instances
 - The *pieces* program is run on all *sgml* instances to store the SGML pieces in the database
- Topic loading functions are performed for each record in the list
 - The document header is written to an information file
 - The header information file is concatenated with the ASCII file to make a load document

- The file name is appended to the checkin load file list
- The checkin WordPerfect document is stored in the archive
- The file name is appended to the archive load file list
- The archive WordPerfect document is stored in the archive
- The incoming directory is cleaned up
- The next record is fetched
- The checkin files are loaded, linked, and secured in Topic
- The archive files are loaded, linked, and secured in Topic
- The document status is changed from checkin to active

The *WriteBatchReport2* function is called to write a header record to the report file. The *RPDGetBatchRecords* function is called to prepare a singly linked list of documents to be checked in. If no records are flagged to be checked in, *RPD_NOTHING_TO_CHECKIN* is written to the report file and the function terminates.

The *RPDMacroStyler* function is called to run the macro styler against each submitted file. The file is copied to file.wp, the file time stamp is obtained, and the macro styler is run against it. When the macro styler completes, the time stamp of the output file is compared to the time stamp of the input file. If the time stamps are different, the function assumes that the macro styler competed successfully. If the time stamps are the same, an error is assumed. This process generates a styled file (fn.wp) for each checkin file. The *FlushBatchReport* function is called to close and reopen the batch report file for appending.

The *RPDIntellitag* function is called to submit the WordPerfect files created by the macro styler to the Intellitag program for processing. The output of the Intellitag program is a tagged SGML file (fn.wp.sgm), that is processed by the *pieces* program. The *FlushBatchReport* function is called to close and reopen the batch report file for appending.

If the checkin processing is successful, the *RPDUpdateCheckinRecords* function is called to change the status in the RPD table for all records in the document link list to "ACTIVE". Then the files in the .bat subdirectory are moved to the .hdr subdirectory in the outgoing directory so that the remote server can access them during synchronization, and the *RPDFreeBatchRecords* function is called to free memory. The *RPDPieces* function is called to parse the document into pieces and store them in the *PIECES* table. The *RPDPieces* calls *RPDPieceDocument* and the *RPD_PIECE_DOCUMENT* script is modified and executed to run the *pieces* program. When the *pieces* program competes, any error message are written to the report file, the *FlushBatchReport* function is called to free memory, and the function terminates.

2.4.4.3 RPD Batch Execution—Retire

The *RPDBatch* function calls the *RPDBatchRetireDocuments* function. The *RPDBatchRetireDocuments* function batch processes RPD document retirement transactions and generates a batch report. The function fetches all records in the document set marked for retire into a list. For each record the function performs the following process:

- All records in the document set that are marked for retirement are fetched and inserted into a singly linked list
- Topic loading functions are performed for each record in the list
 - The file name is appended to the archive load file list
 - The archive WordPerfect document is stored in the archive directory
 - The incoming directory is cleaned up
 - The next record is fetched
 - The archive files are loaded, linked, and secured in Topic
- The status of the most record is updated and the pieces are deleted

The *RPDBatchRetireDocuments* calls the *WriteBatchReport2* function to write a header record to the report file. The *RPDGetBatchRecords* function is called to prepare a singly linked list of documents to be retired. If no records are flagged to be retired, *RPD_NOTHING_TO_RETIRE* is written to the report file and the function terminates.

The *RPDInitLoad* function is called select the proper partition and delete the appropriate file list. The *SLLHead* function is called to initialize a singly linked list to point to the head of the list. The *RPDDelArchive* function is called to delete the of the document from the active Topic partition. The *RPDWriteFileList* is called to append the file name to the file name list. The *RPDMoveArchive* function is called to move the previous version of the document from the active to archive Topic partition.

When all records in the list have been processed, if the record count is greater than zero , the *RPDLoadDocuments* function is called to load the Topic partition by creation or insertion, add links to launch Word Perfect, and secure the documents. The *RPDUpdateRetireRecords* function is called to update the status for the retired records in the RPD table. The *RPDDelRetirePieces* function is called to delete the pieces of the retired document records from the *PIECES* table. The *RPDSummaryReport* function is called to produce a summary report of good and bad records. Then the *RPDFreeBatchRecords* function is called to free memory and the function terminates.

2.4.5 Logout and Exit

When the Batch processing is complete, the *LogOut* function is called to log out from the database, and the Batch process terminates.

3 COMPILATION AND EXECUTION INSTRUCTIONS

This section describes the methods for compiling and executing the various components of the CDOCS system. Wherever possible, platform independent methods are discussed first, with platform dependent methods discussed in separate sections.

3.1 COMPILING CDOCS COMPONENTS

The CDOCS system is made up of seven major executables and a number of minor “helper” modules. The major executables include the batch and server modules, which are compiled and executed on the server machine, and five client modules that are compiled and executed on a machine of that client type. The batch and server modules were developed on a SUN workstation under Solaris 2.3. The five clients include a Windows 3.1 client developed on a PC under Microsoft Windows 3.1, an OS/2 client developed on a PC under OS/2 Version 2.1, a Macintosh developed on a Macintosh under System 7, a SUN OS client developed on a SUN workstation under SUN OS Version 4.1.3, and a SOLARIS client developed on a SUN workstation under Solaris 2.3.

Wherever possible, tools like *make* or *nmake* were used to automate the build process. To the greatest extent possible, paths within makefiles were specified based upon defined environmental variables to eliminate the need to rewrite makefiles at installation time. To recompile a component after installation of the CDOCS system, the user needs to verify that the necessary environmental variables are defined for the users system configuration and then run *make*.

Table 3-1 summarizes information about all of the executables that make up the CDOCS system. Script files are listed as executables.

3.1.1 CDOCS Server

The main CDOCS server executable, *da_server*, is compiled by changing to the server distribution directory and running *make* against the supplied makefile, i.e.

```
cd (distribution_base_directory)/rpdocs/development/server
make
```

Table 3-1. CDOCS executables

Executable Name	Type	Platform	Operating System	Function
main	Compiled C	SUN	Solaris 2.3	Client
main	Compiled C	SUN	SUN OS 4.1.3	Client
main.exe	Compiled C	PC	Windows 3.1	Client
startwp.exe	Compiled C	PC	Windows 3.1	WordPerfect Launch
winprint.exe	Compiled C	PC	Windows 3.1	Printing

Table 3-1. CDOCS executables (cont'd)

Executable Name	Type	Platform	Operating System	Function
rpc16.dll	Compiled C	PC	Windows 3.1	RPC 32 bit to 16 bit "thunking"
main.exe	Compiled C	PC	OS/2 2.1	Client
startwp.exe	Compiled C	PC	OS/2 2.1	WordPerfect Launch
startjp.exe	Compiled C	PC	OS/2 2.1	PM JPEG Viewer Launch
main	Compiled C	Macintosh	System 7	Client
da_server	Compiled C	SUN	Solaris 2.3	Server
restartda	sh script	SUN	Solaris 2.3	Server Management
writer	Compiled C	SUN	Solaris 2.3	Report Writer
pieces	Compiled C	SUN	Solaris 2.3	Document Parser
da_help	Compiled C	SUN	Solaris 2.3	Help Modification Routines
da_kill	Compiled C	SUN	Solaris 2.3	Kill Running Server
kill_da	Compiled C	SUN	Solaris 2.3	Kill Running Server
restart_da	sh script	SUN	Solaris 2.3	Kill Server and Restart Server
da_batch	Compiled C	SUN	Solaris 2.3	Batch Loading

This compiles da_server in debug mode, i.e. with support for interactive debugging using dbx or another related tool. Table 3-2 summarizes information from Makefile for standard (default) compilation.

By contrast, a release version without debugging support and optimized for speed is produced by adding the **release** keyword to make, i.e.

```
cd (distribution_base_directory)/rpdocs/development/server
make release
```

The -g compiler flag is replaced by -O (optimize for speed) and the variable **DEBUG** is not defined (therefore, no debugging support).

An alternative to invoking make with the default (no) options is to invoke make with the **debug** keyword, i.e.

```
cd (distribution_base_directory)/rpdocs/development/server
make debug
```

Table 3-2. CDOCS server build summary

Item	Attribute
Platform	SUN workstation
Operating System	SOLARIS Version 2.3
Compiler	SPARCompiler Version 3.0
Compiler Flags	-c -v -Xc -g -DDEBUG
Include File Dependencies	/usr/include/rpc
Library Dependencies	sql sqlnet ora nlsrtl cv6 core socket nsl m dl aio
Loader Flags	-g -xcg92

This is exactly the same as invoking make without arguments.

Executing make with the `install` keyword copies all of the server executables (compiled executables as well as shell scripts) and read.me files to the defined install directory (\$INS). One of the post-installation tasks for CDOCS is to change all of the makefiles' definitions of the install directory to match NRC's layout on the CDOCS server.

Finally, the `clean` keyword "touches" all of the source files, forcing a complete recompilation of all source files on the next invocation of make (as opposed to compiling only the changed files).

The server Makefile is shown in Appendix C.1. There are six items that should be illustrated (these items are called out as numbers in parenthesis at the right margin in the Makefile listing in Appendix C.1). The first item is the definition of the installation directory definition:

```
INS = /db3/runtime_executables/samson_solaris_installs/server
```

This must be updated following installation of CDOCS at NRC to reflect the directory structure of the server machine. The second is that support for the ORACLE Pro*C pre-compiler is coded in this section, including a compilation directive (`.pc.c:`) and include directories (`/usr/include/rpc` and `../rpc`). The third item concerns C compiler definitions (compiler to be used, compiler flags, include flags, and `.c.o:` compilation directive). The fourth item addresses library dependencies. It should be noted that the `PRCLIBS` and `CORLIBS` statements repeat the libraries, i.e.

```
PRCLIBS = -lsql -lsqlnet -lora -lsqlnet -lora
CORLIBS = -lnlsrtl -lcv6 -lcore -lnlsrtl -lcv6 -lcore
```

This is used to prevent "symbol not found" errors in the linking step. The fifth item is the definition of the makefile keywords (release, clean, debug, install) that may be invoked as part of the `make` command. The last item is the listing of all of the file dependencies of the server; it is this information that "drives" the entire make process and that determines whether an individual file is compiled base upon "last modified" date.

3.1.2 CDOCS Batch

The main CDOCS batch executable, *da_batch*, is compiled by changing to the batch distribution directory and running make against the supplied Makefile, i.e.

```
cd (distribution_base_directory)/rpdocs/development/batch
make
```

This compiles *da_batch* in debug mode, i.e. with support for interactive debugging using dbx or another related tool. Table 3-3 summarizes information from Makefile for standard (default) compilation.

Table 3-3. CDOCS batch build summary

Item	Attribute
Platform	SUN workstation
Operating System	SOLARIS Version 2.3
Compiler	SPARCompiler Version 3.0
Compiler Flags	-c -v -Xc -g -DDEBUG
Include File Dependencies	/usr/include/rpc
Library Dependencies	sql sqlnet ora nlsrtl cv6 core socket nsl m dl aio
Loader Flags	-g -xcg92

By contrast, a release version without debugging support and optimized for speed is produced by adding the **release** keyword to make, i.e.

```
cd (distribution_base_directory)/rpdocs/development/batch
make release
```

The **-g** compiler flag is replaced by **-O** (optimize for speed) and the variable **DEBUG** is not defined (therefore, no debugging support).

An alternative to invoking make with the default (no) options is to invoke make with the **debug** keyword, i.e.

```
cd (distribution_base_directory)/rpdocs/development/batch
make debug
```

This is exactly the same as invoking make without arguments.

Executing make with the **install** keyword copies all of the batch executables (compiled executables as well as shell scripts) and read.me files to the defined install directory (\$INS). One of the

post-installation tasks for CDOCS is to change all of the makefiles' definitions of the install directory to match NRC's layout on the CDOCS server.

Finally, the `clean` keyword "touches" all of the source files, forcing a complete recompilation of all source files on the next invocation of `make` (as opposed to compiling only the changed files).

The batch Makefile is shown in Appendix C.2. Six items are of interest to the system developer (these items are called out as numbers in parenthesis at the right margin in the Makefile listing in Appendix C.1). The first item is the definition of the installation directory definition:

```
INS = /db3/runtime_executables/samson_solaris_installs/batch
```

This must be updated following installation of CDOCS at NRC to reflect the directory structure of the server machine. The second is that support for the ORACLE Pro*C pre-compiler is coded in this section, including a compilation directive (`.pc.c:`) and include directories (`/usr/include/rpc` and `../rpc`). The third item concerns C compiler definitions (compiler to be used, compiler flags, include flags, and `.c.o:` compilation directive). The fourth item addresses library dependencies. It should be noted that the `PRCLIBS` and `CORLIBS` statements repeat the libraries, i.e.

```
PRCLIBS = -lsql -lsqlnet -lora -lsqlnet -lora  
CORLIBS = -lnlsrtl -lcv6 -lcore -lnlsrtl -lcv6 -lcore
```

This is used to prevent "symbol not found" errors in the linking step. The fifth item is the definition of the makefile keywords (`release`, `clean`, `debug`, `install`) that may be invoked as part of the `make` command. The last item is the listing of all of the file dependencies of the server; it is this information that "drives" the entire make process and that determines whether an individual file is compiled base upon "last modified" date.

3.1.3 CDOCS Clients

3.1.3.1 General

There is much more variety between the makefiles used to build the client executable because of the differences in platform type. Even the name of the utility used to build the executable is not uniform (`make` on SUN OS and Solaris and `nmake` on Windows and OS/2).

3.1.3.2 Solaris

There are several items that need explanation in the makefile (see listing of items called out as numbers in parenthesis at the right margin in Appendix C.3). First, the environmental variable `GALAXYHOME` must be defined prior to running `make`. This environmental variables points to the installation point of galaxy for include and lib references in the build process. The second item concerns debugging. The symbol `vdebugDEBUG` is set causing a debug version to be built. The lib path correctly points to the galaxy debug library for proper completion of the link step. The third item concerns the `make install` keyword. As written, the `install` keyword for `make` causes installation to be performed into the directory structure at CNWRA. As part of post-installation tasks these lines need to be updated to represent the NRC installation directories. Table 3-4 summarizes information about the Solaris client make process.

Table 3-4. Solaris client make information

Item	Attribute
Platform	SUN workstation
Operating System	SOLARIS Version 2.3
Compiler	SPARCompiler Version 3.0
Compiler Flags	-D sun -g -c -v -Xc -xsb -DvdebugDEBUG=1
Include File Dependencies	/usr/include/rpc
Library Dependencies	ns1 socket vgalaxy-debug Xext X11 m
Loader Flags	N/A

3.1.3.3 SUN OS

The SUN OS Makefile does not differ from that of the Solaris Makefile. The same items discussed in Section 3.1.3.2 apply to the SUN OS build process as listed in Appendix C.4. Table 3-5 summarizes information about the Solaris client make process.

3.1.3.4 Windows

Client Makefile

The Windows 3.1 client was developed using the Watcom C/C++ compiler version 9.5. Watcom's make is named wmake. Appendix C.5.1 lists the makefile for the client. Note that this is a simple makefile which includes an environment file (see Appendix C.5.2) and then lists all the object modules that need to be produced. All of the intelligence of this makefile is contained within the environ file which checks the platform type and environmental variables to set compiler flags, library paths, etc. to ensure a successful build. The environ file was supplied by Visix Corporation with the Galaxy for Windows development package. Table 3-6 summarizes the Windows make process.

Client Build Batch File

Appendix C.5.3 lists a simple batch file that invokes wmake to rebuild the client. This batch file is named make, so that the client can be built by issuing the command *make*.

Makefile for rpc16.dll

In addition to the Windows client module there is a module which interfaces the client, written in 32-bit code using the Win32s functions, to the RPC communications module which is written in 16-bit code. This interface code is stored in a Dynamic Load Library (DLL) and functions are loaded as called. The only purpose of this interface code is to “thunk” between 16-bit and 32-bit, i.e. translate segmented architecture addresses to flat memory model addresses.

Table 3-5. SUN OS client make information

Item	Attribute
Platform	SUN workstation
Operating System	SOLARIS Version 2.3
Compiler	acc
Compiler Flags	-D sun -g -c -v -Xc -xsb -DvdebugDEBUG=1
Include File Dependencies	/usr/include/rpc
Library Dependencies	ns1 socket vgalaxy-debug Xext X11 m
Loader Flags	N/A

Table 3-6. Windows 3.1 client make information

Item	Attribute
Platform	PC
Operating System	Windows 3.1
Compiler	Watcom Version 9.5
Compiler Flags	(see !environ)
Include File Dependencies	(see !environ)
Library Dependencies	(see !environ)
Loader Flags	(see !environ)

This dll was built using the Microsoft C/C++ Version 7.0 development environment. Microsoft's make is named `nmake`; therefore, to build the dll the command `nmake -f makefile.win` would be issued. Table 3-7 summarizes the makefile for `rpc16.dll`.

Table 3-7. rpc16.dll make information

Item	Attribute
Platform	PC
Operating System	Windows 3.1
Compiler	Microsoft C/C++ Version 7
Compiler Flags	-c -W3 -Asw -G2sw -Od -Zp -Zi -DDEBUG -D_WATCOMC__ -D_MSC__ -D_FTP__
Include File Dependencies	
Library Dependencies	libw sdllcew rpc4win
Loader Flags	/align:16 /CO /LI /MAP /NOD /NOE

3.1.3.5 OS/2

OS/2 Client Build Process

To build the OS/2 CDOCS client, certain support packages must be installed on the development machine. These are packages are listed in Table 3-8.

Table 3-8. OS/2 development packages

Package	Version
IBM Cset++	2.1
TCP/IP for OS/2 Base	2.0
TCP/IP for OS/2 Programmer's Toolkit	2.0
Galaxy for OS/2	2.5

IBM Cset++ provides the C compiler for the OS/2 environment, as well as the OS/2 programmer's toolkit (include files and libraries for OS/2 functions).TCP/IP Base provides basic TCP/IP functions including RPC run-time support. The TCP/IP Programmer's toolkit provides RPC functions, including XDR routines for converting data to eXternal Data Representation. Finally, Galaxy provides the common API used to design the GUI and many system calls used by the client. Table 3-9 summarizes information about the OS/2 build process.

Table 3-9. OS/2 client make information

Item	Attribute
Platform	PC
Operating System	OS/2 Version 2.1
Compiler	IBM CSet++
Compiler Flags	/Ss+ /Q+ /c /Gm+ /Sm /Gt /Ti -DvportDEBUG=0
Include File Dependencies	
Library Dependencies	vg3cosa os2386 so32dll rpc32dll tcp32dll dde4sbs
Loader Flags	/B"/STACK:0xffff" /B"/PM:PM"

Appendix C.6.1 lists the makefile used to build main.exe, the client executable. Items called out as numbers in parenthesis at the right margin within the makefile are references to the items discussed below for the makefile contents. At item 1, the Galaxy include directory is defined in reference to the environment variable \$GALAXYHOME. This variable should be set with a SET GALAXYHOME = {INSTALL_DIRECTORY} statement in the CONFIG.SYS file at boot-up time. The second item defines various sets of compiler switches that may be chosen to alter the results of the compilation. Options include producing a debug version and producing a compiler listing. Item 3 defines the galaxy libraries that will be linked at link-edit time. At item 4, options to the linker are defined. These include setting the stack size to be xfff bytes long and defining the output program to be a Presentation Manager application. In Item 5 additional compiler options and libraries are defined to provide RPC function support for the TCP/IP communications functions. In item 6 the compiler is defined. Item 7 defines the resource editor call (the resource editor is used to compile a binary resource file from an ascii file and to bind that resource file to the executable. This allows the association of an icon to an application.) Item 8 invokes the linker, passing the name of the response file rpd.l to provide link edit information. Finally, at item 9 the resource compiler is again invoked to bind the binary resource file to the executable file.

Make Batch File

Appendix C.6.2 lists *make.cmd*, a short OS/2 command file that builds the client. The client is built by issuing the command *make* which invokes *nmake*, the OS/2 Cset++ make program, passing it the -f rpd.mak argument and producing a compiler listing in the file *nmake.lst*.

Linker Response File

The linker response file contains all of the information necessary for link386, the OS/2 linker, to produce an executable file from the object modules output from the compiler. It is listed in appendix C.6.3.

Startjp Helper Application

The startjp helper application is an application that starts up PMJPEG, the image viewer for OS/2. The build command file, the makefile and the linker response file are listed in appendices C.6.4, C.6.5, and C.6.6, respectively.

Startwp Helper Application

The startwp helper application is an application that starts up PMJPEG, the image viewer for OS/2. The build command file, the makefile and the linker response file are listed in appendices C.6.4, C.6.5, and C.6.6, respectively.

3.1.3.6 Macintosh

The Apple Macintosh client is built with the Symantec Think-C development environment. This environment provides an easy to use GUI project manager which replaces the functionality of a make-driven build process on other platforms. Appendix C.7.1 contains a report from the project manager listing the segments that make up the client module and information about those segments. Table 3-10 summarizes Macintosh build processes.

Table 3-10. Macintosh client make information

Item	Attribute
Platform	Macintosh Quadra
Operating System	System 7
Compiler	Symantec Think-C
Compiler Flags	(Menu-driven)
Include File Dependencies	(Menu-driven)
Library Dependencies	(Menu-driven)
Loader Flags	(Menu-driven)

3.2 EXECUTION

Running of individual CDOCS executables is discussed in Chapter 4 as described in the individual readme files.

4 INDEXED README FILES

Many CDOCS system-specific directions, operational considerations, and other useful information are summarized in a set of readme files. These files are listed in their entirety in Section 4.2. Section 4.1 provides an index to finding information in these files.

4.1 READ.ME FILES INDEX

Table 4.1 Read.me index

Topics	File	Section
Batch Process Parameters	howstart.bat	4.2.1
Batch Server Files	readme.bat	4.2.5
CDOCS Client	readme.clt	4.2.6
Client Environment Variables	readme.clt	4.2.6
Client Files	readme.clt	4.2.6
Clients Lock-up System	howstart.sql	4.2.3
Help System	howstart.hlp; readme.hlp	4.2.2; 4.2.7
Help System Database	readme.hlp	4.2.7
Initializing Server	readme.svr	4.2.9
Killing Server Process	howstart.svr	4.2.4
Macintosh Client	readme.mac	4.2.8
Making Batch Server	readme.bat	4.2.5
Making Help System	readme.hlp	4.2.7
Making Server	readme.svr	4.2.9
Running Batch Server	readme.bat	4.2.5
Running Client	readme.clt	4.2.6
Server Database	readme.svr	4.2.9
Server Files	readme.svr	4.2.9
Server SQL Scripts	readme.svr	4.2.9
Server Versions	readme.svr	4.2.9
SQL *Net Listener	howstart.sql	4.2.3

Table 4.1 Read.me index (cont'd)

Topics	File	Section
Starting Batch Process	howstart.bat	4.2.1
Starting Client	readme.clt	4.2.6
Starting Help System	howstart.hlp; readme.hlp	4.2.2; 4.2.7
Starting Macintosh Client	readme.mac	4.2.8
Starting Server	howstart.svr	4.2.4
Starting SQL *Net Listener	howstart.sql	4.2.3
Starting Windows Client	readme.win	4.2.10
Using Help System	readme.hlp	4.2.2
Windows Client	readme.win	4.2.10

4.2 READ.ME FILE LISTINGS

4.2.1 Read.Me on How to Start CDOCS Batch

Filename: howstart.bat

Use the following sequence of commands to start the CDOCS 2.0.5 (test), 2.0.6 (CNWRA production) and 2.0.7 (NRC production) batches.

```
samson% su - topic40
Password: xxxxxxx
samson% cd /db3/runtime_executables/samson_solaris_installs/server
samson% sh
$ da_batch <V> <D> <O># see below
```

Batch arguments are:

```
<V>      =   version
<D>      =   dbaccess string - uid/pwd@DB
DB        =   database name
<O>      =   operation, one of -
cdocs     =   cdocs batch (newest version needed, runs both tdocs and rpd batch - all
               partitions)
tdocs     =   tdocs batch (all tdocs partitions only)
rpd       =   rpd batch (all rpd partitions only)
rpt       =   report writer batch - generates canned reports
cds [#]   =   generate cds documents, optional document number
```

cdm [#] = generate cdm documents, optional document number
oits [#] = generate oits documents, optional document number

Explanation: (a) log in as topic40, dash required; (b) use topic40 password; (c) cd to the install directory; (d) report writer requires bourne shell.

Note that, depending on <D>, reports are stored in /u03/DB/reports/writer and documents are generated in directories, cds, cdm, or oits, under writer.

4.2.2 Read.Me on How to Start CDOCS Help System

Filename: howstart.hlp

Use the following commands to start the CDOCS help:

```
da_help <DBA> load <FILE>
da_help <DBA> list <FILE>
da_help <DBA> dump <FILE>
```

where <DBA> is rpdocs database access and <FILE> is a file name.

These three utility programs are used to edit/modify the following tables that hold the CDOCS user HELP SYSTEM:

RPDOCS_HELP - help headers and text
RPDOCS_HELP_INDEX - indexes help by button, subkey, and privilege
RPDOCS_PRIVILEGES - valid privilege

4.2.3 Read.Me on How to Run SQLNet

Filename: howstart.sql

Sometimes clients hang on login even though the server is up and running. What's likely happened is the ORACLE SQL*Net Listener has hung. Here's the simplest remedy:

samson% su - oracle	# su to oracle with -
Password: xxxxxxx	# oracle password
samson% ps -efl grep LISTENER	# get the listener's pid
samson% kill -9 <pid>	# kill it
samson% lsnrctl start	# restart it
samson% exit	

4.2.4 Read.Me on How to Start CDOCS Server

Filename: howstart.svr

Use the following sequence of commands to start the CDOCS 2.0.5 (test), 2.0.6 (CNWRA production) and 2.0.7 (NRC production) servers.

```
samson% su - topic40
Password: xxxxxxxx
samson% cd /db3/runtime_executables/samson_solaris_installs/server
samson% sh
$ restartda <V> # where <V> = 205, 206, or 207
```

Explanation: (a) log in as topic40, dash required; (b) use topic40 password; (c) cd to the install directory; (d) report writer requires bourne shell; (e) kills any running and starts server.

Version 2.0.5 serves as a test server, 2.0.6 serves the CNWRA CDOCS; version 2.0.7 serves the NRC CDOCS.

When starting either, heed immediate output messages concerning log files and success or failure of startup. If the server, single tasking in development, is hung you may have to use kill <pid>.

4.2.5 Read.Me for Making and Running the Batch Server

Filename: readme.bat

Making da_batch...

SOURCE: The batch source resides in /db3/rpdocs/development/batch and /db3/rpdocs/development/server (only sllist.h/c). A file listing can be found at the end of this file. Bring those over to your own work space.

MAKE: In order to make da_batch run “make” in batch. It is suggested, however, that “make debug” be used to make a debug version or “make release” to make a non-debug version of da_batch both of which force a complete remake (requires a few minutes).

RUNNING: In order to run a da_batch, su to topic40 and run da_batch <UID>/<PWD>@t:goliath:<DB> <SET>.

where UID and PWD are a dba's user id and password; DB is test1 or prod1; and, SET is as found in batch.h.

If you don't run as topic40 you'll get file permission problems.

INSTALL: Install either the debug or release version of the server to /db3/rpdocs/batch.

FILES: File listing for da_batch:

Makefile	-	makefile for batch
Makefile.debug	-	makefile for debug batch
Makefile.warnings	-	makefile for debug and warnings batch
READ.ME	-	this file
da_batch*	-	executable
batch.c	-	main module
batch.h	-	tdocs/rpd shared header
batchchk.pc	-	rpdc checkin batch
batchcsp.pc	-	tdocs correspondence batch
batchdb.h	-	tdocs/rpd shared database access header
batchdb.pc	-	tdocs/rpd shared database access
batchdef.pc	-	rpdc define batch
batchqa.pc	-	tdocs quality assurance batch
batchrpd.h	-	rpdc define, checkin shared header
batchrpd.pc	-	main rpd batch
batchrpt.c	-	tdocs/rpd shared batch report
batchrpt.h	-	tdocs/rpd shared batch report header
batchrw.pc	-	rpdc report writer
batchtdi.pc	-	tdocs technical document index batch
batchtdocs.h	-	tdocs csp, qa, tdi shared header
batchtdocs.pc	-	main tdocs batch
create_batch.sql	-	batch report table
pieces.h	-	rpdc pieces header
writer.h	-	rpdc writer header

Related utilities of this project (in /db3/rpdocs/batch):

pieces	-	pieces SGML docs to piece table
writer	-	writes pieces from piece table to SGML doc
wpchar	-	coverts SGML chars to WP chars

Related unix script and awk files (in /db1/prod1/tools):

rpdocs_delete_doc	-	topic delete script
rpdocs_dum_asc.awk	-	script to generate vacant ascii cds, cdm, and rps documents
rpdocs_dum_sgm.awk	-	script to generate vacant sgml cds, cdm, and rps documents
rpdocs_insert_doc	-	topic insert script
rpdocs_link_doc	-	topic ascii/WP link script
rpdocs_secure_doc	-	topic secure script
rpdocs_cds.awk	-	awks cds doc type, review plan, and title
rpdocs_cdm.awk	-	awks cdm doc type, review plan, and title
rpdocs_rps.awk	-	awks rps doc type, review plan, and title
rpdocs_retire.awk	-	awks new status and partition for retires

Related vacant rpd documents (in /db3/SGML/vacant):

CDM_DUM.ASC	-	vacant ascii cdm template
CDM_DUM.SGM	-	vacant sgml cdm template
CDS_DUM.ASC	-	vacant ascii cds template
CDS_DUM.SGM	-	vacant sgml cds template
RPS_DUM.ASC	-	vacant ascii rps template
RPS_DUM.SGM	-	vacant sgml rps template

Other related wp macro and sgml files:

ipx4/wp/tgbin/*.rul	-	sgml rule files
db3/SGML/dtd/[t]*.dtd	-	sgml dtids (t for test)
db3/SGML/macros/[t]*.wpm	-	wp macros (t for test)

This is not a complete list of dependencies, for there are more embedded in WordPerfect macros and possibly other report writer modules.

4.2.6 Read.Me for Client Code for All Platforms (SUN OS, Solaris, Windows, OS/2, Macintosh)

Filename: readme.clt

The following set of files make up the CDOCS client for all platforms.

(Files located on the SUN in ../rpdocs/development/interface)

check.c checkin.c circulat.c cleanup.c common.c contain.c
ctest.c define.c edit.c efilter.c etcbutt.c file.c filechsr.c
help.c init.c intern.c label.c launch.c listview.c logon.c main.c
maint.c memory.c mosaic.c options.c prefs.c printer.c report.c
scan.c search.c submit.c subset.c svc.c tdocs.c tdocserr.c utils.c vrpc.c

checkin.h common.h define.h docdb.h globals.h help.h hitlist.h
init.h intern.h logon.h main.h maint.h options.h pref.h
prefs.h report.h search.h submit.h svc.h tdocs.h vrpc.h

(Files located on the SUN in ../rpdocs/development/rpc)

client.c clntrpc.c da_clnt.c da_xdr.c
client.h clntrpc.h da.h dadefs.h

Note: launch.c and printer.c are used only on the Macintosh.

Additionally, the file main.vr is the resource file for the CDOCS client. The files intern.c and intern.h are the C source files that correspond to this resource file and are generated through Galaxy's VRE (Visual Resource Editor.)

This client is compiled with the native C compiler on the SUN (cc). The directory structure is as follows:

```
{USERHOME}/rpdocs/development/rpc  
{USERHOME}/rpdocs/development/interface
```

where {USERHOME} is the developer's home directory (e.g. /home/samson/dlincoln)

The makefile and the resource file (main.vr) must reside in:

```
{USERHOME}/rpdocs/development
```

To create the client for SUN OS or SOLARIS platforms, run the make utility against the makefile 'Makefile' in the directory:

```
{USERHOME}/rpdocs/development
```

To create the CDOCS client for the WINDOWS platform:

- (1) Make sure these environment variables have been set:

```
SET PATH      = C:\WATCOM\BIN;C:\WATCOM\BINB;C:\WATCOM\BINW  
SET LIB       = C:\WATCOM\LIB386;C:\WATCOM\LIB386\WIN  
SET INCLUDE   = C:\GALAXY\INCLUDE;C:\WATCOM\H;C:\WATCOM\H\WIN  
SET WATCOM    = C:\watcom
```

- (2) Ensure that all of the source files listed above and the make.bat file are located in the directory in which the executable is to be created.
- (3) Run the make.bat batch file.

To run the CDOCS client:

- (1) Ensure the resource file (main.vr) is in the same directory as the executable.
- (2) Create a program item whose:
 - (a) Command line = {APPDIR}\main.exe -nonativechoosers where {APPDIR} is the application's directory path (e.g. D:\cdocs)
 - (b) Working Directory = Working directory (e.g. C:\cdocs)

4.2.7 Read.Me for Help System

Filename: readme.hlp

Contents

This file describes help configuration management, make, and execution.

Configuration Management

The help source resides in workspace (WS)

WS/help

and

WS/help/ASCII

A file listing can be found at the end of this file.

Make

In order to make da_help run “make” in the WS/help directory and da_help_io the WS/help/ASCII directory.

In order to install run “make install” in both places.

Execution

Run da_help by entering:

```
da_help -laf <GUI>
```

where <GUI> is motif or openlook.

Click the Login button. Whom you log in as, will determine which database you access. Use one of the following:

- (1) for test1, rpdocs/ichiban@test1
- (2) for prod1, rpdocs/ichiban@prod1

You may then insert, update and delete records. Any changes are automatically stored in the appropriate database.

The Import and Export utilities will normally NOT be used. Import is used to load a pre-formatted file into the database. Export is used to generate an ASCII file of the contents of the database. Import looks for a file named help.imp. Export generates a file named help.exp.

Run da_help_io in various ways:

```
da_help <DBA> load <FILE>
da_help <DBA> list > <FILE>
da_help <DBA> dump > <FILE>
```

where <DBA> is rpdocs database access and <FILE> is some file. Briefly, the load file requires each record to be formatted as follows:

```
<<<PRIV <comma-separated list of privileges>
<<<KEY <GUI key - usually a button name>
<<<SUBKEY <currently only 'None' is supported>
<<<SORT <currently not used>
<<<HEADER <multiline header, carriage returns removed>
<<<TEXT <multiline help text, only double carriage returns kept for paragraphs>
```

Load will reject records that are duplicates, those whose subkey is not 'None,' and those for which there are no corresponding privilege. These errors are printed to standard out.

Database

The following tables are used:

RPDOCS_HELP	-	help headers and text
RPDOCS_HELP_INDEX	-	indexes help by button, subkey, and privilege
RPDOCS_PRIVILEGES	-	valid privilege

See the sql script rpdocs_help_create.sql in the workspace directory db.

File List

In WS/help

max.h	-	defines fields widths and other maximums
db.h	-	defines database access macros and prototypes
db.pc	-	implements database access
help.c	-	implements CDOCS help system
main.c	-	control loop
memory.c	-	allocate/deallocate dynamic memory
login.c	-	implement user login and menu control
list.c	-	controls list maintenance
update.c	-	implements CDOCS document updates
insert.c	-	implements CDOCS document inserts
delete.c	-	implements CDOCS document delete

In WS/help/ASCII

max.h	-	defines fields widths and other maximums
rec.h	-	defines a help record (joins rpdocs_help and rpdocs_help_index)
db.h	-	defines database access macros and prototypes
db.pc	-	implements database access
io.h	-	defines input/output macros and prototypes
io.pc	-	implements input/output

4.2.8 Read.Me for Macintosh-Specific Applications for CDOCS

Filename: readme.mac

(Files located in ../rpdocs/development/macserver)

(See READ.ME.CLIENT to learn how to create the CDOCS client for MAC)

The following set of files make up the MACINTOSH intermediary server (known as macserver) for CDOCS:

- (1) connect.c
- (2) macclient.c
- (3) macserver.c
- (4) request.c
- (5) connect.h

In addition, the file 'environ' is used by the makefile. Run UNIX's make in the working directory.

To start the MAC server, type:

```
macserver <DBNAME> &
```

where DBNAME is the production database name.

4.2.9 Read.Me for Making and Running the Server

Filename: readme.Svr

Contents

This file describes server configuration management, make, versions, and execution.

Configuration Management

The server source resides in several places in the parent workspace (WS):

WS/development/server	-	main module
WS/development/kill	-	server kill utility
WS/development/rpc	-	rpc module

A file listing can be found at the end of this file. Because the parent WS contains many versions, bring only those files listed over to your own work space.

Make

In order to make the server, run “make” in the WS/server directory.

For debugging run “make debug.” And for a release version run “make release.” Both makes force a complete remake.

In order to install the server in its proper place, run “make install.”

Versions

The server is versioned as follows: if the client is at version major.minor then the server is at major.minor.subversion. For example, if the client is at 2.0 then the server is at 2.0 with possible subversions 0-9, i.e., 2.0.0, 2.0.1, etc. This is necessary to ensure that the client and server requests are synchronized and that the server can validate and log those requests. Versions are set at runtime.

Execution

In order to run a version of the da_server, su to topic40

```
su - topic40      # the dash is required
sh                # boerne shell required
```

and run

```
da_server <VER> <UID>/<PWD>@<DB>
```

where VER is the version number (e.g., 205, 206, etc.), UID and PWD are a dba's user id and password, DB is test1 or prod1. Remember, 205 is the test server and 206 is the production server which is used for the operational system at the CNWRA, and 207 is the production server which is used for the operational system at the NRC.

If you don't run as topic40 you'll get file permission problems. It's not significant which database you log into just that you log in. All transaction requests, inputs, and results are logged in /db3/run*/sam*/server/svc_log in files identified by server version and start up date.

```
/etc/rc.local lines should read
su oracle -c /usr/oracle/bin/dbstart;\
su - topic40 -c '/db3/run*/sam*/server/da_server <VER> <UID>/<PWD>@<DB>';\
```

with an su to topic40 for each <VER> that needs to run.

Initialization File

The server needs to be able to find an initialization file, da_svc_<VER>.ini, in its start up directory. This file provides information on where to find TOPIC databases, SGML data, and so on. It also determines whether the server runs in multitasking or singletasking mode, and whether TDOCS synchronization is turned on.

When debugging turn multitasking and synchronization off. Both involve forks the debugger cannot follow.

Database

The following tables are accessed:

RPDOCS_USERS	-	userids and privileges
RPDOCS_NAMES	-	user names
RPDOCS_PRIVILEGES	-	privilege descriptions
RPDOCS_HELP	-	help headers and text
RPDOCS_HELP_INDEX	-	help index
PEOPLE	-	CNWRA names
TDOCS	-	tdocs headers
TDOCS_ADDRESSEES	-	tdocs addressees
TDOCS_AUTHORS	-	tdocs authors
TDOCS_ASSIGNED_CODES	-	tdocs assigned codes
TDOCS_ASSIGNED_NUMBERS	-	tdocs assigned numbers
TDOCS_CIRCULATION	-	tdocs circulation
TDOCS_REPORTS	-	tdocs reports
TDOCS_DOCUMENT_SETS	-	tdocs document set data
TDOCS_DOCUMENT_STATUS	-	tdocs document status description
TDOCS_HYDRO	-	hydrology records
TDOCS_NIST	-	NIST records
TDOCS_NIST_ABSTRACTS	-	NIST abstracts
TDOCS_NIST_KEYWORDS	-	NIST keywords
TDOCS_NIST_REVIEWS	-	NIST reviews
TDOCS_NUREG	-	demo
TDOCS_REP	-	demo
RPD	-	rpdoc records
RPD_DOCUMENT_SETS	-	rpdoc document sets
RPD_REPORTS	-	rpdoc report writer reports
RPD_PIECES	-	rpdoc report writer pieces

See the following sql scripts in the db workspace directory:

rpdocs_users_create.sql	-	drop and create
rpdocs_help_create.sql	-	drop and create
misc_people_create.sql	-	drop and create
tdocs_1st_create.sql	-	} ordered create
tdocs_2nd_create.sql	-	
tdocs_3rd_create.sql	-	
tdocs_docset_insert.sql	-	document set data
tdocs_major_drop.sql	-	ordered drop
tdocs_minor_drop.sql	-	ordered drop
tdocs_hydro_create.sql	-	create
tdocs_nist_create.sql	-	create
tdocs_nureg_create.sql	-	create

tdocs_rep_create.sql	-	create
rpd_1st_create.sql	-	} ordered create
rpd_2nd_create.sql	-	
rpd_3rd_create.sql	-	
rpd_all_drop.sql	-	order drop
rpd_docset_insert.sql	-	document set data

The TDOCS_DOCUMENT_SETS table is used by the server to determine which document sets are visible to a particular site. When the client does an RPC request for a list of site-specific document sets, the server sends a SQL request against this table to return those sets with the following table fields set as shown:

- (1) location = 'LOCAL'
- (2) loadable = 'Y'

For example, the CNWRA might have the following sets defined:

<u>DOCUMENT SET</u>	<u>L LOCATI</u>
CTD	Y LOCAL
NTD	Y REMOTE

whereas the NRC might have them defined as:

<u>DOCUMENT SET</u>	<u>L LOCATI</u>
CTD	Y REMOTE
NTD	Y LOCAL

File List

File listing for da server (\$S=WS/development/server, \$R=WS/development/rpc):

\$S/READ.ME.server	-	this file
\$S/Makefile	-	makefile
\$S/da_svc_*.ini	-	initialization file
\$S/da_svc_proc.c	-	service procedure
\$S/dat.c	-	packet decoding, results encoding
\$S/dat.h	-	packet decoding, results encoding
\$S/db.h	-	general database header
\$S/max.h	-	database column lengths
\$S/db.pc	-	database VARCHARS
\$S/doc.h	-	file read and write header
\$S/doc.c	-	file read and write functions
\$S/env.h	-	header for ini file reader
\$S/env.c	-	ini file reader
\$S/log.h	-	transaction logging header

\$S/log.c	-	transaction logging functions
\$S/mem.h	-	memory header
\$S/mem.c	-	memory functions
\$S/slist.h	-	header for single link list
\$S/slist.c	-	"generic" single link list
\$S/usr.pc	-	rpdocs user functions
\$S/rpt.h	-	header for "generic" reports
\$S/rpt.c	-	"generic" reports
\$S/rpd.h	-	rpdocs header
\$S/rpd.pc	-	rpdocs general database
\$S/rpdchk.pc	-	rpdocs checkin
\$S/rpddef.pc	-	rpdocs definition
\$S/rpddtd.h	-	header for rpdocs SQL*DTD
\$S/rpddtd.pc	-	rpdocs SQL*DTD
\$S/rpddoc.pc	-	rpdocs documents
\$S/rpdret.pc	-	rpdocs retire
\$S/rpdrpt.h	-	header for rpdocs reports
\$S/rpdrpt.pc	-	rpdocs reports
\$S/rpdwr.h	-	header for rpdocs report writer
\$S/rpdwr.pc	-	rpdocs report writer
\$S/rpdtop.pc	-	rpdocs topic
\$S/tdocs.h	-	tdocs header
\$S/tdocschk.pc	-	tdocs checkin/checkout
\$S/tdocsdat.pc	-	tdocs data
\$S/tdocsdel.pc	-	tdocs deletion
\$S/tdocsdoc.h	-	header for tdocs files
\$S/tdocsdoc.pc	-	tdocs files
\$S/tdocshyd.h	-	header for tdocs hydrology
\$S/tdocshyd.pc	-	tdocs hydrology
\$S/tdocslbl.h	-	header for tdocs labels
\$S/tdocslbl.pc	-	tdocs labels
\$S/tdocsnms.pc	-	tdocs names
\$S/tdocsnst.h	-	header for tdocs NIST
\$S/tdocsnst.pc	-	tdocs NIST
\$S/tdocsrec.pc	-	tdocs document records
\$S/tdocsrpt.h	-	header for tdocs reports
\$S/tdocsrpt.pc	-	tdocs reports
\$S/tdocssub.pc	-	tdocs submission
\$S/tdocsin.pc	-	tdocs sync in
\$S/tdocsout.pc	-	tdocs sync out
\$S/tdocsupd.pc	-	tdocs update
\$R/da_svc.c	-	server main
\$R/da_xdr.c	-	server xdr
\$R/da.h	-	rpc definitions
\$R/dadefs.h	-	request definitions

Related unix script and awk files:

See readme.bat.

4.2.10 Read.Me for Windows-Specific Applications for CDOCS

Filename: readme.win

(Files located in ../rpdocs/development/win)

(See READ.ME.CLIENT to learn how to create the CDOCS client for WINDOWS)

The following set of files make up the WINDOWS print driver for CDOCS:

- (1) print.c - winprint source code
- (2) winprint.c - winprint source code
- (3) winprint.rc - winprint resource file
- (4) winprint.mak - windows makefile
- (5) make.bat - winprint make batch file

To create the winprint.exe executable, make sure these environment variables have been set:

```
SET PATH      = C:\WATCOM\BIN;C:\WATCOM\BIN;C:\WATCOM\BINW
SET LIB       = C:\WATCOM\LIB386;C:\WATCOM\LIB386\WIN
SET INCLUDE   = C:\GALAXY\INCLUDE;C:\WATCOM\H;C:\WATCOM\H\WIN
SET WATCOM    = C:\watcom
```

Run the make.bat batch file.

The next set of files make up an executable that TOPIC uses to launch WordPerfect and the user-selected viewer as defined in the preferences dialog in the CDOCS WINDOWS client. This set of code was compiled using the Borland C 4.0 compiler.

- (1) startwp.c - startwp.exe source code
- (2) startwp.ide - borland C project file
- (3) startwp.def - windows definition file

To create the startwp.exe executable, load the project file into Borland's project manager and run make.

The next set of files make up the CDOCS16 DLL that the CDOCS WINDOWS client uses to translate data between 32-bit and 16-bit. This set of code was compiled using the Microsoft 7.0 C compiler.

- (1) cdocs16.c - cdocs16.dll source code
- (2) cdocs16.def - windows definition file
- (3) makefile.win - windows makefile

In addition, the following files must be present to compile the code:

(Files located in ../rpdocs/development/rpc)

- (1) davers.h - rpc version definitions
- (2) da.h - rpc definitions
- (3) dadefs.h - rpdocs defines
- (4) client.h - port client side rpdocs defines
- (5) clntrpc.h - non-portable client side rpdocs defines

Make sure these environment variables have been set:

```
set LIB      = c:\c700\lib;c:\pctcp\lib
set INCLUDE  = c:\c700\include;c:\pctcp\include
```

Run Microsoft C7.0's nmake from the dos command line in the working directory as:

```
nmake - fmakefile.win
```

Note: Make will copy cdocs16.dll to c:\windows.

5 INSTALLATION

5.1 OVERVIEW

This chapter contains the installation instructions for CDOCS on three different hardware/software environments.

- Microsoft Windows using IBM PS/2 or compatible hardware
- OPEN LOOK or MOTIF using Sun IPX hardware
- IBM OS/2 using IBM PS/2 compatible hardware

Although the specific steps required to install the CDOCS on a workstation varies for each of the platforms—the concept of what is required for installation can be described by four major actions. They are:

1. Verify that the specific workstation meets the hardware and software prerequisites as defined in Section 5.2.1, 5.3.1, or 5.4.1.
2. Provide workstation access to the TOPIC client executables. The TOPIC client executables require approximately 11 Mb of disk space. They may be installed on:
 - The workstation's local drive.
 - A file server.
 - The database server.
3. Provide workstation access to the CDOCS executables. The CDOCS executable files require approximately 2 Mb of disk space. The CDOCS executables may be installed on the workstation's local drive or on a file server. If the CDOCS executables are installed on a file server, then approximately 400 Kb of local disk space is required for the local CDOCS access files (such as the user's preference file).
4. Setup the workstation's environment. A specific directory structure (`<drive>:\docs\utopic`) must be created on the workstation (where `<drive>` is any local drive) and access to the database server must be provided via specified network file system (NFS) “mounts.” If either CDOCS or TOPIC is installed on a file server then access to that file server must be provided. A preference file must also be created for the specific arrangement implemented.

Sections 5.2 through 5.5 document the specific actions required to accomplish these steps on each of the supported platforms.

The diagram in Figure 5-1 represents a sample layout for the CDOCS installation. As is shown, the CDOCS and TOPIC executables may reside on the workstation, or on a file server, or they may be installed on the Sun Server. Regardless of where the executables are installed, the workstation must have program items and paths pointing to the executables. If the executables are installed on a server, the workstation's environment must be setup to access the network drive.

In addition to paths to the CDOCS, TOPIC, and viewer executables, the workstation must also have access to the Sun Server for a NFS mount.

The workstation must also have a specific local directory structure (with a path to it) for storing of documents, saved queries, and the user preferences file. This directory structure is always <drive>:\docs\utopic.

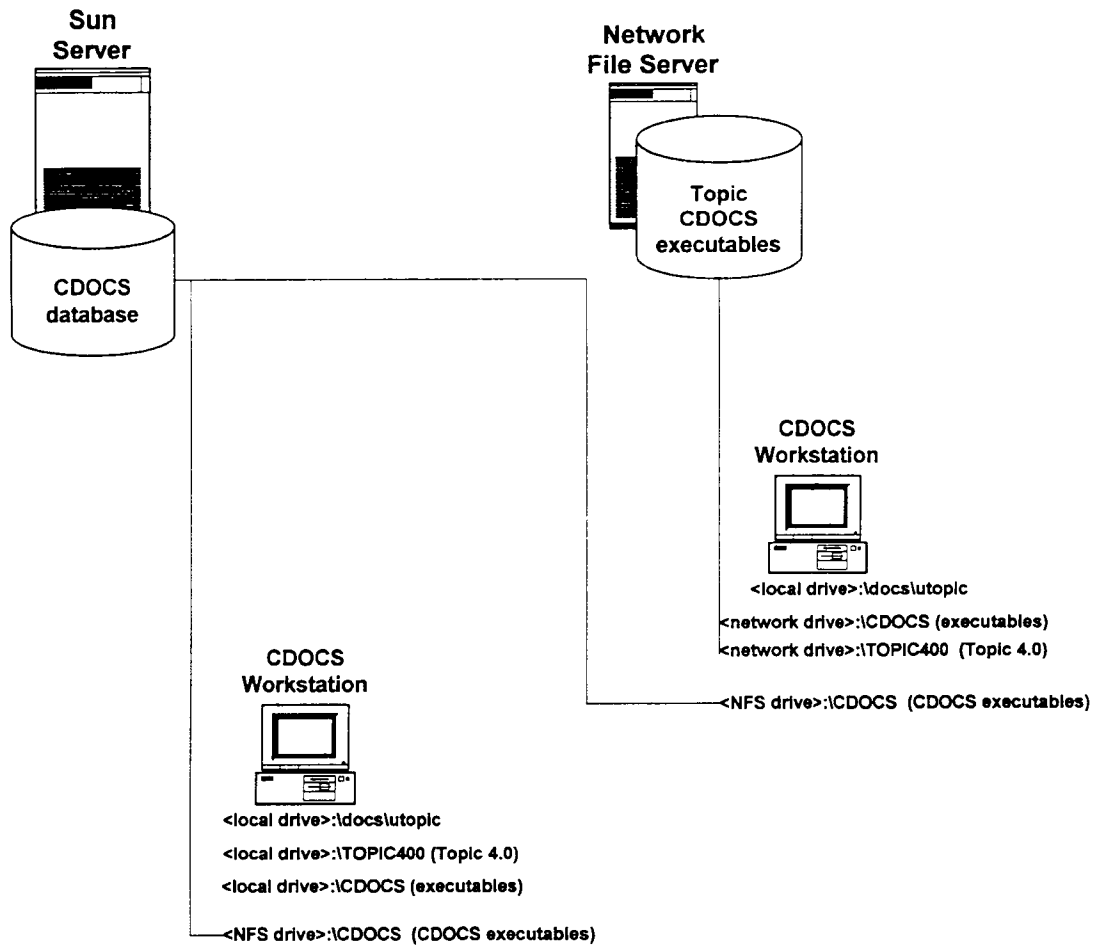


Figure 5-1. Sample installation layout

5.2 INSTALLATION ON A WINDOWS WORKSTATION

Sections 5.2.1 through 5.2.5, and Section 5.5, provide instructions for installing the CDOCS on a Windows workstation.

5.2.1 Ensure That the Workstation Meets the Minimum Hardware and Software Requirements for a Windows Workstation

Verify the workstation meets, or exceeds, the minimum hardware and software requirements:

- Intel 80486 CPU-based computer
- 8 MB Random Access Memory (RAM)
- 10 MB free hard disk space
- Microsoft Windows version 3.1
- PC/TCP Version 3.0 (also known as OnNet Version 1.1) by FTP Software

5.2.2 Install TOPIC 4.0

TOPIC's client executables may be installed on the workstation's local hard drive or on a connected file server. If TOPIC is installed on a file server, then skip to Section 5.2.3. Approximately 11 Mb of local disk space is required to install TOPIC on the workstation.

TOPIC arrives on CD ROM media. The CD ROM that contains the client executables has a label that includes *Text Retrieval System, Version 4.0.0* and *WinTDE*. If the installation machine has a CD drive, TOPIC may be installed directly from the CD. If the installation machine does not have a CD drive, then installation diskettes must be created from the CD. There are eighteen diskette image directories on the CD. These diskette image directories are under the following CD directory:

```
<cd drive>:\topic400\_dos\
```

To create TOPIC installation diskettes, copy the contents of each CD diskette image directory onto a diskette (DISK1, DISK2... through DISK18).

To install TOPIC:

1. Start Windows.
2. If installing from diskette, run **SETUP.EXE** from DISK1.

If installing from the CD, run **SETUP.EXE** from the following CD directory:

```
<cd drive>:\topic400\_dos\disk1
```


3. When prompted, select the drive and directory for TOPIC.
4. When prompted to choose between Custom or Complete installation, select *Custom Install*.
5. Select installation options as shown in Figure 5-2.
6. When all of the files have been installed, select to *Run* the licensing program.
7. TOPIC provides a LICENSE Sheet with the Site Id and SITE KEY. When prompted, enter the Site Id for *Site name* and the SITE KEY for *Key*.

TOPIC is now installed. The Windows desktop should have a group titled *Verity* and an icon in that group titled *WinTopic*.

Note: TOPIC requires a preferences file to run. If the *WinTopic* icon is double clicked at this time TOPIC will fail to run due to lack of the preference file. The CDOCS installation process will create a TOPIC preference file.

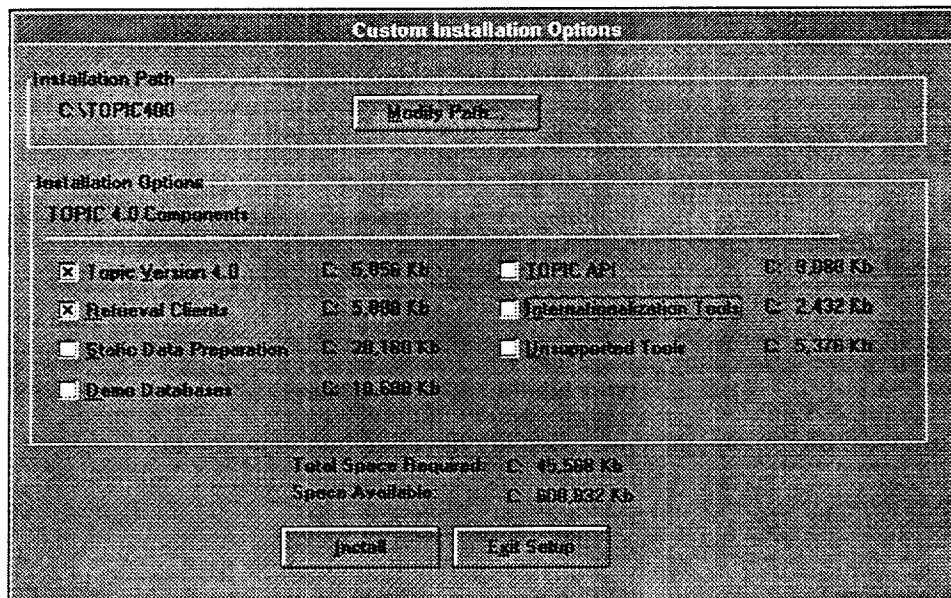


Figure 5-2. TOPIC Custom Installation Options screen

5.2.3 Install the CDOCS

The CDOCS executables may be installed on the workstation's local hard drive or they may reside on a connected file server (see Figure 5.1). The CDOCS executables require approximately 2 Mb of disk space. If the CDOCS executables reside on a file server, approximately 400 Kb of local disk space is required for installation of the CDOCS access files. The user will be importing documents and creating query files that will be stored on the local drive. Therefore, the local CDOCS files should be installed on the local drive that has the largest available free space. After the CDOCS files are installed the workstation must be setup to run the CDOCS (see Section 5.2.4).

5.2.3.1 To Install CDOCS on a Windows Workstation:

1. Obtain the following files from your system administrator.

`topic31.pwd`

`main.exe`

`vgalaxy2.vr`

`rpd16.dll`

`pctcp.dll`

`startwp.exe`

2. Create a directory for CDOCS.

Example: `<drive>:\md cdocs`

3. Copy `topic31.pwd` into the following TOPIC directory:

`<drive>:\topic400_dos\bin\`

4. Copy the remaining files into the directory created in Step 2.

5.2.3.2 To Install the CDOCS Files Required for a Windows Workstation on a Connected File Server

1. Obtain the following CDOCS files from your system administrator.

`topic31.pwd`

`vgalaxy2.vr`

2. Create a directory for CDOCS.

Example: `<drive>:\md cdocs`

3. Copy **topic31.pwd** into the following TOPIC directory:

`<drive>:\topic400_dos\bin\`

4. Copy **vgalaxy2.vr** into the directory created in Step 2.

5.2.4 Setup of Windows Workstation Environment

This procedure will create a specified directory structure on the workstation (`<drive>:\docs\utopic`). These directories will be utilized to store the user's documents, saved TOPIC queries, and the user's Preference File.

1. Create the directory `\docs` on the workstation's local drive.

Example: `C:\>md docs`

2. Create the subdirectory `\docs\utopics` on the local drive.

Example: `C:\docs>md utopics`

3. Create a new program item in Windows for the CDOCS, using the following information. If the CDOCS is installed on the local workstation, then `<path>` is the path to the directory into which the CDOCS files were copied in Section 5.2.3.1 Step 4. If the CDOCS is installed on a connected file server, then `<path>` is the network path to the file server directory where the CDOCS executables reside.

Description = CDOCS

Command Line = `<path>\main.exe`

Working Directory = `<drive>:\docs`

4. If TOPIC was installed on a file server instead of a workstation local drive, then create a new program item in Windows for TOPIC, using the following information. `<drive>` is the local drive where TOPIC was installed in Section 5.2.2.

Description = TOPIC

Command Line = **wintopic.exe**

Working Directory = `<drive>:\topic400_dos\bin`

5. Add the following set of statements to your **autoexec.bat** file:

— `set GALAXYHOME=<drive>:\cdocs`

— `set HOME=<drive>:\cdocs`

6. Add the following paths to the path statement in your **autoexec.bat** file:
 - <drive where TOPIC is installed>:\topic400_dos\bin
 - <drive where TOPIC is installed>:\topic400_dos\lib
 - <drive where CDOCS is installed>:\<directory where CDOCS is installed>
6. Setup the user preference file (reference Section 5.5).

5.2.5 Windows Workstation Installation Summary

The following checklist may be used as a final quick-check to verify that all of the actions necessary for installing CDOCS on a Windows workstation have been completed.

Table 5-1. Windows workstation installation summary list

Step	Page	Action
1	5-3	Verify workstation meets, or exceeds hardware/software prerequisites
2	5-3	Install TOPIC
3	5-5	Create a directory for the local CDOCS files
4a	5-5	If installing CDOCS on the local workstation, then copy the following files into the directory created in Step 3 main.exe vgalxy2.vr rpd.dll pctcp.dll start.wp
4b	5-5	If CDOCS is installed on a connected file server, then copy vgalaxy2.vr into the directory created in Step 3
5	5-6	Copy topic31.pwd into <drive>:\topic400_dos\bin
6	5-6	Create <drive>:\docs\utopic directory structure on workstation
7	5-6	Ensure workstation has a CDOCS program item (main.exe, working directory of <drive>:\cdocs)
8	5-6	Ensure workstation has a Topic program item (wintopic.exe, working directory of <drive>:\topic400_dos\bin)

Table 5-1. Windows workstation installation summary list (cont'd)

Step	Page	Action
9	5-6	Ensure workstation's <code>autoexec.bat</code> path statements includes entries for: <code><drive>:\cdocs</code> <code><drive>:\docs</code>
9	5-6	<code><drive>:\docs\utopic</code> <code><drive>:\topic400_dos\bin</code> <code><drive>:\topic400_dos\lib</code> <code><drive>:\<path></code> to directory where CDOCS executables are installed <code>set Galaxy HOME=<drive>:\cdocs</code> <code>set HOME=<drive>:\cdocs</code>
10	5-18	Ensure the User Preference File has been setup for the following: path to WordPerfect executable path to TOPIC path to Viewer path to CDOCS Server

5.3 INSTALLATION ON A SUN WORKSTATION

Sections 5.3.1 through 5.3.5, and Section 5.5, provide instruction for installing the CDOCS on a Sun Workstation.

5.3.1 Ensure That the Workstation Meets the Minimum Hardware and Software Requirements for a SUN Workstation

Verify the workstation meets, or exceeds, the minimum hardware and software requirements:

- Sun 4 Architecture (IPX or faster)
- 32 MB Random Access Memory (RAM)
- 10 MB free hard disk space
- Sun OS 4.1.3 or Solaris 2.3

5.3.2 Install TOPIC 4.0

TOPIC client executables may be installed on the workstation's local hard drive or on a connected file server. If TOPIC is installed on a file server, skip to Section 5.3.3. Approximately 50 Mb of local disk space is required to install TOPIC on the workstation.

To install TOPIC:

1. Create a `topic40` user account and home directory.

Example: `mkdir topic40`

2. Create a TOPIC group consisting of the users who will access TOPIC. This step is not required, however, it is strongly recommended.
3. Load the TOPIC installation media on the appropriate device. Change to the directory created in Step 1 and issue the appropriate `tar` or `ciop` command.

Example: `tar -xvf /dev/rmt/0`

4. Run the TOPIC `install` program from the top-level TOPIC directory (`topic40` directory). The `install` program asks questions about the workstation environment, and which TOPIC options to install. The `install` program initializes TOPIC, and verifies that the installation procedure has been performed properly.
5. Set the path for the `topic40` account to point to the TOPIC executables.

Note: Detailed TOPIC installation instructions ship with the TOPIC software.

5.3.3 Install the CDOCS

The CDOCS executables may be installed on the workstation's local hard drive or on a connected file server (see Figure 5-1). If the CDOCS executables reside on a file server, skip to Section 5.3.4. Approximately 2 Mb of local disk space is required to install CDOCS on the workstation. The user will be importing documents and creating query files that will be stored on the local drive. Therefore, the local CDOCS files should be installed on the local drive that has the largest available free space. After the CDOCS files are installed the workstation must be setup to run the CDOCS. Instructions for this setup are given in Section 5.4.4.

To install CDOCS on the workstation's local drive:

1. Obtain the following CDOCS files from your system administrator.

`edit_wp_file`

`vgalaxy1.vr`

`main.vr`

main

start_topic

2. Create a directory for CDOCS.

Example: **mkdir cdocs**

3. Copy the files obtained in Step 1 into the directory created in Step 2.

5.3.4 Setup of the SUN Workstation Environment

This procedure will create a specified directory structure on the workstation (/cdocs/utopic). These directories will be utilized to store the user's documents and saved TOPIC queries.

1. Create the directory /docs on the local drive.

Example: **mkdir docs**

2. Create the subdirectory /docs/utopics on the local drive.

Example: **/docs>mkdir utopics**

3. Create a new program item in the file system for CDOCS, using the following information. Drive, is the local drive used in Step 1 of this procedure.

Description = **CDOCS**

Executable File = **main**

Working Directory = <path>/**docs**

4. Create a program item in the file system for TOPIC.

5. Ensure the workstation contains path settings for:

— **set GALAXYHOME=<path>:/cdocs**

— **set HOME=<path>:\cdocs**

— **<path where TOPIC is installed>/topic40/_sso/23/bin**

— **<path where TOPIC is installed>/topic40/_sso/23/lib**

— **<path where CDOCS is installed>/<directory where CDOCS is installed>**

5. Setup the user preference file (reference Section 5.5).

5.3.5 SUN Workstation Installation Summary

The following checklist may be used as a final quick-check to verify that all of the actions necessary for installing CDOCS on a Sun workstation have been completed.

Table 5-2. SUN workstation installation summary list

Step	Page	Action
1	5-8	Verify workstation meets, or exceeds hardware/software prerequisites
2	5-8	Install TOPIC
3	5-9	Create a directory for the local CDOCS files
4	5-10	Copy the CDOCS files into the directory created in Step 3
5	5-10	Create <code><path>/docs/utopic</code> directory structure on workstation
6	5-10	Ensure workstation has a CDOCS program item (main , working directory of <code><path>/docs</code>)
7	5-10	Ensure workstation has a Topic program item (xtopic , working directory of <code>/topic40/_sso/23/bin</code>)
8	5-10	Ensure workstation's path statement includes entries for: <code>/docs</code> <code>/docs/utopic</code> path to directory where TOPIC executables are installed path to directory where CDOCS executables are installed <code>set GALAXYHOME=<path>/cdocs</code> <code>set HOME=<path>/cdocs</code>
9	5-18	Ensure the User Preference File has been setup for the following: path to WordPerfect executable path to TOPIC path to Viewer path to CDOCS Server

5.4 INSTALLATION ON AN OS/2 WORKSTATION

Sections 5.4.1 through 5.4.5, and Section 5.5, provide instructions for installing the CDOCS on an OS/2 workstation.

5.4.1 Ensure That the Workstation Meets the Minimum Hardware and Software Requirements for a Windows Workstation

Verify the workstation meets, or exceeds, the minimum hardware and software requirements:

- Intel 80486 CPU-based computer
- 16 MB Random Access Memory (RAM)
- 10 MB free hard disk space
- IBM OS/2 Version 2.1 or later
- IBM Transmission Control Protocol/Internet Protocol (TCP/IP) Version 2.0, or later, configured for Network File System (NFS).

5.4.2 Install TOPIC 4.0

TOPIC's client executables may be installed on the workstation's local hard drive or on a connected file server. If TOPIC is installed on a file server, then skip to Section 5.4.3. Approximately 50 Mb of local disk space is required to install TOPIC on the workstation.

To install TOPIC:

1. Start OS/2 and open an OS/2 window or full screen session.
2. Insert the *TOPIC* Install diskette into the diskette drive and type **a :** to make the A drive current.
3. At the prompt, type **install** and press the *Enter* key.
4. When prompted to *(I)nstall Licensed Software* or *(D)emonstration Only*, enter **I** for *Install Licensed Software*.
5. When prompted, enter the appropriate *Site Id* and *Key* values.
6. When prompted, select the drive and directory for TOPIC.
7. When prompted to choose between *Custom* or *Complete installation*, select *Custom Install*.
8. Select installation options as shown in Figure 5.2.
9. When the install program has completed, edit the `config.sys` file's path statements to add the TOPIC paths. In the following example <drive> is the drive where TOPIC was installed.

Example:

```
set path=.....<drive>:\topic400\_os2\bin;
```

```
libpath=.....<drive>:\topic400\_os2\lib;
```

10. Reboot the workstation so the changes made during the TOPIC installation take effect.
11. Open the OS/2 Templates group.
12. Drag the Program icon in the Templates group to the Desktop.
13. Enter Program Settings for the new Program icon and set the following:

Path and file name: <drive>:\topic400_os2\bin\pmtopic.exe

Parameters: - prefs topic.prf

Working directory: <drive>:\topic400\demo_31\users\pm

Program Name: TOPIC 4.0

Note: <drive> is the drive where TOPIC was installed.

TOPIC requires a preferences file to run. The CDOCS execution process will create a TOPIC preference file.

5.4.3 Install the CDOCS

The CDOCS executables may be installed on the workstation's local hard drive or they may reside on a connected file server (see Figure 5.1). The CDOCS executables require approximately 2 Mb of disk space. If the CDOCS executables reside on a file server, approximately 400 Kb of local disk space is required for installation of the CDOCS access files. The user will be importing documents and creating query files that will be stored on the local drive. Therefore, the local CDOCS files should be installed on the local drive that has the largest available free space. After the CDOCS files are installed the workstation must be setup to run the CDOCS (see Section 5.4.4).

5.4.3.1 To Install CDOCS on an OS/2 Workstation:

1. Use a text editor (such as OS/2 Edit) to open the following file:

```
<drive>:\tcpip\etc\FSTAB
```

2. If FSTAB contains a mount statement for the server drive containing the CDOCS database, then close FSTAB and go to Step 5.

3. If **FSTAB** does not contain a mount statement for the server drive containing the CDOCS database, then add the necessary mount statement. The format of the mount statement is given below:

mount <-llogonid><-ppassword><-userid><-ggroupid><drive>:<server>:<directory>

Example:

mount -lrpd -prpd/user -u2222 -g100 r: samson:/u03

4. If it is necessary to add or edit the mount statements in the **FSTAB** file, the new mount statement will not be in effect until the machine is rebooted, or the mount statements are executed through an OS/2 window.
5. Obtain the following CDOCS files from your system administrator:

topic31.pwd

main.exe

vgalaxy2.vr

rpc16.dll

pctcp.dll

startwp.exe

prt_padb_rpt.cmd

6. Create a directory for CDOCS.

Example: <drive>:\>**md CDOCS**

7. Copy **topic31.pwd** into the following TOPIC directory:

<drive>:\topic400_dos\bin\

8. Copy the remaining files into the directory created in Step 6.

5.4.3.2 To Install the CDOCS Files Required for an OS/2 Workstation to Access the CDOCS on a Connected File Server

1. Use a text editor (such as OS/2 Edit) to open the following file:

<drive>:\tcpip\etc**FSTAB**

2. If **FSTAB** contains mount statements for the server drives containing the CDOCS database and the CDOCS executables, then close **FSTAB** and go to step 5.

3. If **FSTAB** does not contain mount statements for the server drives containing the CDOCS database and the CDOCS executables, then add a mount statement for each. The format of the mount statement is given below:

```
mount <-llogonid><-ppassword><-userid><-ggroupid><drive>:<server>:<directory>
```

Example:

```
mount -lrpd -prpd/user -u2222 -g100 r: samson:/u03
```

```
mount -lrpd -prpd/user -u2222 -g100 s: mammoth:/lan/apps/os2apps
```

4. If it is necessary to add or edit the mount statements, the mounts will not be in effect until the machine is rebooted, or the mount statements are executed through an OS/2 window.
5. Obtain the following CDOCS files from your system administrator:

```
topic31.pwd
```

```
vgalaxy2.vr
```

```
prt_padb_rpt.cmd
```

6. Create a directory for CDOCS.

Example: <drive>:\>md CDOCS

7. Copy **topic31.pwd** into the following TOPIC directory:

```
<drive>:\topic400\_dos\bin\
```

8. Copy the remaining files into the directory created in Step 6.

5.4.4 Setup of OS/2 Workstation Environment

This procedure will create a specified directory structure on the workstation (<drive>:\docs\utopic). These directories will be utilized to store the user's documents, saved TOPIC queries, and reports.

1. Create the directory **\docs** on the workstation's local drive.

Example: C:\>md docs

2. Create the subdirectory **\docs\utopics** on the local drive used in Step 6 of 5.4.3.

Example: C:\docs>md utopics

3. Open the OS/2 Templates group.

4. Drag the Program icon in the Templates group to the Desktop.
5. Enter Program Settings for the Program icon, using the following information. If the CDOCS is installed on the local workstation, then <path> is the path to the directory into which the CDOCS files were copied in Section 5.4.3.1 Step 8. If the CDOCS is installed on a connected file server, then <path> is the network path to the file server directory where the CDOCS executables reside. <drive> is the directory created in Step 1.

Path and file name: <path>:\cdocs\main.exe

Working directory: <drive>:\docs

Program Name: CDOCS

6. Create a new program item for TOPIC, using the following information. If TOPIC was installed on the local workstation, then <drive> is the drive used in Section 5.2.2 Step 3. If TOPIC is installed on a file server then <drive> is the network drive where the TOPIC executables are located.

Path and file name: <drive>:\topic400_os2\bin\pmtopic.exe

Parameters: - prefs topic.prf

Working directory: <drive>:\topic400\demo_31\users\pm

Program Name: TOPIC

7. Ensure the workstation config.sys file contains path settings for:

```
—      set GALAXYHOME =<drive>:\cdocs
—      set HOME=<drive>:\cdocs
—      <drive where TOPIC is installed>:\topic400\_dos\bin
—      <drive where TOPIC is installed>:\topic400\_dos\lib
—      <drive where CDOCS is installed>:\<directory where CDOCS is
      installed>
```

If alterations are made to the config.sys file, reboot the machine before continuing.

8. Setup the user preference file (reference Section 5.5).

5.4.5 OS/2 Workstation Installation Summary

The following list may be used as a final quick check to verify that all of the actions necessary for installing CDOCS on a OS/2 workstation have been completed.

Table 5-3. OS/2 workstation installation summary list

Step	Page	Action
1	5-11	Verify workstation meets, or exceeds hardware/software prerequisites
2	5-12	Install TOPIC
3	5-13	Ensure FSTAB contains mount statement for necessary server drives
4	5-14	Create a directory for the local CDOCS files
5a	5-14	If installing CDOCS on the local workstation, copy the following CDOCS files into the directory created in Step 4 <code>topic31.pwd</code> <code>main.exe</code>
5a	5-14	<code>vgalaxy2.vr</code> <code>rpc16.dll</code> <code>pctcp.dll</code> <code>startwp.exe</code> <code>prt_padb_rpt.cmd</code>
5b	5-15	If the CDOCS is installed on a file server, copy the following CDOCS files into the directory created in Step 4 <code>topic31.pwd</code> <code>vgalaxy2.vr</code>
		<code>prt_padb_rpt.cmd</code>
6	5-15	Copy <code>topic31.pwd</code> into <code><drive>:\topic400_os2\bin</code>
7	5-15	Create <code><drive>:\docs\utopic</code> directory structure on workstation
8	5-15	Ensure workstation has a CDOCS program item (<code>main.exe</code> , working directory of <code><drive>:\cdocs</code>)
9	5-16	Ensure workstation has a Topic program item (<code>pmtopic.exe</code> , working directory of <code><drive>:\topic400_os2\bin</code>)

Table 5-3. OS/2 Workstation installation summary list (cont'd)

Step	Page	Action
10	5-16	Ensure workstation's <code>config.sys</code> path statements includes entries for: <code><drive>:\cdocs</code> <code><drive>:\docs</code>
		<code><drive>:\cdocs\utopic</code> <code><drive>:\topic400_os2\bin</code> <code><drive>:\topic400_os2\lib</code> <code><drive>:\<path></code> to directory where CDOCS executables are installed <code>set GALAXYHOME=<drive>:\cdocs</code>
		<code>set HOME=<drive>:\cdocs</code>
11	5-18	Ensure the User Preference File has been setup for the following: path to WordPerfect executable path to TOPIC path to Viewer path to CDOCS Server

5.5 SELECTING USER PREFERENCES

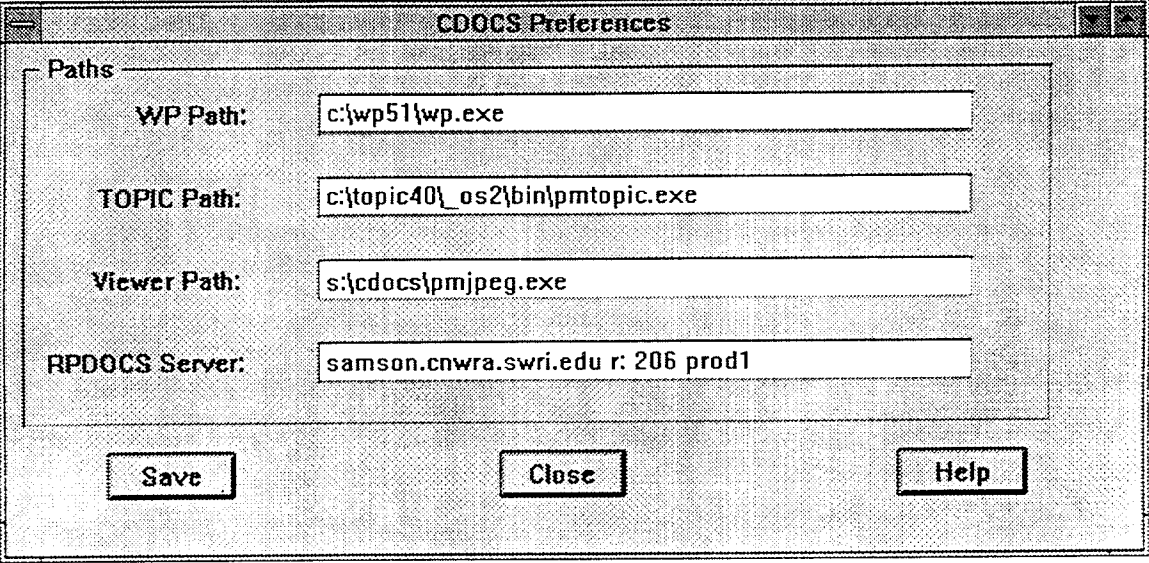
The preferences file must be set to reflect the workstation operating environment (network drives and paths, WordPerfect path, etc). The following procedure is used at installation time to initially set the workstation's environment. The same procedure may be used by the CDOCS custodian/administrator to update the preference file, at a later date, to reflect changes to the workstation's environment. A CDOCS custodian/administrator level logon and password are required to set preferences.

Setup the user preference file.

1. Open program settings for the CDOCS icon.
2. In settings window enter the following Optional Parameter: `- i`
3. Close the CDOCS program settings window.
4. Double click on the CDOCS desktop icon with the right mouse button.
5. The system displays the *CDOCS Preferences* screen (Figure 5-3).

6. Enter the full path and file name of the WordPerfect program in the *WordPerfect Path* box.
Example: `c:\wp51\wp.exe`
7. Enter the full path and file name of the TOPIC program field in the *TOPIC Path* box.
Example: `c:\topic315_os2\bin\pmtopic.exe`
8. Enter the full path and file name of the Viewer program in the *Viewer Path* box.
Example: `s:\cdocs\pmjpeg.exe`
9. Enter the FTP path to the server and drive where the CDOCS database resides in the *RPDOCS Server* box.
Example: `samson.cnwra.swri.edu r: 206 prod1`
10. Review the selected information and make corrections if necessary. When all of the information is correct, select the *Save* push-button at the bottom of the *CDOCS Preferences* screen to save the new preference information.
11. To exit from the *CDOCS Preferences* screen, select the *Save* push-button at the bottom of the screen. If the screen is exited (*Close* button) before selecting the *Save* push-button, any information that was entered on the screen is discarded, and the previous preference information is retained.
12. Open the CDOCS program settings window and remove the Optional Parameter entered in step 2.
13. Close the CDOCS program settings window.

For help in using the *CDOCS Preferences* screen, select the *Help* push-button at the bottom of the screen to display help information.



The image shows a Windows-style dialog box titled "CDOCS Preferences". It has a "Paths" section with four labeled text input fields. The "WP Path" field contains "c:\wp51\wp.exe". The "TOPIC Path" field contains "c:\topic40_os2\bin\pmtopic.exe". The "Viewer Path" field contains "s:\cdocs\pmjpeg.exe". The "RPDOCS Server" field contains "samson.cnwra.swri.edu r: 206 prod1". At the bottom of the dialog are three buttons: "Save", "Close", and "Help".

CDOCS Preferences	
Paths	
WP Path:	c:\wp51\wp.exe
TOPIC Path:	c:\topic40_os2\bin\pmtopic.exe
Viewer Path:	s:\cdocs\pmjpeg.exe
RPDOCS Server:	samson.cnwra.swri.edu r: 206 prod1
Save Close Help	

Figure 5-3. CDOCS Preferences screen

5.6 INSTALLATION OF CDOCS SERVER

The server portion of the CDOCS system is distributed on four 8-mm tapes for the Solaris 2.3 operating system. It is assumed that dependent software packages such as ORACLE and TOPIC have been properly installed and tested according to the package's native documentation. The following steps describe the server installation using the four 8-mm distribution tapes.

1. Untar the first tape, labeled "CDOCS System Source Code" into the preferred directory used for software configuration management (e.g., `cd install_dir`; `tar -xvf /dev/rmt/0`). Because the precompiled system makes no direct use of the source code, the location is not critical to execution.
2. Untar the second tape, labeled "CDOCS System Runtime Executables" into the preferred directory. All server processes, as well as client processes execute from these directories. Therefore, all CDOCS processes need access to this subdirectory structure. This tape provides not only the server processes and scripts (such as the batch and server processes) but also the client processes for all of the supported platforms.
3. The owner of all objects in the TOPIC database is topic40 and belongs to the topic group (userid 7770, groupid 201). Ensure that this account exists before manipulating any files from Step 4.
4. Untar the third tape, labeled "CDOCS System Database" into the destination directory. That directory should be at the root level of a 2-GB partition dedicated to the database only. The name of the destination directory (eg prod1 or NMSS) must be identical to the name of the database specified in the system .ini files that are used to startup the server process.
5. Untar the fourth tape, labeled "CDOCS ORACLE Database Dump" into a preferred location, such as /tmp. These files are used to create the database object for the CDOCS database owner.
6. The owner of all database objects is rpdocs (p/w ichiban, see Chapter 6). Use the appropriate ORACLE utilities to import the objects owned by rpdocs into the ORACLE database using the files from Step 5.
7. Edit the server .ini files, located in the subdirectory:

preferred_directory/runtime_executables/samson_solaris_installs/server

underneath the executables installation directory, to include appropriate values for database name, synchronization, etc. as directed in the comments within that .ini file. Remember, the NRC should operate the 207 process in order to properly synchronize with the CNWRA 206 process.

8. Start the server process as documented in Chapter 2.

9. The batch operation will read the server .ini files for its proper startup values. A cron job will need to be added to run batch every night after local backup operations are complete. The batch command is documented in Chapter 2.

For further discussion of startup parameters for server and batch processes, please see Chapters 2, 7, and 8, and specifically the read.me files in Chapter 4 that document those processes.

6 STRUCTURE OF DATABASES

The CDOCS application uses two commercial products (ORACLE and TOPIC) each of which have their own databases and database management systems. In addition, CDOCS has its own database/directory structure for storing various files particularly those associated with the report writer.

6.1 CDOCS ORACLE DATABASE

This chapter describes the CDOCS database which is called RPDOCS and is located at the CNWRA on the server Samson, (SUN SPARC).

6.1.1 Databases and the Database Administrator

The ORACLE database for CDOCS has a single DBA, rpdocs, who owns all CDOCS objects (or tables) under RPDOCS (formerly called TDOCS and RPC databases).

6.1.2 Instances

The prod1 (the production database for use with the 206 server) and test 1 (the test database for use with the 205 server) are the two ORACLE instances which have been set up at the CNWRA. Their configuration fields are: \$ORACLE_HOME/dbs/initprod1.ora which are linked to:

/u01/home/dba/oracle/admin/test1/pfile/config.ora
and /u01/home/dba/oracle/admin/prod1/pfile/config.ora

6.1.3 Tablespaces

The test1 users have a default tablespace called APPS and a temporary tablespace called TEMP. All RPDOCS objects were explicitly created in the APPS tablespace. The prod1 instance is the same. Non-RPDOCS objects, if any, are explicitly created in the USERS tablespace in order to separate data.

6.1.4 Database Objects

The database objects are logically grouped into shared TDOCS objects, RPDOCS objects, and RPD objects. These objects are prefixed to identify the logical group, with the exception of the PEOPLE object which has more general application to other applications under ORACLE.

6.2 TDOCS Objects

TDOCS objects are TDOCS, TDOCS_AUTHORS, TDOCS_ADDRESSEES, TDOCS_ASSIGNED_CODES, TDOCS_ASSIGNED_NUMBERS, TDOCS_DOCUMENT_SETS, TDOCS_DOCUMENT_STATUS, TDOCS_REPORTS, TDOCS_HYDRO, and TDOCS_NIST. The two objects TDOCS_NUREG and TDOCS_REP are not longer used. These two objects were used for demonstration purposes in the past in the former TDOCS system.

Table 6-1. TDOCS

TDOCS (holds header information for each document in the former TDOCS partitions)			
Column Name	Data Type	Constraints	Explanation
DOCUMENT_SET	VARCHAR2(20)	NOT NULL	allowable partitions of former TDOCS portion: NTD, BNTD, TDI, QA, CSP, CTD, REP, REG, TP, FCSP, NUREG, NIST, HYDRO
DOCUMENT_ID	NUMBER(8)	NOT NULL	identifies document in set, document bases file name
DOCUMENT_STATUS	VARCHAR2(1)	NOT NULL	document status: I= input, S=submit, L=loaded, U=update, D=delete, X=deleted
SOURCE_TYPE	NUMBER	NOT NULL	identifies types of associated documents and/or images
SUBMISSION_DATE	DATE	NOT NULL	date submitted/updated/deleted
CUSTODIAN	VARCHAR2(50)	NOT NULL	identifies who submitted/updated/deleted
DOCUMENT_NUMBER	VARCHAR2(13)	NOT NULL, UNIQUE	external unique identifier, system supplied
TITLE	VARCHAR2(500)	NOT NULL	document title
PUBLICATION	VARCHAR2(500)		publication information
DOCUMENT_DATE	DATE	NOT NULL	publication date
DURATION	VARCHAR2(20)		retention period, etc.
NOTE	VARCHAR2(500)		miscellaneous
SHARABLE	VARCHAR2(1)		determines if document is to be shared through synchronized servers 1=share document, 0=not share document
INDEXES: PRIMARY KEY (DOCUMENT_SET, DOCUMENT_ID), TDOCS_TDOCS_STAT_INDEX (SUBMISSION_DATE, DOCUMENT_NUMBER)			

Table 6-2. TDOCS_AUTHORS

TDOCS_AUTHORS (holds detailed document author information from header)			
Column Name	Data Type	Constraints	Explanation
DOCUMENT_SET	VARCHAR2(20)	NOT NULL	document set (See Table-TDOCS)
DOCUMENT_ID	NUMBER(8)	NOT NULL	document id (See Table -TDOCS)
AUTHOR	VARCHAR2(50)	NOT NULL	document author
SORT_ID	NUMBER(2)	NOT NULL	sorts primary key records
INDEXES: PRIMARY KEY TDOCS_AUTHORS_INDEX (DOCUMENT_SET, DOCUMENT_ID) FOREIGN KEY (DOCUMENT_SET, DOCUMENT_ID) REFERENCES TDOCS (DOCUMENT_SET, DOCUMENT_ID)			

Table 6-3. TDOCS_ADDRESSEES

TDOCS_ADDRESSEES (holds detailed addressees information from header)			
Column Name	Data Type	Constraints	Explanation
DOCUMENT_SET	VARCHAR2(20)	NOT NULL	document set (See Table-TDOCS)
DOCUMENT_ID	NUMBER(8)	NOT NULL	document id (See Table-TDOCS)
ADDRESSEE	VARCHAR2(50)	NOT NULL	correspondence addressee
SORT_ID	NUMBER(2)	NOT NULL	sorts primary key records
INDEXES: PRIMARY KEY TDOCS_ADDRESSEES_INDEX (DOCUMENT_SET, DOCUMENT_ID) FOREIGN KEY (DOCUMENT_SET, DOCUMENT_ID) REFERENCES TDOCS (DOCUMENT_SET, DOCUMENT_ID)			

Table 6-4. TDOCS_DOCUMENT_STATUS

TDOCS_DOCUMENT_STATUS (holds status for each document)			
Column Name	Data Type	Constraints	Explanation
DOCUMENT_STATUS	VARCHAR2(1)	PRIMARY KEY	I=input, S-submit, L-loaded, U-update, D-delete, X-deleted
SORT_ID	NUMBER(1)	NOT NULL	not used currently in this table
DESCRIPTION	VARCHAR2(50)	NOT NULL	describes status
INDEXES: PRIMARY KEY (DOCUMENT_STATUS)			

Table 6-5. TDOCS_ASSIGNED_CODES

TDOCS_ASSIGNED_CODES (holds detailed NRC/CNWRA subject code information from header)			
Column Name	Data Type	Constraints	Explanation
DOCUMENT_SET	VARCHAR2(20)	NOT NULL	document set (See Table-TDOCS)
DOCUMENT_ID	NUMBER(8)	NOT NULL	document id (See Table-TDOCS)
ASSIGNED_CODE	VARCHAR2(30)	NOT NULL	code assigned to document
SORT_ID	NUMBER(2)	NOT NULL	sorts primary key records
INDEXES: PRIMARY KEY TDOCS_ASSIGNED_CODES_INDEX (DOCUMENT_SET, DOCUMENT_ID) FOREIGN KEY (DOCUMENT_SET, DOCUMENT_ID) REFERENCES TDOCS (DOCUMENT_SET, DOCUMENT_ID)			

Table 6-6. TDOCS_ASSIGNED_NUMBERS

TDOCS_ASSIGNED_NUMBERS (holds document number from header)			
Column Name	Data Type	Constraints	Explanation
DOCUMENT_SET	VARCHAR2(20)	NOT NULL	document set (See Table-TDOCS)
DOCUMENT_ID	NUMBER(8)	NOT NULL	document id (See Table-TDOCS)
ASSIGNED_NUMBER	VARCHAR2(100)	NOT NULL	number assigned to document
SORT_ID	NUMBER(2)	NOT NULL	sorts primary key records
INDEXES: PRIMARY KEY TDOCS_ASSIGNED_NUMBERS_INDEX (DOCUMENT_SET, DOCUMENT_ID) FOREIGN KEY (DOCUMENT_SET, DOCUMENT_ID) REFERENCES TDOCS (DOCUMENT_SET, DOCUMENT_ID)			

Table 6-7. TDOCS_REPORTS

TDOCS_REPORTS (holds content information for system reports, e.g., NIST output report)			
Column Name	Data Type	Constraints	Explanation
REPORT	VARCHAR2(255)	PRIMARY KEY	report id
TITLE	VARCHAR2(255)	NOT NULL	report title
HELP	VARCHAR2(255)	NOT NULL	report help
INDEXES: PRIMARY KEY (REPORT)			

Table 6-8. TDOCS_DOCUMENT_SETS)

TDOCS_DOCUMENT_SETS (holds document partition information)			
Column Name	Data Type	Constraints	Explanation
DOCUMENT_SET	VARCHAR2(20)	PRIMARY KEY	NTD, BNTD, TDI, QA, CSP, CTD, REP, REG, TP, FCSP, NUREG, NIST, HYDRO
SORT_ID	NUMBER(3)	NOT NULL	sorts document set display
TITLE	VARCHAR2(255)	NOT NULL	full document set name
SPECIFICATION	VARCHAR2(12)	NOT NULL	specifies mandatory versus optional fields for document set
LOADABLE	VARCHAR2(1)	NOT NULL	determines if set is batch loadable
TABLE_NAME	VARCHAR2(30)	NOT NULL	table set data is stored in
LOCATION	VARCHAR2(6)		number assigned to document location of data (LOCAL versus REMOTE server)
SHARABLE	NUMBER2(1)		determines if set is sharable across sites
DOCUMENT_SET_ID	VARCHAR2(1)		
INDEXES: PRIMARY KEY (DOCUMENT_SET)			

Table 6-9. TDOCS_CIRCULATION

TDOCS_CIRCULATION [holds information on who has the hard copy (checked out) of a document from a library facility]			
Column Name	Data Type	Constraints	Explanation
DOCUMENT_NUMBER	VARCHAR2(255)	NOT NULL	Identifies document checked out (See Table-TDOCS)
CHECKOUT_NAME	VARCHAR2(100)	NOT NULL	person document is checked out to
CHECKOUT_DATE	DATE	NOT NULL	date document was checked out
SORT_ID	VARCHAR2(255)	NOT NULL	sorts on document number
INDEXES: PRIMARY KEY TDOCS_CIRCULATION_DOC_INDEX (DOCUMENT_NUMBER), TDOCS_CIRCULATION_NAME_INDEX (CHECKOUT_NAME)			

Table 6-10. TDOCS_HYDRO

TDOCS_HYDRO [holds content of the NRC Hydrologic database (Bill Ford)— this partition cannot be updated by the custodian]			
Column Name	Data Type	Constraints	Explanation
DOCUMENT_ID	NUMBER(8)	PRIMARY KEY	unique id
DOCUMENT_SET	VARCHAR2(20)	NOT NULL	HYDRO
DOCUMENT_STATUS	VARCHAR2(1)	NOT NULL	document status (See Table-TDOCS_DOCUMENT_STATUS)
CUSTODIAN	VARCHAR2(50)	NOT NULL	person who submitted
SUBMISSION_DATE	DATE	NOT NULL	submission date
DOCUMENT_NUMBER	VARCHAR2(13)	NOT NULL	external identifier
AUTHOR	VARCHAR2(255)		document author(s)
TITLE	VARCHAR2(255)	NOT NULL	document title
DOCUMENT_DATE	DATE	NOT NULL	publication date
ORG_NUMBER	VARCHAR2(255)		organization number
INDEXES: PRIMARY KEY (DOCUMENT_ID), TDOCS_HYDRO_INDEX (DOCUMENT_NUMBER), TDOCS_HYDROSUB_INDEX (SUBMISSION_DATE)			

Table 6-11. TDOCS_NIST_ABSTRACTS

TDOCS_NIST_ABSTRACTS (holds the NRC NIST database information)			
Column Name	Data Type	Constraints	Explanation
DOCUMENT_ID	NUMBER(8)	NOT NULL	document id (See Table-TDOCS_NIST)
ABSTRACT	LONG	NOT NULL	listing of abstract
INDEXES: PRIMARY KEY (DOCUMENT_ID)			

Table 6-12. TDOCS_NIST

TDOCS_NIST [holds the content of the NRC NIST database (Chuck Interrante) —this partition cannot be updated by the custodian]			
Column Name	Data Type	Constraints	Explanation
DOCUMENT_ID	NUMBER(8)	PRIMARY KEY	unique id
DOCUMENT_SET	VARCHAR2(20)	NOT NULL	NIST
DOCUMENT_STATUS	VARCHAR2(1)	NOT NULL	document status
CUSTODIAN	VARCHAR2(50)	NOT NULL	person who submitted
SUBMISSION_DATE	DATE	NOT NULL	date submitted
DOCUMENT_NUMBER	VARCHAR2(6)	NOT NULL	external id
AUTHOR	VARCHAR2(255)		document author(s)
TITLE	VARCHAR2(255)	NOT NULL	document title
SOURCE	VARCHAR2(255)		document source
DOCUMENT_DATE	DATE	NOT NULL	publication date
CHAPTER	VARCHAR2(25)		chapter in collection
VOLUME	VARCHAR2(15)		volume in collection
PAGE_REFERENCE	VARCHAR2(15)		page reference
AVAILABILITY	VARCHAR2(150)		where to get document
NIST_NUMBER	VARCHAR2(50)		NIST number
CONTRACTOR	VARCHAR2(560)		contractor related to document
CONTRACT_NUMBER	VARCHAR(75)		contract number related to document
SPONSOR	VARCHAR2(175)		sponsor related to document
PUBLISHER	VARCHAR2(150)		document publisher
FILE_LOCATION	VARCHAR2(150)		location of document file
ISSUE	VARCHAR2(10)		issue in collection
EDITORS	VARCHAR2(100)		document editors
NOTES	VARCHAR2(560)		notes on document
REPORT_NUMBER	VARCHAR2(50)		document report number
RELATED_CITATIONS	VARCHAR2(150)		related citations
ABSTRACT	VARCHAR2(1)		is there an abstract?
KEYWORDS	VARCHAR2(1)		are there keywords?
REVIEW	VARCHAR2(1)		is there a review?
REVIEW_TYPE	VARCHAR2(150)		review type (if any)

Table 6-12. TDOCS_NIST (cont'd)

TDOCS_NIST [holds the content of the NRC NIST database (Chuck Interrante) —this partition cannot be updated by the custodian]			
FLAG	VARCHAR2(50)		flag text
INDEXES: TDOCS_NIST_INDEX (DOCUMENT_NUMBER), TDOCS_NISTSUB_INDEX (SUBMISSION_DATE)			

Table 6-13. TDOCS_NIST_REVIEWS

TDOCS_NIST_REVIEWS (holds the NRC NIST database information)			
Column Name	Data Type	Constraints	Explanation
DOCUMENT_ID	NUMBER(8)		document id (See Table-TDOCS_NIST)
REVIEW	LONG	NOT NULL	listing of reviews
INDEXES: PRIMARY KEY (DOCUMENT_ID)			

Table 6-14. TDOCS_NIST_KEYWORDS

TDOCS_NIST_KEYWORDS (holds the NRC NIST database information)			
Column Name	Data Type	Constraints	Explanation
DOCUMENT_ID	NUMBER(8)	NOT NULL	document id (See Table-TDOCS_NIST)
KEYWORDS	LONG	NOT NULL	list of keywords
INDEXES: PRIMARY KEY (DOCUMENT_ID)			

6.3 RPDOCS Objects

Shared RPDOCS objects are PEOPLE, RPDOCS_USERS, RPDOCS_NAMES, RPDOCS_PRIVILEGES, RPDOCS_HELP_INDEX, and RPDOCS_HELP. RPDOCS_USERS and RPDOCS_NAMES are used for logging into CDOCS including custodians, etc. PEOPLE is used by CDOCS to track document check out and check in; it can and is also used by other ORACLE applications such as the Scheduler Program at CNWRA.

Table 6-15. PEOPLE

PEOPLE (used to store full names of users)			
Column Name	Data type	Constraints	Explanation
NAME	VARCHAR2(50)	NOT NULL	system user proper names
INDEXES: PRIMARY KEY (NAME)			

Table 6-16. RPDOCS_USERS

RPDOCS_USERS (used to store user names, passwords, and privileges)—was table USER_ID_PRIVILEGE (under the old TDOCS and RPD systems)			
Column Name	Data Type	Constraints	Explanation
USER_ID	VARCHAR2(20)	NOT NULL	abbreviated user names, typically first initial and first six letters of surname
NAME_ID	NUMBER(4)	NOT NULL	unique sequential number for each name
PRIVILEGE	NUMBER(2)	NOT NULL	NRC user=0, NRC custodian=1, CNWRA user=2, CNWRA custodian=3, DBA=60
INDEXES: PRIMARY KEY (USER_ID)			

Table 6-17. RPDOCS_NAMES

RPDOCS_NAMES (used to uniquely and numerically refer to each user)			
Column Name	Data Type	Constraints	Explanation
NAME_ID	NUMBER(4)	NOT NULL	(See Table-RPDOCS_USERS)
NAME	VARCHAR2(50)	NOT NULL	(See Table-PEOPLE)
INDEXES: PRIMARY KEY (NAME_ID)			

Table 6-18. RPDOCS_HELP

RPDOCS_HELP (used to store text for CDOCS HELP system)			
Column Name	Data Type	Constraints	Explanation
ID	NUMBER	NOT NULL	unique number of HELP statement
HEADER	VARCHAR2(2000)		HELP dialog box title/header
TEXT	LONG		text of HELP statement
INDEXES: PRIMARY KEY (ID, HEADER)			

Table 6-19. RPDOCS_HELP_INDEX

RPDOCS_HELP_INDEX (used to store appropriate access and button labels for HELP system)			
Column Name	Data Type	Constraints	Explanation
PRIVILEGE	NUMBER(3)	NOT NULL	NRC user=0, NRC custodian=1, CNWRA user=2, CNWRA custodian=3, DBA=60
BUTTON	VARCHAR2(64)	NOT NULL	Button labels for HELP screens
SUBKEY	VARCHAR2(64)	NOT NULL	all contain "NONE", i.e. currently not used
SORT-ID	NUMBER		all contain "0", i.e. currently not used
ID	NUMBER		unique index number for a complete HELP screen (button label, header, text, and appropriate privilege for display)
INDEXES: PRIMARY KEY (PRIVILEGE, ID)			

Table 6-20. RPDOCS_PRIVILEGES

RDOCS_PRIVILEGES (used to describe privilege classes)			
Column Name	Data Type	Constraints	Explanation
PRIVILEGE	NUMBER(3)	NOT NULL	(See Table-RPDOCS_HELP_INDEX)
DESCRIPTION	VARCHAR2(256)		text description
INDEXES: PRIMARY KEY (PRIVILEGE)			

6.4 RPD OBJECTS

RPD objects include RPD, RPD_DOCUMENT_SETS, RPT_PIECES, AND RPD_REPORTS.

Table 6-21. RPD_DOCUMENT_SETS

RPD_DOCUMENT_SETS—was VALID_DOCUMENT_TYPE (under old RPD system)			
Column Name	Data Type	Constraints	Explanation
DOCUMENT_SET	VARCHAR2(8)	NOT NULL	allowable partitions of former RPD portion: CDS, CDM, RPS, OITS - old name was DOCUMENT_TYPE;
PARTITION	VARCHAR2(8)		p_cds, p_cdm, p_rps, p_oits
INDEXES: PRIMARY KEY (DOCUMENT_SET)			

Table 6-22. RPD_REPORTS

RPD_REPORTS (used to store content information for the Report Writer to generate formatted reports, e.g., LARP)—was REPORT + QUERY (under old RPD system)			
Column Name	Data Type	Constraints	Explanation
REPORT	VARCHAR2(20)		report identifier, DTD & macro base file names
TYPE	VARCHAR2(10)	NOT NULL, (ALL, INDIVIDUAL)	complete versus subset report; old name was QUERY SUBPART
TITLE	VARCHAR2(255)	NOT NULL	report title display
HELP	VARCHAR2(255)		report help display
SORT_ID	NUMBER(6)		sorts report list display
PRIVILEGE	VARCHAR2(10)	NOT NULL,	determines who can see which report; old name was REPORT TYPE
BATCH	VARCHAR2(1)	NOT NULL,	determines if complete report is generated during batch; Y=yes, N=no
SGML_TAG	VARCHAR2(255)	NOT NULL	DTD SGML tag where query is “hung;” old name was TAG
QUERY	LONG	NOT NULL	query to retrieve pieces for report
INDEXES: PRIMARY KEY (REPORT, TYPE), RPD_REPORTS_RPTTAG_INDEX (REPORT, SGML_TAG)			

Table 6-23. RPD_PIECES

RPD_PIECES—was PIECE (under old RPD system)			
Column Name	Data Type	Constraints	Explanation
DOCUMENT_SET	VARCHAR2(8)	REFERENCES RPD_DOCUMENT_ SETS (DOCUMENT_SET)	(See Table-RPD_DOCUMENT_SETS) was formerly DOCTYPE
DOCUMENT_NUMBER	VARCHAR2(20)	NOT NULL	review plan number or OITS ID; was formerly DOCNUM
SORT_ID	NUMBER	NOT NULL	sorted numeric review plan number or OTIS ID
SEQ_NO	NUMBER	NOT NULL	orders review plan pieces
SGML_TAG	VARCHAR2(255)	NOT NULL	SGML tag; was formerly TAG
PIECE	LONG		SGML piece
SEARCH	VARCHAR2(2000)		ASCII piece
INDEXES: PRIMARY KEY (DOCUMENT_SET, DOCUMENT_NUMBER, SEQ_NO), RPD_PIECES_DOCNUM_INDEX (DOCUMENT_NUMBER), RPD_PIECES_SEQNO_INDEX (SEQ_NO), RPD_PIECES_SORT_INDEX (SORT_ID), RPD_PIECES_TAG_INDEX (SGML_TAG)			

Table 6-24. RPD

RPD (holds header and system information for former RPD partitions)			
Column Name	Data Type	Constraints	Explanation
DOCUMENT_SET	VARCHAR2(8)	NOT NULL	(See Table-RPD_DOCUMENT_SETS)
DOCUMENT_ID	NUMBER(8)	NOT NULL	unique system number established at "define"
INSTANCE_ID	NUMBER(3)	NOT NULL	number of times document has been updated (i.e., copies are archived)
SORT_ID	NUMBER	NOT NULL	system sort number
PARTITION	VARCHAR2(8)	NOT NULL	(See Table-RPD_DOCUMENT_SETS)
FILE_NAME	VARCHAR2(12)	NOT NULL	unix system file name made up of prefix=DOCUMENT_ID and suffix=INSTANCE_ID
DOCUMENT_NUMBER	VARCHAR2(20)	NOT NULL	review plan/OITD ID designation
TITLE	VARCHAR2(255)	NOT NULL	document title
DOCUMENT_STATUS	VARCHAR2(12)	NOT NULL	vacant=defined but not yet loaded, active=currently loaded archive=copy which was previously loaded
MAJOR_VERSION	NUMBER(3)	NOT NULL	prefix starting at "0", for tracking document revisions
MAJOR_VERSION_NAME_ID	NUMBER(4)		user name of custodian who entered document
MAJOR_VERSION_DATE	DATE		date of document entry
MINOR_VERSION	NUMBER(3)	NOT NULL	suffix starting at "0", for tracking minor document revisions
MINOR_VERSION_NAME_ID	NUMBER(4)		user name of custodian who entered document
MINOR_VERSION_DATE	DATE		date of document entry
CHECKOUT_NAME_ID	NUMBER(4)		currently not used
CHECKOUT_DATE	DATE		currently not used
INDEXES: PRIMARY KEY (DOCUMENT_SET, DOCUMENT_ID, INSTANCE_ID)			

6.5 CONSOLIDATED DOCUMENTS SYSTEM TOPIC DATABASE

To implement CDOCS using the client/server architecture the relational, full-text, and image data repositories must reside on a single central server. The decision to implement the CDOCS using this client/server architecture with the server facilities resident on a UNIX-based platform implied that the server portion of the relational database and full-text search and retrieval facilities must be able to operate in a UNIX environment.

The server database is the primary repository for document headers, text, images, and indexes. This database resides on a server computer that provides sufficient capacity to support the database storage functions and the projected number of users. Information in the CDOCS is stored in several repositories to support different system requirements:

- A relational database management system for control information
- A file system for text, work processing, and image files
- A full-text index to support search and retrieval capabilities

Although multiple software tools are used to implement this module, it is accessed and maintained as a single database. Headers are stored in the RDBMS and indexed in the full-text search and retrieval system. Document text and images are physically stored in the file system, and the textual information is indexed in the full-text repository.

In addition to the RDBMS and the full-text repositories, *archive* text and image repositories reside in the file system of the central server platform. The UNIX file system provides for password-protected access as well as group and user level read/write privileges on directories and files. Access is controlled according to privileges assigned to individual users or groups of users. Password-protected access and read/write privileges are used to protect documents, headers, and their indexes from unauthorized modification or deletion. The UNIX system also provides many of the utilities necessary for system administration and maintenance.

The primary function of the file system is the storage of documents. Both the RDBMS and full-text search and retrieval facilities make extensive use of the UNIX file system for storage. Text and images are stored in partitions that correspond to sets of documents. These partitions are formed as directory structures that parallel the partition structure used to store indexes in the full-text search and retrieval facility. Examples of document sets include NRC technical document (NTD) at the DWM and full text correspondence (FCSP) at the CNWRA.

6.5.1 Directory Structures in the UNIX File System

Document storage in the CDOCS repository requires support for text, images full-text display, and hyperlinks. Accordingly, parallel directory structures are provided in the UNIX file system, as illustrated in Figure 6-1.

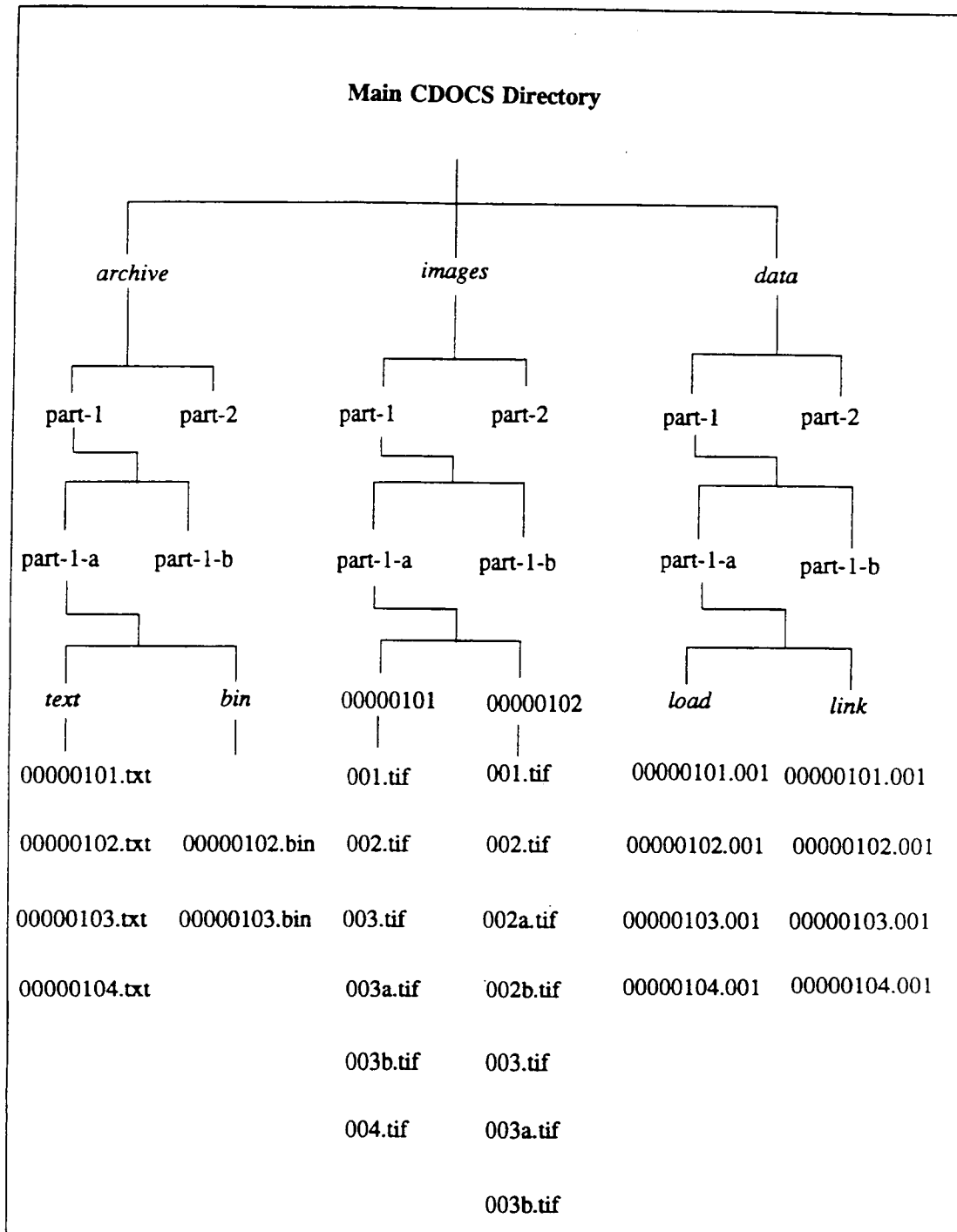


Figure 6-1. Directory structures in the UNIX file system

- *archive* directories—contain the text of the document as originally submitted to the system
- *image* directories—contain full-page images as well as any images of figures, equations, etc.
- *data* directories—contain copies of the document text formatted to support full-text display and hyperlink functionality

Under each of the primary directories (i.e. *archive*, *images*, and *data*), there are multiple partitions (directories) corresponding to the document sets within CDOCS. Under each document set partition, there are multiple subpartitions (subdirectories). The purpose of these subpartitions is to accommodate a limitation that permits a maximum of 1,024 file name arguments to be passed to a command. Therefore, the first 1,024 documents added to a document set are stored under the first subpartition; the next 1,024 documents are stored under the second subpartition, etc. The structures of the *archive*, *images*, and *data* directories are parallel down to the subpartition level. Below this level, the structures diverge to accommodate the requirements for each type of information.

6.5.1.1 Archive Subdirectories

The partitions and subpartitions under the *archive* directory contain the original text. The text may have been entered in ASCII or in a word processing format such as *WordPerfect*. Therefore, two subdirectories are created under each *archive* subpartition to accommodate ASCII and word processing text:

- *txt*—files stored in ASCII format
- *bin*—word processing files stored in word processing (binary) format

The Document ID, a unique internally generated number associated with each document, is used as the file name of the ASCII and word processing files under the *archive* directory. An appropriate file type is used as the file extension to indicate the format of the text. ASCII files always have a file name extension of *txt*. Word processing files always have a file extension of *bin*.

An ASCII copy of each submitted document is required to support loading and indexing of the document for full-text search and retrieval. When a document is submitted in word processing format, it must be converted to ASCII for this purpose, but the word processing file is also stored. The ASCII copy of the text is stored in the *txt* directory, and the corresponding word processing file, if any, is stored in the *bin* directory.

6.5.1.2 Images Subdirectories

The partitions and subpartitions under the *images* directory may contain the full-page images and/or images of figures, equations, etc. An *images* directory will exist for a document if and only if images are submitted with that document. Thus, a document submitted with images will have the submitted image files stored in a separate subdirectory that is named with the unique internally generated Document ID for that document. The submitted image files may represent full-page images and/or individual figures. The *images* subdirectory for a scanned document will normally contain an image file for each page. These full-page images files use their respective page numbers as file names (e.g., *001*, *002*,

003, etc.). Images files for figures, equations, etc., are named with a composite of the page number and an alphabetic suffix indicating the image sequence within the page (e.g., 001a, 001b, 002a, 003a, etc.). The file name extensions for image files indicate the format of the file (e.g, gif, tif, pcx, etc.).

6.5.1.3 Data Subdirectories

The partitions and subpartitions under the *data* directory contain processed copies of the document text that are used to support full-text display and hyperlinks. Under each *data* subpartition, there is a *load* subdirectory and a *link* subdirectory. Each document has a copy of the text in both the *load* and *link* subdirectories, and these files have identical file names that consist of the system-generated document number and file name extension of 001. The file extension permits future implementation of version control, if and when it is required.

The *load* subdirectory contains a single file for each document. The document text is prefixed with a formatted ASCII copy of the document header and includes the document text and embedded hyperlink labels. The files in the *load* subdirectory are used to support viewing of the document.

The *link* subdirectory contains a single file for each document. The text for each document is prefixed with a formatted ASCII copy of the document header and includes the document text and embedded hyperlink labels. The files in the *link* subdirectory are input to the full-text indexing process.

6.5.2 Directory Structures in the TOPIC Specific File System

The TOPIC file system is comprised of many directories which are described below. Data, database components, and user directories can be located anywhere. Specific directory names are not required, but it is highly recommended to use the basic structure described.

styles directories - contain style files that describe each document partition

parts directory - contains partitions that comprise the TOPIC database

data directories - contain system topics

partind directory - contains the topic database and word list

users directory - contains all user accounts

sources directory - contains user defined topic queries

tools directory - contains user defined notes, scripts and special programs

Styles Subdirectories

The subdirectories under the styles directory are named according to the document set names, prefixed by "p_". For example, the OITS document set style files are stored in the subdirectory p_oits.

Each subdirectory contains three files:

- style.ddd - defines the fields to be included in the partition
- style.dft - defines how the document is to be displayed
- style.dmv - defines how the fields are to be populated

In addition, the style.ddd information is used in the TOPIC Results list and in the Form Query and the Sort specification. The style.ddd fields are also used in other query-types for field searching.

Parts Subdirectories

The subdirectories under the parts directory are named according to the document set names, prefixed by "p_". For example, the OITS document set partition files are stored in the subdirectory p_oits. The files in these subdirectories are generated by TOPIC when a new partition is added. No user intervention is required.

Stopics Directory

The stopics directory normally contains system-defined topics. The CDOCS system does not contain any system topics.

Data Subdirectories

The subdirectories under the data directory are named according to the document set names, prefixed by "p_". for example, the OITS document set data files are stored in the subdirectory p_oits.

The subdirectories under the data directory contain processed copies of the document text that are used to support full-text display and hyperlinks. Under each data subdirectory, there is a "link" and "load" directory. Each document has a copy of the text in both the "link" and "load" directory, and these files have identical names.

The "load" directory contains a single file for each document. The document text is prefixed with the formatted ASCII copy of the document and includes the document text and hyperlinks. The files in the "load" directory are used to support viewing of the document.

The "link" directory contains a single file for each document. The document text is prefixed with the formatted ASCII copy of the document and includes the document text and hyperlinks. The files in the "link" directory are input to the full-text indexing process.

Partind Directory

The partind directory contains the TOPIC database and word list. The files in this directory are generated by TOPIC utility functions. They are updated by CDOCS whenever a document is added, updated or deleted.

Users Directory

The users directory contains all user accounts. The accounts are defined and maintained by the CDOCS system administrator.

Sources Directory

The sources directory contains user defined TOPIC queries. These files have a .qry extension.

Tools Directory

The tools directory contains user defined notes, scripts and special programs. The CDOCS system used many of these scripts to maintain the TOPIC database.

6.6 REPORT WRITER FILE STRUCTURE

The files needed to operate the report writer are organized in a separate set of directories. The directory **/db3/SGMS/dtd** contains both document technical description (**DTD**) files(.dtd) and logic files (.lgc) files. These files are required for document input (entry) and production of output documents. The files in the **/db3/SGMS/dtd/** are as follows:

CDM.lgc	ilarp.dtd	oitsv1r.dtd
CDS.lgc	oitsv1.dtd	oitsv2.dtd
OITS.lgc	kturpt.dtd	oitsv2o.dtd
RPS.lgc	ktutopic.dtd	oitsv2r.dtd
appende.dtd	larp.dtd	oitsv3o.dtd
cdm.dtd	larp2.dtd	readme.txt
ktu.dtd	oits.dtd	writer.log
cdm1.dtd	oitsrupt.dtd	
cds.dtd	oitsv3r.dtd	
cds.lgc	oitsv1o.dtd	

The directory **/db3/SGML/macros** contain macro files (.wpm), library files (.lib), and style files (.001). These files are required for document entry and production of output reports. The files in the **/db3/SGML/dtd** directory are as follows:

001.files	cdsstyle.001	ioitsv1r.wpm
52.sgm	graphics.lib	ioitsv2o.wpm
README.macros	ippende.wpm	ioits2r.wpm
a033001.wpg	icdem.wpm	ioitsv3o.wpm
appende.wpm	icds.wpm	ioitsv3r.wpm
blackend.wpm	ikturpt.wpm	kturpt.wpm
backfrmt.wpm	iktutopi.wpm	ktutopic.wpm
bad_style	ilarp.wpm	larp.wpm
cdmstyl.spm	ioits.wpm	larpstyl.001
cdmstyl.001	ioitsrupt.wpm	larpsup.wpm
cdsstyl.wpm	ioitsv1o.wpm	oitsrpt.wpm

oitsstyl.rpt
oitsstyl.wpm
oitstyle.001
oitsv1o.wpm
oitsv1r.wpm
oitsv2o.wpm
oits2r.wpm
oitsv3o.wpm
oitsv3r.wpm
outstyle.001

rpsstyl.wpm
rpsstyle.001
tables.lib
tappende.wpm
tbackend.wpm
tcdmstyl.wpm
tcdsstyl.wpm
test1.wpm
tgraphics.lib
tiappend.wpm
ticdm.wpm
ticds.wpm

tikturpt.wpm
tiktutop.wpm
tilarp.wpm
tioits.wpm
tioitsrp.wpm
tkturpt.wpm
tktutopi.wpm
tlarp.wpm
toitsrpt.wpm
toitssty.wpm
toitsstyl.wpm
trpsstyl.wpm
ttables.lib

Lastly, the directory **/db3/SGML/rules** contain the rules files (.rul) which are required for document entry (previously called check-in). All of the aforementioned files are listed in Appendices A and B. The files in the **/db3/SGML/rules** directory are as follows:

cdm-sgm.rul
cds-sgm.rul
oits-sgm.rul
rps-sgm.rul

7 INDEXED INTERNAL MODULE COMMENTS—CLIENT

Chapter 7 contains listings of the internal comments for the client custom code modules. An index cross-referencing various functions, actions, and subjects of interest precedes the listings of the C code. The header files (.h) comments are also provided, but are not indexed.

7.1 CLIENT MODULE INDEX

Table 7-1. Client c index

Description	Module
Abort RPD Check In	checkin.c
CDOCS Format Check	tdocserr.c
CDOCS Circulation	circulat.c
CDOCS New Acquisition	circulat.c
Change Password	options.c
Check In CDOCS Record	check.c
Check In RPD Record	checkin.c
Check Out CDOCS Record	check.c
Clean Up Memory	cleanup.c
Cleanup Routine	main.c
Clear Menu	common.c
Client Function Implementation	da_clnt.c
Confirm Action	check.c
Copy File	file.c
Database Login	da_svc.c
Database Logout	da_svc.c
Define RPD Record	define.c
Delete CDOCS Record	ctest.c
Delete Document	check.c
Delete Menu Items	logon.c
Dialog Box Buttons	etcbuttn.c

Table 7-1. Client c index (cont'd)

Description	Module
Edit Functions	edit.c
Error Checking	svc.c
Event Loop	main.c
External Data Representation Format Functions	da_xdr.c
File Chooser	filechsr.c
File Manager	filechsr.c
Force RPD Check In	checkin.c
Forced Check Out	check.c
Format Check	checkin.c
Galaxy Interfaces	rpcvstr.c
Get Check In Names	check.c
Get Control Title	common.c
Get Document Record	check.c
Get Document Record Field Value	check.c
Help	help.c
Help Menus	help.c
Initialize CDOCS	init.c
Load Documents	tdocs.c
Load List	listview.c
Logon	logon.c
Memory Handling	memory.c
Memory Models	memory.c
Open Dialog Box	common.c
Password Subclass	efilter.c
Preferences	prefs.c
Print Reports	printer.c
Reference Reports	report.c

Table 7-1. Client c index (cont'd)

Description	Module
Report Writer	report.c
Reports	subset.c
Retire RPD Record	maint.c
RPC Add User	client.c
RPC Block Event	clntrpc.c
RPC Change Password	client.c
RPC Check In CDOCS Record	client.c
RPC Check Out CDOCS Record	client.c
RPC Client Connect	clntrpc.c
RPC Close Document Info	client.c
RPC Close Document Subset	client.c
RPC Close Document types	client.c
RPC Close Fetch	client.c
RPC Close Help	client.c
RPC Close Reference Report List	client.c
RPC Close Report List	client.c
RPC Close User List	client.c
RPC Close User Names	client.c
RPC Copy	client.c
RPC Delete CDOCS Record	client.c
RPC Document Check	client.c
RPC Document Check In	client.c
RPC Document Define	client.c
RPC Document Retire	client.c
RPC Drop User	client.c
RPC Fetch Check Out Names	client.c
RPC Fetch Document Info	client.c

Table 7-1. Client c index (cont'd)

Description	Module
RPC Fetch Document Subset	client.c
RPC Fetch Document Types	client.c
RPC Fetch Names	client.c
RPC Fetch Para	client.c
RPC Fetch Reference Report List	client.c
RPC Fetch Report List	client.c
RPC Fetch User List	client.c
RPC Fetch User Names	client.c
RPC Free Binary	clntrpc.c
RPC Get Circulation Report	client.c
RPC Get Help Record	client.c
RPC Get Labels	client.c
RPC Get Mismatch	client.c
RPC Get New Acquisition Report	client.c
RPC Get Reference Report	client.c
RPC Get Statistics Report	client.c
RPC Get User Information	client.c
RPC Initialize Row Fetch	client.c
RPC Open Document Info	client.c
RPC Open Document Subset	client.c
RPC Open Document Types	client.c
RPC Open Help	client.c
RPC Open Names List	client.c
RPC Open Reference Report List	client.c
RPC Open Report List	client.c
RPC Open User List	client.c
RPC Open User Names	client.c

Table 7-1. Client c index (cont'd)

Description	Module
RPC Read Binary File	clntrpc.c
RPC Report	client.c
RPC Report Writer	client.c
RPC Request (General)	clntrpc.c
RPC Server Connect	client.c
RPC SQL Close Report	client.c
RPC SQL Delete Report	client.c
RPC SQL Fetch Query	client.c
RPC SQL Fetch Report	client.c
RPC SQL Open Query	client.c
RPC SQL Open Report	client.c
RPC SQL Select Report	client.c
RPC SQL Update Report	client.c
RPC Submit CDOCS Record	client.c
RPC Update CDOCS Record	client.c
RPC Write ASCII File	clntrpc.c
RPC Write Binary Files	clntrpc.c
RPD Server Disconnect	client.c
Scan	scan.c
SQL Errors Returned	svc.c
Submit CDOCS Records	submit.c
Subset Reports	subset.c
TOPIC Launch	search.c
Update CDOCS Document	check.c
Update CDOCS Record	ctest.c
User ID Maintenance	options.c
User Preferences	prefs.c

Table 7-1. Client c index (cont'd)

Description	Module
Utility Routines	utils.c
Validate Name Syntax	check.c
WordPerfect Launch to Client	file.c

7.2 CLIENT LISTING

7.2.1 Client C

```

/*-----
* File name:      check.c
* Date:           3/28/94
* Author:         dtl
* Description:     This file contains the routines to check a CDOCS
*                  document in and out, delete a CDOCS document and
*                  confirm CDOCS deletions.
*
*-----
* Archive:        /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.check.c
* Author:
* Revision:       2.0.6
* Date:           94/03/28-16:38:08
*-----
*
* Encompasses: Checkout, CheckoutProc, RPCCheckout, LoadCheckout,
*              ForceCheckout, ForceCheckoutProc, LoadForceCheckout,
*              TDOCSCheckin, CheckinProc, RPCCheckin, LoadTDOCSCheckin,
*              TDOCSCheckinNames, CheckinNamesProc, RPCCheckinNames,
*              LoadTDOCSCheckinNames,
*              Delete, DeleteProc, RPCDelete, LoadDelete,
*              Confirm, ConfirmProc, RPCGetRec, LoadConfirm,
*              NamesSelectProc, RPCGetNames,
*              TDOCSUpdate, TDOCSUpdateProc, LoadTDOCSUpdate, TDOCSUpdateGetRec,
*              FetchNames, ValidateName, GetTDOCSRecordFieldValue,
*              RemoveExtraneousWhiteSpace
*-----
*/

/* FORCE CHECKOUT PROCEDURES */

```

```

/*
 * Function:   LoadForceCheckout
 *
 * Description: Load the Force Checkout dialog box and establish
 *              call backs for the OK and Cancel buttons
 * Parameters: None
 * Returns:    Nothing
 */

```

```

/*
 * Function:   ForceCheckout
 *
 * Description: Open the Force Checkout dialog box
 * Parameters: None
 * Returns:    Nothing
 */

```

```

/*
 * Function:   ForceCheckoutProc
 *
 * Description: Determine whether the user wishes to check out
 *              a document or cancel the function
 * Parameters: button - ptr to button chosen: OK, or CANCEL
 *              event - ptr to current event
 * Returns:    Nothing
 */

```

```

/* CHECKOUT PROCEDURES */

```

```

/*
 * Function:   LoadCheckout
 *
 * Description: Load the checkout dialog box and establish
 *              call backs for the OK and Cancel buttons
 * Parameters: None
 * Returns:    Nothing
 */

```

```

/*
 * Function:   Checkout
 *
 * Description: Open the Checkout dialog box
 * Parameters: None
 * Returns:    Nothing
 */

```

```

/*
* Function:   CheckoutProc
*
* Description: Determine whether the user wishes to check out
*              a document or cancel the function
* Parameters: button - ptr to button chosen: OK, or CANCEL
*              event - ptr to current event
* Returns:    Nothing
*/

/*
* Function:   RPCCheckout()
*
* Description: Make the RPC call to check out a record
* Parameters: documentSet - TDI, QA, CSP
*              documentNumber - document assession number
*              name - person who submitted the record
* Returns:    Success or failure
*/

/*
* Function:   RemoveExtraneousWhiteSpace
*
* Description: Remove leading and trailing white space from a string
* Parameters: str - ptr to string
* Returns:    Nothing
*/

/* CHECKIN PROCEDURES */

/*
* Function:   LoadTDOCSCheckin
*
* Description: Load the Checkin dialog box and establish
*              call backs for the OK and Cancel buttons
* Parameters: None
* Returns:    Nothing
*/

/*
* Function:   TDOCSCheckin
*
* Description: Open the Checkin dialog box
* Parameters: None
* Returns:    Nothing
*/

```

```

/*
* Function:    CheckinProc
*
* Description: Allow the user to check a document in
*              or cancel the function
* Parameters: button - ptr to button chosen: OK or CANCEL
*              event - ptr to current event
* Returns:    Nothing
*/

/*
* Function:    RPCCheckin()
*
* Description: Make the RPC call to check in a record
* Parameters: documentSet - TDI, QA, CSP
*              documentNumber - document assession number
* Returns:    Success or failure
*/

/* CHECKIN NAMES PROCEDURES */

/*
* Function:    LoadTDOCSCheckinNames
*
* Description: Load the Checkin Names dialog box and establish
*              call backs for the OK and Cancel buttons
* Parameters: None
* Returns:    Nothing
*/

/*
* Function:    TDOCSCheckinNames
*
* Description: Open the Checkin names dialog box
* Parameters: None
* Returns:    Nothing
*/

/*
* Function:    CheckinNamesProc
*
* Description: Allow the user to check a document in
*              or cancel the function
* Parameters: button - ptr to button chosen: OK or CANCEL
*              event - ptr to current event
* Returns:    Nothing
*/

```



```

/*
* Function:   RPCCheckinNames()
*
* Description: Make the RPC call to check in a record
* Parameters: None
* Returns:    Success or failure
*/

/*  DELETE PROCEDURES */

/*
* Function:   LoadDelete
*
* Description: Load the Delete dialog box
* Parameters: None
* Returns:    Nothing
*/

/*
* Function:   Delete
*
* Description: Open the Delete dialog box
* Parameters: None
* Returns:    Nothing
*/

/*
* Function:   DeleteProc
*
* Description: Allow the user to delete a document
*              or cancel the function
* Parameters: button - ptr to button chosen: OK or CANCEL
*              event - ptr to current event
* Returns:    Nothing
*/

/*
* Function:   RPCDelete()
*
* Description: Make the RPC call to delete a record
* Parameters: documentSet - TDI, QA, CSP
*              documentNumber - document assession number
*              deleteShare - delete on remote server?
* Returns:    Success or failure
*/

/*  CONFIRM PROCEDURES */

```

```

/*
 * Function:    Confirm
 *
 * Description: Open the Confirm dialog box
 * Parameters:  operation - which operation: delete, check in, check out
 * Returns:     Nothing
 */

/*
 * Function:    LoadConfirm
 *
 * Description: Load the Confirm dialog box
 * Parameters:  None
 * Returns:     Nothing
 */

/*
 * Function:    ConfirmProc
 *
 * Description: Allow the user to confirm or cancel an operation
 * Parameters:  button - ptr to button chosen: OK or CANCEL
 *              event - ptr to current event
 * Returns:     Nothing
 */

/* NAMES SELECT PROCEDURES */

/*
 * Function:    NamesSelectProc
 *
 * Description: Determine whether the user wishes to select a list of
 *              names or cancel the function
 * Parameters:  NDialog - ptr to Names dialog
 *              NamesText - specific name to select
 *              NamesList - list of all user names
 * Returns:     Success or Failure
 */

/*
 * Function:    ValidateName
 *
 * Description: Check the syntax of a user defined name for TDOCS submission
 *              for checkout or circulation reports
 * Parameters:  nameVstr - name to validate
 * Returns:     Success or failure
 */

```

```

/*
* Function:   RPCGetNames
*
* Description: Make the RPC call to get a list of valid user names
* Parameters: NamesDialog - ptr to Names dialog box
*              NamesList - ptr to list of valid user names
* Returns:    Success or failure
*/

/*
* Function:   FetchNames
*
* Description: Make the RPC call to fetch the name list from the
*              Oracle database
* Parameters: namesDialog - ptr to Names dialog box
*              NamesList - ptr to list of valid user names
*              pResults - results structure
* Returns:    Success or failure
*/

/*  UPDATE PROCEDURES */

/*
* Function:   LoadTDOCSUpdate
*
* Description: Load the TDOCS Update dialog box
* Parameters: None
* Returns:    Nothing
*/

/*
* Function:   TDOCSUpdate
*
* Description: Open the TDOCS Update dialog box
* Parameters: None
* Returns:    Nothing
*/

/*
* Function:   TDOCSUpdateProc
*
* Description: Allow the user to update a document
*              or cancel the function
* Parameters: button - ptr to button chosen: OK or CANCEL
*              event - ptr to current event
* Returns:    Nothing
*/

```

```

/*
* Function:   TDOCSUpdateGetRec
*
* Description: Retrieve the requested TDOCS record
* Parameters: None
* Returns:    Nothing
*/

/*
* Function:   RPCGetRec()
*
* Description: Make the RPC call to get a TDOCS record
* Parameters: documentSet - TDI, QA, CSP
*              documentNumber - document assessment number
*              record - record contents
* Returns:    Success or failure
*/

/*
* Function:   GetTDOCSRecordFieldValue()
*
* Description: Get the value of the TDOCS field
* Parameters: record - the SQL record of the current document
*              fieldName - field name (e.g. TITLE)
*              fieldValue - value of field name (e.g. "Preliminary Report")
* Returns:    Value of the TDOCS field
*/

```

```

/*-----
* File name:  checkin.c
* Date:      10/5/93 1345
* Author:    alj,rlm
* Description: Checkin is used to checkin an SRA record after
*             it has been defined. A filename representing
*             the WordPerfect document is given to checkin.
*-----
*
* 03/31/94 - rlm - v1.2 modify to remove ORACLE calls and use RPC
*             server exclusively
*-----
* Archive:    /home/goliath/rmarsha/tdocs/development/interface/SCCS/s.checkin.c
* Author:
* Revision:   1.4.0
* Date:       94/05/03-16:38:08
*-----
*
* Encompasses:    loadCheckin,closeCheckin,checkin,
*                 applyFileName,checkinfilechooser
*-----
*/

/*
* Function:    loadCheckin
* Description: Display the checkin dialog
*
* Parameters:  None
* Returns:     Nothing
*/

/*
* Function:    closeCheckin
* Description: Close the Checkin dialog and perform any cleanup necessary.
*
* Parameters:  None
* Returns:     Nothing
*/

/*
* Function:    checkin
* Description: Checks in the SRA record.
*             ORACLE Fields are updated and header & file submitted
*             to server for TOPIC and WP Repository.
* Parameters:  void (reads from dialog)
* Returns:     0 = OK, -1 = Error
*/

```

```

/*
 * Function versionMismatch deleted
 *
 * void versionMismatch( vchar *version, vchar *msg)
 *
 */

/*
 * Function:    forcecheckin
 * Description: forces checkin after a title mismatch.
 * Parameters:  void (reads from dialog)
 * Returns:     0 = OK, -1 = Error
 */

/*
 * Function:    abortcheckin
 * Description: aborts checkin after a title mismatch.
 *              ORACLE transaction is rolled-back
 * Parameters:  void
 * Returns:     0 = OK, -1 = Error
 */

/*
 * Function:    loadFormatCheck
 * Description: Display the format check dialog
 *
 * Parameters:  None
 * Returns:     Nothing
 */

/*
 * Function:    closeFormatCheck
 * Description: Close the FormatCheck dialog and perform any cleanup necessary.
 *
 * Parameters:  None
 * Returns:     Nothing
 */

/*
 * Function:    formatCheck
 * Description: Format checks the SRA record.
 *              File submitted
 *              to server for TOPIC and WP Repository.
 * Parameters:  void (reads from dialog)
 * Returns:     0 = OK, -1 = Error
 */

```

```
/*
* Function:   abortformatcheck
* Description: displays errors on format check.
* Parameters: void
* Returns:    0 = OK, -1 = Error
*/

/*
* Function:   GetOptionMenuTitle
* Description: Return the title string from the option menu
*              after a selection
* Parameters: dialog - Name of the dialog box
* Returns:    dialog box title
*/
```

```

/*-----
* File name:  circulat.c
* Date:      5/12/94
* Author:    dtl
* Description: Functions to create circulation reports
*
*-----
* Archive:      /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.circulat.c
* Author:
* Revision:     2.0.6
* Date:         94/05/03-16:38:08
*-----
*
* Encompasses:  CirculationProc, Circulation, GetWhichCirculationReport,
*               LoadCirculation, NewDocsProc, NewDocs,
*               LoadNewDocs, CirculationGetNames, showSaveAsText
*-----
*/

/*
* Function:    Circulation
*
* Description: Open the Circulation dialog box
* Parameters:  None
* Returns:     Nothing
*/

/*
* Function:    LoadCirculation
*
* Description: Load the circulation dialog box and
*               establish call backs for the OK and
*               Cancel buttons
* Parameters:  None
* Returns:     Nothing
*/

/*
* Function:    CirculationProc
*
* Description: Determine whether the user wishes to get a
*               report or cancel the function.
* Parameters:  button - ptr to button chosen: OK, or CANCEL
*               event - ptr to current event
* Returns:     Nothing
*/

```



```

/*
* Function:   CirculationGetNames
*
* Description: Open the Names Select dialog box and
*              enable/disable the appropriate controls
* Parameters: control - ptr to current control
*              event - ptr to current event
* Returns:    Nothing
*/

/*
* Function:   GetWhichCirculationReport
*
* Description: Get which circulation report to generate.
*              In the case of "Who has a given doc,"
*              return Doc #, and in the case of "Which doc
*              does a given person have", return Who requested
*              the search.
* Parameters: None
* Returns:    Nothing
*/

/*
* Function:   NewDocs
*
* Description: Open the New Docs dialog box
* Parameters: None
* Returns:    Nothing
*/

/*
* Function:   LoadNewDocs
*
* Description: Load the new docs dialog box and
*              establish call backs for the OK and
*              Cancel buttons
* Parameters: None
* Returns:    Nothing
*/

/*
* Function:   NewDocsProc
*
* Description: Determine whether the user wishes to get a
*              new acquisition report or cancel the operation
* Parameters: button - ptr to button chosen: OK, or CANCEL
*              event - ptr to current event
* Returns:    Nothing
*/

```

```
/*  
* Function:    showSaveAsText  
*  
* Description: Display CDOCS reports and Database Report Writer reports  
* Parameters: title - title of CDOCS report dialog box  
* Returns:    Nothing  
*/
```

```

/*-----
* cleanup.c
*
* 09/20/93 - rlm - created.
* 03/25/94 - rlm - modified for rpd v1.1 development
* 05/23/94 - alj - modified for changes to HELP dialogs
* 05/26/94 - alj - modified for cleanup of HELP Custodian Dialog
* 07/26/94 - alj - modified for cleanup or HELP dialogs for TDOCS
*-----
* Archive:    /home/goliath/rmarsha/v1.1/development/interface/SCCS/s.cleanup.c
* Author:
* Revision:   1.1.0
* Date:       94/03/10-15:03:37
*-----
*
* encompasses:  cleanup().
*-----
*/

/*
* Function:    cleanup
* Description: Destroys dialogs and frees resources at end of job.
* Parameters:  The inputs are the external dialogs defined in main.c
* Returns:     None.
*/

```

```

/*
* client.c
*
* What's new:
* RPCReportWriter - can do document generation - 06/27/95 - cjm
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/rpc/SCCS/s.client.c
* Author:
* Revision:   1.26.0
* Date:       95/08/30-10:02:04
*-----
*
* RPDOCS interface to client rpc.
*
* On startup, call RPCCConnect, RPC*, and, on exit, RPCDisconnect.
*
* *Important* Interface must not free data received from client.
*
* Code below is written for the following:
*
* platform      compiler      defined by compiler
*
* SMI sun        cc            __sparc
* MS windows     Watcom C/C++   __WATCOMC__
* Microsoft      C7            __MSC__
* IBM OS/2       IBM C/C++ Set  __OS2__
* Apple Macintosh Semantic C    __MAC__
*
* __MSC__ used to build the rpc16.dll only -- it and __WATCOMC__
* are defined in makefile.win.
*
* 08/12/94 - cjm - moved constants to dadefine.h
* 08/26/94 - cjm - windows port
* 09/07/94 - cjm - merged with other ports
* 09/16/94 - cjm - merged in OS/2 port
* 02/07/95 - cjm - solaris port (removed gnuc, acc, borland stuff)
* 06/07/95 - rlm - add conditionals to get proper os2 rpc support
* 06/22/95 - cjm - dynamic versioning
*-----
*/

```

```

/*
 * global variables
 */

char dbaccess[256]; /* db access */
extern
tRPCResults results; /* status, error, rows, data */
pRPCResults pResults=NULL; /* global pointer to results */
tRPCFetch fetch; /* fetch columns, row */

/*

/*****
 *
 * RPDOCS
 *
 *****/
/*
 * RPCCConnect
 *
 * Connects to rpc server and sets timeout. Returns rpc results.
 *
 * Called by login.
 *
 * Calls _RPCCConnect.
 */

/*
 * RPCCDisconnect
 *
 * Disconnects from rpc server - actually just informs server of exit.
 *
 * Called by main.
 *
 * Calls _RPCRequests.
 */

/*
 * RPCCOpenHelpListCursor
 *
 * Fetches help list for privilege level. Returns rpc results.
 *
 * Called by BuildHelpIndex
 *
 * Calls _RPCRequest, _RPCInitFetchData.
 */

```

```

/*
 * RPCFetchHelpListRow
 *
 * Fetches help key, subkey, header, or -2, -2, "" at end of list.
 *
 * Called by BuildHelpIndex
 *
 * Calls _RPCFetchData.
 */

/*
 * RPCCloseHelpListCursor
 *
 * Cleans up after fetching help list.
 *
 * Called by BuildHelpIndex
 *
 * Calls _RPCCloseFetch.
 */

/*
 * RPCGetHelp
 *
 * Gets help header and text given privilege, key, and subkey. Returns rpc
 * results.
 *
 * Called by ShowHelp
 *
 * Calls _RPCRequests.
 */

/*
 * RPCGetUserInfo
 *
 * Gets privilege level and name for user identified by user id. Returns rpc
 * results.
 *
 * Called by login.
 *
 * Calls _RPCRequests.
 */

```

```

/*
 * RPCTOpenUserNamesCursor
 *
 * Fetches all user names and returns status, possibly error, and rows.
 * Returns rpc results.
 *
 * Called by None
 *
 * Calls _RPCRequests, _RPCInitFetchData.
 */

```

```

/*
 * RPCTFetchUserNamesRow
 *
 * Fetches a user name or "" at end of list from list.
 *
 * Called by None
 *
 * Calls _RPCFetchData.
 */

```

```

/*
 * RPCTCloseUserNamesCursor
 *
 * Cleans up after fetching user names.
 *
 * Called by None
 *
 * Calls _RPCTCloseFetch.
 */

```

```

/*
 * RPCTOpenUserListCursor
 *
 * Fetches all users' ids, privileges, and names. Returns rpc results.
 *
 * Called by loadUserIDs.
 *
 * Calls _RPCRequest, _RPCInitFetchData.
 */

```

```

/*
 * RPCFetchUserListRow
 *
 * Fetches a user's id, privilege, and name or "" at end of list.
 *
 * Called by loadUserIDs.
 *
 * Calls _RPCFetchData.
 */

/*
 * RPCCloseUserListCursor
 *
 * Cleans up after fetching user info.
 *
 * Called by loadUserIDs.
 *
 * Calls _RPCCloseFetch.
 */

/*
 * RPCChangePassword
 *
 * Changes password of user identified by userid. Returns rpc results.
 *
 * Called by changePassword.
 *
 * Calls _RPCRequest.
 */

/*
 * RPCAddUser
 *
 * Adds a rpd user identified by userid, passwd, privilege and name. Returns
 * rpc results.
 *
 * Called by addUserIDs.
 *
 * Calls _RPCRequest.
 */

/*
 * RPCDropUser
 *
 * Drops a rpd user identified by userid. Returns rpc results.
 *
 * Called by deleteUserIDs.
 *
 * Calls _RPCRequest.
 */

```



```

/*
 * RPCOpenDocTypeCursor
 *
 * Fetches all document types and returns status. Returns rpc results.
 *
 * Called by loadDocTypes.
 *
 * Calls _RPCRequest, _RPCInitFetchData.
 */

/*
 * RPCFetchDocTypeRow
 *
 * Fetches a document type or "" at end of list.
 *
 * Called by loadDocTypes.
 *
 * Calls _RPCFetchData.
 */

/*
 * RPCCloseDocTypeCursor
 *
 * Cleans up after fetching document types.
 *
 * Called by loadDocTypes.
 *
 * Calls _RPCCloseFetch.
 */

/*
 * RPCOpenDocInfoCursor
 *
 * Fetches all document information (review plan number, title, and version)
 * and returns status. Returns rpc results.
 *
 * Called by loadDocs.
 *
 * Calls _RPCRequest, _RPCInitFetchData.
 */

```

```

/*
 * RPCFetchDocInfoRow
 *
 * Fetches a document's review plan number and title or "" at end of list.
 *
 * Called by loadDocs.
 *
 * Calls _RPCFetchData.
 */

```

```

/*
 * RPCCloseDocInfoCursor
 *
 * Cleans up after fetching document types.
 *
 * Called by loadDocs.
 *
 * Calls _RPCCloseFetch.
 */

```

```

/*
 * RPCOpenDocSubsetCursor
 *
 * Fetches all document number and title for those consistent with title-
 * identified report query conditions and returns status. Returns rpc results.
 *
 * Called by loadSubsetDocs
 *
 * Calls _RPCRequest, _RPCInitFetchData.
 */

```

```

/*
 * RPCFetchDocSubsetRow
 *
 * Fetches a document's number and title or "" at end of list.
 *
 * Called by loadSubsetDocs
 *
 * Calls _RPCFetchData.
 */

```

```

/*
 * RPCCloseDocSubsetCursor
 *
 * Cleans up after fetching document numbers and titles.
 *
 * Called by loadSubsetDocs
 *
 * Calls _RPCCloseFetch.
 */

/*
 * RPCDefineDocument
 *
 * Sends document definition to server. Returns rpc results.
 *
 * Called by define.
 *
 * Calls _RPCRequest.
 */

/*
 * RPCCheckFormat
 *
 * Sends document for format check. Returns rpc results.
 *
 * Called by formatCheck.
 *
 * Calls _RPCRequest.
 */

/*
 * RPCCheckinDocument
 *
 * Sends document for attempted or forced checkin to server. Note that for
 * OITS num is oitsid, and that for any document type where user does not
 * specify version information majmin is MAJOR_VERSION_CHANGE and ver is "".
 * Returns rpc results.
 *
 * Called by checkin.
 *
 * Calls _RPCRequest.
 */

```

```

/*
 * RPCGetMismatches
 *
 * Returns mismatches. Values returned will be either expected or received
 * values or "". Mismatches are stored in returned data as follows:
 *
 * (L\2E3R\1)*
 *
 * where L labels mismatch, E is expected value, and R is received value. They
 * are ordered, if they appear, document type, review plan number, title.
 * Mismatches will involve either version or document type, review plan num and
 * title.
 *
 * Called by formatCheck, checkin.
 *
 * Calls nothing.
 */

```

```

/*
 * RPCRetireDocument
 *
 * Sends document retirement to server. Returns rpc results.
 *
 * Called by reallyretire.
 *
 * Calls _RPCRequest.
 */

```

```

/*
 * RPCCopyWP
 *
 * Requests by document type and review plan number a WP document and writes it
 * to file path. Returns rpc results.
 *
 * Called by filecopy.
 *
 * Calls _RPCRequest.
 */

```

```

/*
 * RPCReport
 *
 * Requests and writes status, content, or dba report to file identified by
 * file path. Returns rpc results.
 *
 * Called by loadreport.
 *
 * Calls _RPCRequest.
 */

```

```

/*
 * RPCTOpenRptListCursor
 *
 * Fetches all report writer reports, titles, privileges, and helps. Returns
 * rpc results.
 *
 * Called by MenuReportWriter.
 *
 * Calls _RPCRequest, _RPCInitFetchData.
 */

/*
 * RPCTFetchRptListRow
 *
 * Fetches a report writer report, title, privilege, and help; or "" at end of
 * list.
 *
 * Called by MenuReportWriter.
 *
 * Calls _RPCFetchData.
 */

/*
 * RPCTCloseRptListCursor
 *
 * Cleans up after fetching report writer reports, titles,
 * types, and helps.
 *
 * Called by MenuReportWriter.
 *
 * Calls _RPCCloseFetch.
 */

/*
 * RPCTReportWriter
 *
 * Request a report writer report. If title={CDS,CDM,OITS}, type="GENERATE",
 * and nums=<a docnum>, a single document will be generated and returned.
 * Returns rpc results.
 *
 * Called by MenuReportWriterProc.
 *
 * Calls _RPCRequest.
 */

```

```

/*****
*
* SQL*DTD
*
*****/
/*
* RPCSQLDTDOpenReports
*
* Fetches all report reports (ids), titles, and help, and returns status,
* possibly error, and rows. Returns rpc results.
*
* Called by None
*
* Calls _RPCRequests, _RPCInitFetchData.
*/

/*
* RPCSQLDTDFetchReport
*
* Fetches report, title, and help; or '', '', and '' at end of list.
*
* Called by None
*
* Calls _RPCFetchData.
*/

/*
* RPCSQLDTDCloseReports
*
* Cleans up after fetching reports.
*
* Called by None
*
* Calls _RPCCloseFetch.
*/

/*
* RPCSQLDTDSelectReport
*
* Retrieves report data by report id. Returns rpc results. Status SVC_OP_ERROR,
* error SVC_DATA_FIND_ERROR indicates the report does not exist.
*
* Called by None
*
* Calls _RPCRequest.
*/

```

```

/*
 * RPCSQLDTDInsertReport
 *
 * Inserts/creates report data. Returns rpc results.
 *
 * Called by None
 *
 * Calls _RPCRequest.
 */

/*
 * RPCSQLDTDUpdateReport
 *
 * Updates report data by report id. Returns rpc results.
 *
 * Called by None
 *
 * Calls _RPCRequest.
 */

/*
 * RPCSQLDTDDeleteReport
 *
 * Deletes report data by report id. Returns rpc results.
 *
 * Called by None
 *
 * Calls _RPCRequest.
 */

/*
 * RPCSQLDTDOpenQuery
 *
 * Fetches all report tags and search text with query, and returns
 * status, possibly error, and rows. Note that query syntax should be "select
 * tag,piece from piece..." which server will convert to "select tag,search
 * from piece... ." To query on a subset of review plans/oits ids then the
 * conditional part should read something like "...DOCNUM IN (%s)" and subset
 * should be a space-delimited review plan numbers or oits ids that the server
 * will substitute into the query. Returns rpc results.
 *
 * Called by None
 *
 * Calls _RPCRequests, _RPCInitFetchData.
 */

```

```

/*
 * RPCSQLDTDFetchQuery
 *
 * Fetches tags and text from SQL*DTD report query.
 *
 * Called by None
 *
 * Calls _RPCFetchData.
 */

/*
 * RPCSQLDTDCloseQuery
 *
 * Cleans up after fetching tags and text.
 *
 * Called by None
 *
 * Calls _RPCCloseFetch.
 */

/*
 * RPCSQLDTDGetFile
 *
 * Retrieves WP macro (<report>.wpm) or Intellitag DTD (<report>.dtd) file to
 * server for storage. Returns rpc results.
 *
 * Called by None
 *
 * Calls _RPCRequest.
 */

/*
 * RPCSQLDTDPutFile
 *
 * Sends WP macro (<report>.wpm) or Intellitag DTD (<report>.dtd) file to
 * server for storage. Returns rpc results.
 *
 * Called by None
 *
 * Calls _RPCRequest.
 */

```



```

/*****
 *
 * TDOCS
 *
 *****/
/*
 * RPCTOpenDocSetCursor
 *
 * Fetches all document sets and returns status. Returns rpc results.
 *
 * Called by BuildDocSet
 *
 * Calls _RPCRequest, _RPCInitFetchData.
 */

/*
 * RPCTFetchDocSetRow
 *
 * Fetches a document set or "" at end of list.
 *
 * Called by BuildDocSet
 *
 * Calls _RPCFetchData.
 */

/*
 * RPCTCloseDocSetCursor
 *
 * Cleans up after fetching document sets.
 *
 * Called by BuildDocSet
 *
 * Calls _RPCTCloseFetch.
 */

/*
 * RPCTGetDocSetSpec
 *
 * Retrieves a document set specification. Returns rpc results.
 *
 * Called by SetFieldEntryButtons, ValidateSubmitFields
 *
 * Calls _RPCRequest.
 */

```

```
/*
 * RPCGetTDOCSRecord
 *
 * Retrieves a tdocs record. Returns rpc results.
 *
 * Called by RPCGetRec
 *
 * Calls _RPCRequest.
 */
```

```
/*
 * RPCSubmitRecord
 *
 * Submits a tdocs record. Returns rpc results.
 *
 * Called by submitRecord.
 *
 * Calls _RPCRequest.
 */
```

```
/*
 * RPCUpdateRecord
 *
 * Updates a tdocs record. Returns rpc results.
 *
 * Called by submitRecord.
 *
 * Calls _RPCRequest.
 */
```

```
/*
 * RPCDeleteRecord
 *
 * Deletes (i.e., marks for deletion) a tdocs record. Returns rpc results.
 *
 * Called by RPCDelete.
 *
 * Calls _RPCRequest.
 */
```

```
/*
 * RPCCheckoutRecord
 *
 * Checks out (i.e., marks as checked out) a tdocs record. Returns rpc results.
 *
 * Called by RPCCheckout.
 *
 * Calls _RPCRequest.
 */
```

```

/*
 * RPCCheckinRecord
 *
 * Checks in (i.e., marks as checked in) a tdocs record. Returns rpc results.
 *
 * Called by RPCCheckin.
 *
 * Calls _RPCRequest.
 */

/*
 * RPCFetchCheckoutNames
 *
 * Fetches names of people who have a document checked out.
 *
 * Called by FetchNames
 *
 * Calls _RPCFetchData.
 */

/*
 * RPCGetLabels
 *
 * Requests labels for newly submitted records. Returns rpc results.
 *
 * Called by GetLabel.
 *
 * Calls _RPCRequest.
 */

/*
 * RPCGetNewDocsReport
 *
 * Requests report on new documents in tdocs from a given date to a given date.
 * Returns rpc results.
 *
 * Called by NewDocsProc.
 *
 * Calls _RPCRequest.
 */

/*
 * RPCGetCirculationReport
 *
 * Requests report on circulation. Returns rpc results.
 *
 * Called by CirculationProc.
 *
 * Calls _RPCRequest.
 */

```

```

/*
 * RPCOpenRefRptListCursor
 *
 * Fetches all reference report writer titles and helps, and returns status.
 * Returns rpc results.
 *
 * Called by RefReport
 *
 * Calls _RPCRequest, _RPCInitFetchData.
 */

```

```

/*
 * RPCFetchRefRptListRow
 *
 * Fetches a report writer title and help; or "" at end of list.
 *
 * Called by RefReport
 *
 * Calls _RPCFetchData.
 */

```

```

/*
 * RPCCloseRefRptListCursor
 *
 * Cleans up after fetching report writer titles and helps.
 *
 * Called by RefReport
 *
 * Calls _RPCCloseFetch.
 */

```

```

/*
 * RPCGetReferenceReport
 *
 * Requests reference report by title. Returns rpc results.
 *
 * Called by RefReportProc
 *
 * Calls _RPCRequest.
 */

```

```

/*
 * RPCGetStatisticsReport
 *
 * Requests statistics report. Returns rpc results.
 *
 * Called by loadreport
 *
 * Calls _RPCRequest.
 */

```

```

/*
 * RPCOpenNamesCursor
 *
 * Fetches all people names. Returns rpc results.
 *
 * Called by RPCGetNames
 *
 * Calls _RPCRequest, _RPCInitFetchData.
 */

/*
 * RPCFetchNamesRow
 *
 * Fetches a people name; or "" at end of list.
 *
 * Called by FetchNames
 *
 * Calls _RPCFetchData.
 */

/*
 * _RPCInitFetchData
 *
 * Initializes row fetching by _RPCFetchData and sets rows fetched.
 *
 * Called by RPCOpenUserNamesCursor, RPCOpenUserListCursor,
 * RPCOpenDocTypeCursor, RPCOpenDocInfoCursor, RPCOpenRptListCursor,
 * RPCOpenRefRptListCursor, RPCOpenTDOCSOrgCursor,
 * RPCOpenTDOCSOrgNamesCursor.
 *
 * Calls nothing.
 */

/*
 * _RPCFetchData
 *
 * Decodes returned data as a set of data lists encoded as
 *
 * (V(3V)*1)*
 *
 * First call _RPCInitFetchData. Returns 0 on success or 1 on failure.
 *
 * Called by RPCFetchUserNamesRow, RPCFetchUserListRow,
 * RPCFetchDocTypeRow, RPCFetchDocInfoRow, RPCFetchRptListRow,
 * RPCFetchRefRptListRow, RPCFetchTDOCSOrgRow,
 * RPCFetchTDOCSOrgNamesRow.
 *
 * Calls nothing.
 */

```

```
/*  
* _RPCCloseFetch  
*  
* Frees fetch data.  
*  
* Called by RPCCloseHelpListCursor  
*  
* Calls nothing.  
*/
```

```

/*
 * clntrpc.c
 *
 * -----
 * Archive:    /home/samson/dlincoln/rpdocs/development/rpc/SCCS/s.clntrpc.c
 * Author:
 * Revision:   1.20.0
 * Date:       95/08/17-14:05:35
 * -----
 *
 * Nonportable RPC interface to da server.
 *
 * Code below is written for the following:
 *
 * platform      compiler      define
 *
 * SMI sun        cc            __sparc
 * MS windows     Watcom C/C++  __WATCOMC__ (__WINDOWS_386__)
 * Microsoft      C7            __MSC__
 * IBM OS/2        IBM C/C++ Set __OS2__
 * Apple Macintosh Semantic C    THINK_C
 *
 * __MSC__ used to build the rpc16.dll only -- it and __WATCOMC__
 * are defined in makefile.win.
 *
 * 03/06/94 - cjm
 * 03/28/94 - rlm      - changed include from rpc.h to rpc/rpc.h to
 *                      match Makefile requirements
 * 04/05/94 - rlm      - added conditional compilation directives to
 *                      support os2 port
 * 04/06/94 - cjm      - added tdocs calls
 * 08/10/94 - dtl/cjm   - modified for windows port
 * 08/26/94 - cjm      - windows port
 * 09/07/94 - cjm      - merged with other ports
 * 09/16/94 - cjm      - merged in OS/2 port
 * 02/07/95 - cjm      - solaris port (removed gnuc, acc, borland stuff)
 * 06/22/95 - cjm      - dynamic versioning
 * -----
 */

/*
 * _RPCConnect
 *
 * Connects to rpc server and sets timeout. Implemented once for each
 * compiler/platform. Return rpc results.
 *
 * Called by RPCConnect.
 *

```

```

* Calls _RPCBlockEvents, __RPCConnect.
*/

/*
* _RPCRequest
*
* Requests service of rpc server. All packet building and file i/o
* is handled here. Implemented specially for Watcom. Returns 0 on success
* or 1 on failure.
*
* Called by all RPC functions.
*
* Calls _RPCReadBinaryFile, rpc_N, _RPCFreeBinary, _RPCWriteBinaryFile,
* _RPCWriteAsciiFile, _RPCBlockEvents, __RPCRequest, __RPCResults, __RPCData.
*/

/*
* _RPCBlockEvents
*
* Blocks user input during client/server transaction.
*
* NetManage rpc in Windows is non-blocking. If transactions
* were not blocked the user could continue to request services
* while the client was waiting for the results of the previous
* request. Not sure what the consequences of that would be but
* were the user to exit the application while the client was
* waiting for results his or her entire system would become
* unstable and likely crash.
*
* The current implementation is system modal, i.e., it blocks
* all Windows events to all applications DURING the transaction.
* The user can, however, generate events which the window manager
* queues and then releases when the block is released.
*
* It may be possible to make the block application modal by
* removing the set and release capture functions.
*
* Called by _RPCConnect, _RPCRequest.
*
* Calls nothing.
*/

```



```

/*
 * _RPCReadBinaryFile
 *
 * Reads binary file to packet binary. Sets results status to SVC_SUCCESS,
 * CLNT_MEM_ERROR, or CLNT_FILE_ERROR (in which case error is set to system
 * error). Returns 0 on success or 1 on failure.
 *
 * Called by _RPCRequest.
 *
 * Calls _RPCFreeBinary.
 */

```

```

/*
 * _RPCFreeBinary
 *
 * Frees packet binary.
 *
 * Called by _RPCRequest.
 *
 * Calls nothing.
 */

```

```

/*
 * _RPCWriteAsciiFile
 *
 * Writes ascii list to file named by filename. Frees list nodes. Sets
 * results status and, possibly, error. Returns 0 on success or 1 on failure.
 *
 * Called by _RPCRequest.
 *
 * Calls nothing.
 */

```

```

/*
 * _RPCWriteBinaryFile
 *
 * Writes binary list to file named by filename. Frees list nodes.
 *
 * Sets results status and, possibly, error.
 *
 * filepath - path to file to write to.
 *
 * Returns 0 on success or 1 on failure.
 *
 * Called by _RPCRequest.
 *
 * Calls nothing.
 */

```

```

/*-----
* File name:  common.c
* Date:      9/28/93 1750
* Author:    rlm
* Description: Common contains functions used commonly
*             throughout the application. Common should
*             grow and become common to all Galaxy-based
*             applications.
*-----
*
* 03/28/94 - rlm -  v1.2 removed SQL client calls and convert to
*                  rpd v1.1 rpc server
*
*-----
* Archive:    /home/goliath/rmarsh/v1.1/development/interface/SCCS/s.common.c
* Author:
* Revision:   1.11.0
* Date:      94/06/17-16:37:36
*-----
*
* Encompasses: GetTitle, InitControl, InitEditDialog, InitItemTitle,
*              EmptyMenu, DeleteListEntries, create_client,, copy_header,
*              create_fn, loadDocTypes, loadMaintnames, read
*-----
*/

/*
* Function:   openDialog
* Description: Open the current dialog box and center it
* Parameters: dialog - ptr to dialog box
* Returns:    None
*/

/*
* Function:   GetTitle
* Description: Returns the current selection (title) of
*             the control
* Parameters: vcontrol * to the control
* Returns:    vchar * to the text of the selection
*/

/*
* Function:   InitControl
* Description: Initializes a control by setting its selection
*             range empty.
* Parameters: vcontrol * to the control
* Returns:    void
*/

```

```

/*
 * Function:   InitEditDialog
 * Description: Initializes an edit field by setting the edit
 *              field contents to " ".
 * Parameters: vdialog * to the dialog containing the edit field
 *              const vname * to the tagname of the edit field
 * Returns:    void
 */

/*
 * Function:   InitItemTitle
 * Description: Initializes a dialog item by setting the item's
 *              title to " ".
 * Parameters: vdialog * to the dialog containing the item
 *              const vname * to the tagname of the item
 * Returns:    void
 */

/*
 * Function:   EmptyMenu
 * Description: Empties a menu (for example, a menu associated with
 *              a control) by deleting items from right to left
 *              to minimize re-indexing.
 * Parameters: vmenu * to the menu to be emptied
 * Returns:    void
 */

/*
 * Function:   DeleteListEntries
 * Description: Deletes and destroys (frees) entries in a list and
 *              destroys the list when empty.
 * Parameters: vlist * to the list to be emptied & destroyed
 * Returns:    void
 */

/*
 * Function:   create_fn
 * Description: creates an 8.3 type filename from docId and Instance #
 * Parameters: buffer for results, docId, instance
 * Returns:    Nothing
 */

/*
 * Function:   loadDocTypes
 * Description: Loads valid Document Types into the passed control
 * Parameters: vcontrol * to the control that will contain the valid types
 * Returns:    int rc = 0 (success), -1 (failure)
 */

```

```
/*  
 * Function:    CheckNFSMount  
 *  
 * Description: If topic path not mounted, inform user  
 * Parameters:  None  
 * Returns:     Nothing  
 */
```

```

/*-----
* File name:      contain.c
* Date:           4/8/94
* Author:         dtl
* Description:     Functions that control containers
*
*-----
* Archive:        /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.contain.c
* Author:
* Revision:       2.0.6
* Date:          94/05/03-16:38:08
*-----
*
* Encompasses:    OptionsProc
*
*-----
*/

/*
* Function:       OptionProc
*
*
* Description:    Control which TDOC header set to display.
*                For TDI and CSP, the Mandatory set appears.
*                For QA, there is only one set.
* Parameters:     control - ptr to item (TDI, QA, CSP) selected
*                event - ptr to current event
* Returns:        Nothing
*/

```

```

/*-----
* File name:      ctest.c
* Date:           2/25/94
* Author:         dtl
* Description:    This file contains routines that build a new CDOCS record,
*                pass the record to the server for submission and display
*                the returned document assession number to the user.
*
*-----
* Archive:        /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.ctest.c
* Author:
* Revision:       2.0.6
* Date:           94/05/03-16:38:08
*-----
* Encompasses:    updateRecord, deleteRecord, buildstr, resetDocumentNo,
*                setDocumentNo, setDocumentNo, displayDocumentNo
*-----
*/

/*
* Function:       updateRecord
*
* Description:    Pass the record to the server, display the
*                returned document number, and turn the submit
*                button off.
* Parameters:     strV - Pointer to record
* Returns:        Nothing
*/

/*
* Function:       deleteRecord
*
* Description:    Delete the current record
* Parameters:     None
* Returns:        Nothing
*/

/*
* Function:       buildstr
*
* Description:    Concatenate the current field contained in str
*                to the buffer pointed to by finalstr
* Parameters:     str - Pointer to current field
* Returns:        Nothing
*/

```

```

/*
* Function:   setDocumentNo
*
* Description: Set the appropriate document type's
*              document number to the return value
*              from RPCSubmit.
* Parameters: docnum - document number
* Returns:    Nothing
*/

/*
* Function:   resetDocumentNo
*
* Description: Set the appropriate document type's
*              document number to NULL.
* Parameters: None
* Returns:    Nothing
*/

/*
* Function:   displayDocumentNo
*
* Description: Put the document number in the document
*              number text item
* Parameters: None
* Returns:    Nothing
*/

```

```

/*
 * da_clnt.c
 *
 * -----
 * Archive:    /home/samson/dlincoln/rpdocs/development/rpc/SCCS/s.da_clnt.c
 * Author:
 * Revision:   2.31.0
 * Date:       95/07/07-08:50:03
 * -----
 *
 * Implementation file for RPDOCS RPC client functions.
 *
 * Normally this file is generated from da.x with rpcgen but FTP's
 * rpcgen generates only DOS code and thus this file must be made to
 * conform to the Windows API with special attention to declaring
 * callback functions and pointer as FAR.
 *
 * Code below is written for the following:
 *
 * platform      compiler      defined by compiler
 *
 * SMI sun       cc             __sparc
 * MS windows    Watcom C/C++   __WATCOMC__
 * Microsoft     C7             __MSC__
 * IBM OS/2       IBM C/C++ Set __OS2__
 * Apple Macintosh Semantic C   __MAC__
 *
 * __MSC__ used to build the rpc16.dll only -- it and __WATCOMC__
 * are defined in makefile.win.
 *
 * 08/26/94 - cjm
 * 09/07/94 - cjm - merged with other ports
 * 09/16/94 - cjm - merged in OS/2 port
 * 10/25/94 - rlm - modified OS/2 clnt_call to use global timeout structure
 * 02/07/95 - cjm - solaris port (removed gnucc, acc, borland stuff)
 * -----
 */

```



```

/*
 * da_svc.c
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/rpc/SCCS/s.da_svc.c
 * Author:
 * Revision:   2.26.0
 * Date:       95/07/07-08:50:08
 *-----
 *
 * Implementation file for RPDOCS RPC client functions.
 *
 * Normally this file is generated from da.x with rpcgen but FTP's
 * rpcgen generates only DOS code and thus this file must be made to
 * conform to the Windows API with special attention to declaring
 * callback functions and pointer as FAR.
 *
 * Callback function pointers and functions to make and free proc
 * instances modeled after NetManage's RPC rpcgen are not generated
 * by FTP.
 *
 * 08/26/94 - cjm
 * 02/08/95 - cjm - solaris port (removed gnuc stuff)
 *-----
 */

/*
 * main
 *
 * Main function for da_svc. Initializes server and logs into database.
 */

/*
 * SigChild
 *
 * This acknowledges child exits so system does change them into zombies.
 *
 * Function call signal sets this callback.
 *
 * This is a carry over from SunOS and probably not needed for Solaris.
 */

/*
 * da_prog_N
 *
 * RPC callback dispatch function.
 *
 * Calls rpc_N.
 */

```

```

/*
 * RPCSetVersion
 *
 * Sets program, number, and display version from input number. Returns
 * 0 on success, 1 on failure.
 *
 * Called by automaticLogin, login.
 *
 * Calls nothing.
 */

```

```

/*
 * SVCSetResLim
 *
 * Sets file descriptor resource limit to maximum.
 *
 * Called by main.
 *
 * Calls nothing.
 */

```

```

/*
 * SVCCheckArgs
 *
 * Checks command line arguments. Returns on success, exits on failure.
 *
 * Called by main.
 *
 * Calls nothing.
 */

```

```

/*
 * SVCSetSvcVer
 *
 * Sets server version information. Returns on success, exits on failure.
 *
 * Called by main.
 *
 * Calls nothing.
 */

```

```

/*
 * SVCSetSvcEnv
 *
 * Sets server environment information. Returns on success, exits on failure.
 *
 * Called by main.
 *
 * Calls nothing.
 */

/*
 * SVCCheckUID
 *
 * Checks user id - must be topic. Returns on success, exits on failure.
 *
 * Called by main.
 *
 * Calls nothing.
 */

/*
 * SVCInitLog
 *
 * Initializes request and synchronization logging. Returns on success, exits
 * on failure.
 *
 * Called by main.
 *
 * Calls nothing.
 */

/*
 * SVCRegister
 *
 * Registers server with RPC daemon. Returns on success, exits on failure.
 *
 * Called by main.
 *
 * Calls nothing.
 */

```

```
/*  
 * SVCLogDBAIn  
 *  
 * Logs server in and out of Oracle. Returns on success, exits on failure.  
 *  
 * Called by main.  
 *  
 * Calls nothing.  
 */
```

```
/*  
 * SVCRunSyncIn  
 *  
 * Runs TDOCS synchronization and exits.  
 *  
 * Called by main.  
 *  
 * Calls nothing.  
 */
```

```
/*  
 * SVCInitSyncIn  
 *  
 * Initializes TDOCS synchronization.  
 *  
 * Called by main.  
 *  
 * Calls nothing.  
 */
```

```

/*
* da_xdr.c
*
*-----
* Archive:      /home/samson/dlincoln/rpdocs/development/rpc/SCCS/s.da_xdr.c
* Author:
* Revision:     2.29.0
* Date:        95/07/07-08:50:04
*-----
*
* Implementation file for RPDOCS RPC client functions.
*
* Normally this file is generated from da.x with rpcgen but FTP's
* rpcgen generates only DOS code and thus this file must be made to
* conform to the Windows API with special attention to declaring
* callback functions and pointer as FAR.
*
* Callback function pointers and functions to make and free proc
* instances modeled after NetManage's RPC rpcgen are not generated
* by FTP.
*
* Code below is written for the following:
*
* platform      compiler      defined by compiler
*
* SMI sun       cc             __sparc
* MS windows    Watcom C/C++   __WATCOMC__
* Microsoft     C7             __MSC__
* IBM OS/2      IBM C/C++ Set  __OS2__
* Apple Macintosh Semantic C   __MAC__
*
* __MSC__ used to build the rpc16.dll only -- it and __WATCOMC__
* are defined in makefile.win.
*
* 08/26/94 - cjm
* 09/07/94 - cjm - merged with other ports
* 09/16/94 - cjm - merged in OS/2 port
* 02/07/95 - cjm - solaris port (removed gnuc, acc, borland stuff)
* 06/22/95 - cjm - dynamic versioning
*-----
*/

/*
* xdr_pAsciiList
*
* Callback that translates ascii pointer (report) lists. Returns TRUE on
* success, FALSE on failure.
*/

```

```

/*
 * xdr_pAsciiList
 *
 * Callback that translates ascii (report) lists. Returns TRUE on success,
 * FALSE on failure.
 */

/*
 * xdr_pBinaryList
 *
 * Callback that translates binary (report) lists pointers. Returns TRUE on
 * success, FALSE on failure.
 */

/*
 * xdr_tBinaryList
 *
 * Callback that translates binary (report) lists. Returns TRUE on success,
 * FALSE on failure.
 */

/*
 * xdr_RPCPacket
 *
 * Callback that translates an rpc packet. Returns TRUE on success, FALSE on
 * failure.
 */

/*
 * xdr_RPCReturn
 *
 * Callback that translates an rpc return. Returns TRUE on success, FALSE on
 * failure.
 */

/*
 * xdr_makeprocs
 *
 * Makes proc instances (thunks) of each xdr callback function.
 */

/*
 * xdr_freeprocs
 *
 * Frees each xdr callback function proc instance (thunk).
 */

```

```

/*-----
* File name:      define.c
* Date:           9/29/93 1800
*                10/20/93 2000 Added check for valid charcters in LARP No
* Author:         rlm
* Description:    Define is used to define an SRA record at the
*                time that the SRA record is first conceived but
*                prior to work being checked in as that record.
*-----
*
* 03/24/94 - rlm - v1.2 modify to remove ORACLE calls and use RPC
*                server exclusively
*
*-----
* Archive:        /home/goliath/rmarsh/v1.1/development/interface/SCCS/s.define.c
* Author:
* Revision:       1.2.0
* Date:           94/04/05-08:43:24
*-----
*
* Encompasses:   loadDefine, closeDefine, define.
*
*-----
*/

/*
* Function:      loadDefine
* Description:   Load the Define dialog controls and display the Define dialog.
*
* Parameters:    None
* Returns:       Nothing
*/

/*
* Function:      closeDefine
* Description:   Close the Define dialog and perform any cleanup necessary.
*
* Parameters:
* Returns:
*/

/*
* Function:      define
* Description:   Defines the SRA record by reading the settings of
*                the Define dialog controls and loading the values into
*                ORACLE and TOPIC
* Parameters:    void (reads from dialog)
* Returns:       Success or failure
*/

```

```
/*
* Function:    valid_larp
* Description: Validates the passed LARP number
* Parameters:  LARP Number
* Returns:     TRUE=VALID, FALSE=INVALID
*/

/*
* Function:    alldigits
* Description: Checks that the passed string is all digits
* Parameters:  token - string to be checked
* Returns:     Success or failure
*/
```



```

/*-----
* File name:  edit.c
* Date:      10/11/93 0700
* Author:    rlm
* Description: Support functions for edit fields
*-----
*
* 03/24/94 - rlm - v1.2 modify to remove ORACLE calls and use RPC
*                server exclusively
*
*-----
* Archive:    /home/goliath/rmarsha/v1.1/development/interface/SCCS/s.edit.c
* Author:
* Revision:   1.2.0
* Date:      94/04/05-08:43:05
*-----
*
* Encompasses:
*
*-----
*/

/*
* Create a new empty edit record
*/

/*
* Function:    openDocument
*
* Description: open a new document and read in the specified file into it
* Parameters:  textItem - ptr to text item to be read
*              path - ptr to directory containing the document
* Returns:     Nothing
*/

```

```

/*-----
* File name:      efilter.c
* Date:           10/07/93 930
* Author:         alj
* Description:    efilter.c is used to set up subclassing of
*                text items used in passing the password in
*                the Logon dialog. This text filter is also
*                used in passing a password in the Change
*                Password dialog.
*-----
* Archive:        /home/goliath/rmarsha/v1.1/development/interface/SCCS/s.efilter.c
* Author:
* Revision:       1.2.0
* Date:           94/03/28-11:01:16
*-----
*
* encompasses:    _textfilterInit(), _textfilterInsert(),
*                textfilterGetClass(), _textfilteritemCreateTextData(),
*                _textfilteritemLoadInit(), textfilteritemGetClass(),
*                textfilteritemSetItemFilter(), SetupLogonField(),
*                loginfilter().
*-----
*/

/*
* Function definitions
*/

/* _textfilterInit -- Initializes a text filter object.
*
* -> text -- the text filter to initialize.
*/

/* _textfilterInsert -- override of vtextINSERT. This passes the text
*
*                through the filter before giving it to the
*                Text Manager.
*
* -> text -- the text manager object who is getting text added.
* -> selection -- current selected range. This range will be replaced
*                by the new text.
* -> s -- the string being added.
* -> len -- how long s is .
*/

/* textfilterGetClass-- returns the vtext subclass that does filtering.
*
* <- -- the filtering subclass.
*/

```

```

/* _textfilteritemCreateTextData --
*
*          creates a new filtered vtext for use in the given textitem.
*
* -> textitem    -- a filtered text item. Parameter not used.
* <-            -- a newly created filtered vtext.
*/

/* _textfilteritemLoadInit --
*
*          reads a filtered text item from a resource.
*
* -> textitem    -- the filtered textitem to read from a resource.
* -> resource    -- the resource to read it from.
*
*/

/* textfilteritemGetClass --
*
*          returns a vtextitemClass for a filteredTextitem.
*          filteredTextitems have a filtered vtext attached to
*          them. All of the magic is in the vtext -- no real
*          changes have been made to the vtextitem.
*
* <-            -- the filteredTextitem class.
*/

/* textfilteritemSetItemFilter --
*
*          attaches the given filter to the
*          given filtered textitem.
*
* -> textitem    -- the textitem whose text is being filtered
* -> filter      -- the filter function doing the filtering.
*/

/* loginfilter      -- sample filter for limiting a text field to
*
*          alphabetic characters.
*
* -> text        -- the vtext being filtered.
* -> sel         -- the current selection.
* <-> string     -- the new text being added.
* <-> len        -- the length of the new text.
* <-            -- TRUE if the text should be added, FALSE To cancel.
*/

```

```

/*-----
* File name:      etcButtn.c
* Date:           2/16/94
* Author:         dtl
* Description:    This file contains the functions that control the ETC (...)
*                buttons in the SUBMIT dialog box containers.
*-----
* Archive:        /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.etcButtn.c
* Author:
* Revision:       2.0.6
* Date:           94/05/03-16:38:08
*-----
*
* Encompasses:   EditProc, SetEtcButton
*
*-----
*/

/*
* Function:       EditProc
*
* Description:    Perform one of the following functions from the
*                CDOCS edit dialog box:
*
*                (1)   Cancel all current modifications
*                (2)   Save all modifications and exit the dialog box
*                (3)   Apply modifications to the textitem and remain
*                     in the dialog box
*
* Parameters:    button - ptr to EDIT OK, APPLY or CANCEL button
*                event  - ptr to current event
* Returns:       Nothing
*/

/*
* Function:       SetEtcButton
*
* Description:    Determine whether the "..." string appears at
*                the end of the current CDOCS textitem
* Parameters:    text - ptr to current text string
* Returns:       ptr to text string pointed to by textitemName
*/

```

```
/*  
* Function:   SetEtcButton  
*  
* Description: Determine whether the "..." string appears at  
*              the end of the current CDOCS textitem  
* Parameters: text - ptr to current text string  
* Returns:    ptr to text string pointed to by textitemName  
*/
```

```

/*-----
* File name:      file.c
* Date:           11/10/93 1000
* Author:         rlm
* Description:    File contains routines to copy an SRA record to
*                the client and invoke WordPerfect on the client
*                with an SRA record.
*
* 11/18/93 - rlm - added conditional compilation directives for __TEST1__
*                and __PROD1__
*
* Encompasses:   loadFileCopy,closeFileCopy,fileCopy,
*                applyCopyFileName,filecopyfilechooser,Write,
*                loadFileWP,closeFileWP,fileWP
*-----
*
* 04/12/94 - rlm - modify to remove ORACLE calls and use RPC
*                server exclusively
*-----
*/

```

```

/*
* Function:      loadFileCopy
* Description:   Display the File Copy dialog
*
* Parameters:    None
* Returns:       Nothing
*/

```

```

/*
* Function:      closeFileCopy
* Description:   Close the FileCopy dialog and perform any cleanup necessary.
*
* Parameters:    None
* Returns:       Nothing
*/

```

```

/*
* Function:      filecopy
* Description:   Copies a record to the user specified file.
* Parameters:    void (reads from dialog)
* Returns:       0 = OK, -1 = Error
*/

```

```

/*
* Function:      loadFileWP
* Description:   Display the File WordPerfect dialog
*

```

* Parameters: None
* Returns: Nothing
*/

/*
* Function: closeFileWP
* Description: Close the File WordPerfect dialog and perform any
* cleanup necessary.
*
* Parameters: None
* Returns: Nothing
*/

/*
* Function: filewp
* Description: Invokes WP against a requested record.
* Parameters: void (reads from dialog)
* Returns: 0 = OK, -1 = Error
*/

```

/*-----
* File name:      filechsr.c
* Date:           2/15/94
* Author:         dtl
* Description:     This file contains the functions that open a File Chooser
*                  when the user selects the Browse button.
*-----
* Archive:        /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.filechsr.c
* Author:
* Revision:       2.0.6
* Date:           94/05/03-16:38:08
*-----
*
* Encompasses:    ApplyFileNotify, FileChooserBrowse
*
*-----
*/

/*
* Function:       FileChooserBrowse
*
* Description:    Notification procedure for the Open menu item
*                  in the File menu
*
* Parameters:    button - ptr to file chooser button: APPLY, CANCEL, OK
* Returns:       Nothing
*
*/

/*
* Function:       ApplyFileNotify
*
* Description:    Notification procedure for filechooser for Apply/OK
*                  Checks the path returned and if its a file, then
*                  do something with it and let the file chooser close down.
*                  Else, do nothing and let filechooser keep going.
*
* Parameters:    fileChooser - ptr to fileChooser structure
*                  path - directory path of file
* Returns:       TRUE File chosen was ok - go ahead and let the chooser go away
*                  if OK was selected.
*                  FALSE Don't close chooser - file wasn't acceptable
*/

```



```

/*-----
* File name:  help.c
* Date:      10/13/93 930
* Author:    alj
* Description: help.c is used to set up Main Menu Help by
*             providing help dependent upon the user's
*             authority level. Checks the Main Menu bar
*             for what menu selections are found and sets
*             up the help accordingly.
* Modified 5-23-94  alj - modified intern.vr file to represent
*                   current help text. Modified code for appropriate
*                   display of help text.
* Modified 5-26-94  alj - modified to add help dialogs for
*                   Custodian functions.
* Modified 5-31-94  alj - modified window titlebar for global
*                   variable.
* Modified 6-01-94  alj - modified to add help for Reports.
* Modified 7-26-94  alj - modified to add help for TDOCS.
* Modified 8-02-94  alj - modified to add help index for TDOCS.
* Modified 7-10-95  dtl - revamped entire help system
*-----
* Archive:    /home/goliath/rmarsh/v1.1/development/interface/SCCS/s.help.c
* Author:
* Revision:   1.1.0
* Date:      94/03/10-15:03:40
*-----
*/

```

```

/*
* Function:    LoadHelp
*
* Description: Load the Help and Help Index dialog boxes
* Parameters:  None
* Returns:     Nothing
*/

```

```

/*
* Function:    BuildHelpCursor
*
* Description: Load the Help based upon the user's privilege level
* Parameters:  None
* Returns:     Nothing
*/

```

```

/*
* Function:   HelpIndexProc
*
* Description: Close the help dialog box
* Parameters: button - ptr to button selected: OK or CANCEL
*              event - ptr to current event
* Returns:    Nothing
*/

```

```

/*
* Function:   Help
*
* Description: Display the Help text for the given function
* Parameters: button - not used; for declaration only
*              ev - not used; for declaration only
* Returns:    Nothing
*/

```

```

/*
* Function:   ShowHelp
*
* Description: Open the help dialog box
* Parameters: notInitialLogon - if initial logon help selected,
*              don't try to access the Oracle database
*              key - ptr to selected help text
*              subkey - not used
* Returns:    Nothing
*/

```

```

/*
* Function:   HelpProc
*
* Description: Close the help dialog box
* Parameters: button - ptr to button selected
*              event - ptr to current event
* Returns:    Nothing
*/

```

```

/*-----
 * init.c
 *
 * 09/20/93 - rlm - created.
 *-----
 * Archive:      /home/goliath/rmarsha/v1.1/development/interface/SCCS/s.init.c
 * Author:
 * Revision:     1.1.0
 * Date:         94/03/10-15:03:40
 *-----
 *
 * encompasses:  init().
 *-----
 */

/*
 * Func Name:  init
 * Purpose:    initializes the program and loads resources.
 * Inputs:
 * Outputs:
 */

```

```

/*-----
* File name:      label.c
* Date:           5/12/94
* Author:         dtl
* Description:    This file contains the functions that creates labels,
*                new document reports, batch reports and circulation
*                reports.
*-----
* Archive:        /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.label.c
* Author:
* Revision:       2.0.6
* Date:           94/05/03-16:38:08
*-----
*
* Encompasses:   SaveAsProc, GetReportFileName, GetReportPathName,
*                GetLabel, LoadSaveAs
*-----
*/

/*
* Function:       GetReportFileName
*
* Description:    Open the SaveAs dialog box
* Parameters:     reportNumber - which report to fetch
* Returns:        Nothing
*/

/*
* Function:       Label
*
* Description:    Retrieves the labels and displays them
* Parameters:     None
* Returns:        Nothing
*/

/*
* Function:       GetReportPathName
*
* Description:    Get the current filePath name
* Parameters:     which - which report to fetch
*                0 - Labels
*                1 - Not Labels
* Returns:        Nothing
*/

```

```
/*  
 * Function:   LoadSaveAs  
 *  
 * Description: Load the checkout dialog box  
 * Parameters: None  
 * Returns:    Nothing  
 */
```

```
/*  
 * Function:   SaveAsProc  
 *  
 * Description: get the file name to save as  
 * Parameters: button - ptr to button chosen: OK, or CANCEL  
 *             event - ptr to current event  
 * Returns:    Nothing  
 */
```

```

/*-----
* File name:  listview.c
* Date:      10/8/93 2050
* Author:    rlm
* Description: Listview implements listview functions to
*             support Checkin.
*-----
*
* 04/06/94 - rlm - v1.2 modify to remove ORACLE calls and use RPC
*             server exclusively
*
* 03/14/95 - sv - Added a new function loadSubsetDocs to get titles
*             with all the sections to generate report
*
*-----
* Archive:    /user1/goliath/dlincoln/rpdocs/development/interface/SCCS/s.listview.c
* Author:
* Revision:   1.22.0
* Date:      94/06/28-15:44:20
*-----
* Encompasses:
*-----
*/

```

```

/*
* Function:   loadDocs
*
* Description: Load a document into a listview
*
* Parameters: req - document request number
*             type -
*             listview - ptr to listview
*             list - array of list ptr's
* Returns:    Success or failure: 0 or -1
*/

```

```

/*
* Function:   loadSubsetDocs
*
* Description: Load the Force Checkout dialog box and establish
*             call backs for the OK and Cancel buttons
* Parameters: titleStr- document name
*             type -
*             listview - ptr to listview
*             list - array of list ptr's
* Returns:    Success or failure: 0 or -1
*/

```

```

/*-----
* logon.c
*
* 09/20/93 - rlm - created.
* 10/06/93 - rlm - modified to add new password, then changed back
*               to original condition.
* 10/20/93 - rlm - always set notification text on logon error.
* 11/09/93 - rlm - change user to operator and add user w/o topic.
*-----
*
* 03/24/94 - rlm - v1.2 modify to remove ORACLE calls and use RPC
*               server exclusively
* 05/13/94 - alj - modified to validate the password length field.
*               Minimum of 4 and maximum of 16.
* 07/14/94 - rlm - added privilege levels for TDOCS
*-----
* Archive:      /home/goliath/rmarsha/v1.1/development/interface/SCCS/s.logon.c
* Author:
* Revision:     1.4.0
* Date:         94/03/30-17:09:42
*-----
* 12/20/94 dgm add setstatusVstr
*-----
*
* encompasses:  logon(), login(), logonRetryYes(), setupMenu(),
*               destroyMenu()
*-----
*/

/*
* Function:     logon
* Description:  Display the Logon Dialog.
*
* Parameters:   None
* Returns:      Nothing
*/

/*
* Function:     RPCSetVersion
* Description:  Sets program, number, and display version from input number.
*
* Parameters:   inp   - input number
*               ver   - rpc version
* Returns:      0 on success, 1 on failure.
*/

```

```

/*
* Function:    basicLogin
* Description: This function is invoked by the main.c file. This function
*              is invoked when the user provides "-i" option at the
*              command line. This function ignores the database connection
*              and brings the main interface with limited menu options.
*
* Parameters:  None
* Returns:     Nothing
*/

/*
* Function:    automaticLogin
* Description: Connect to server with no username and no password.
*              If successful, add menu selections based on privilege level.
*              This function is called by the main.c program if the user
*              provides a "-u" option without an "-i" option at the
*              command line.
*
* Parameters:  None
* Returns:     Nothing
*/

/*
* Function:    getServerNameAndVersion
* Description: Retrieve the server's name and version in which to connect
*              to, depending upon the value of which_server. These values
*              are read from the preference file.
*
* Parameters:  server - server name
*              versionNumber - server version
*              app - TDOCS or RPD
* Returns:     Success or failure
*/

/*
* Function:    StripWhiteSpace
* Description: Remove leading and trailing blanks from a string
*
* Parameters:  string - string to parse
* Returns:     Nothing
*/

/*
* Function:    login
* Description: Connect to server. If successful, add menu selections
*              based on privilege level.
*

```



```

* Parameters: None
* Returns:    Nothing
*/

/*
* Function:    determineScanPrivilege
* Description: Determine whether the user has the correct
*              privileges to see the SCAN selection
*
* Parameters:  None
* Returns:     Nothing
*/

/*
* Function:    scanMachine
* Description: If the Calera software is loaded on the
*              current machine, return TRUE
*
* Parameters:  None
* Returns:     True of False
*/

/*
* Function:    disableReportsMenu
* Description: Disable the reports menu
*
* Parameters:  None
* Returns:     Nothing
*/

/*
* Function:    logonRetryYes
* Description: Notify hook function. Sole purpose in life is to
*              dismiss Notification so that user can try again to login.
*
* Parameters:  notice - ptr to Notice item
*              event - ptr to current event
* Returns:     True of False
*/

/*
* Function:    setstatus
* Description: Sets the status line to the passed message
*              in the color specified
* Parameters:  msg, color
* Returns:     Nothing
*/

```

```

/*
 * Function:   setstatusScribed
 * Description: invokes setstatus using scribe
 *
 * Parameters: s, color
 * Returns:   Nothing
 */

/*
 * Function:   setstatusStr
 * Description: invokes setstatus with C string
 *
 * Parameters: msg, color
 * Returns:   Nothing
 */

/*
 * Function:   deleteMenuSeparator
 * Description: After the menu items have been deleted,
 *              remove menu separators at the top or bottom
 *              of the menu, or a separator that immediately
 *              follows another.
 *
 * Parameters: None
 * Returns:   Nothing
 */

/*
 * Function:   deleteSubMenuItems
 * Description: Remove submenus from a menu item that is
 *              to be deleted
 *
 * Parameters: item - menu item ptr
 * Returns:   Nothing
 */

/*
 * Function:   deleteMenuItems
 * Description: Remove menus from the main menu
 *
 * Parameters: menu - menu item ptr
 *              k - menu item index
 * Returns:   Nothing
 */

```

```

/*
* Function:    determineSetupMenu
* Description: Remove menus from the main menu
*              depending on the user's privilege
*              level
*
* Parameters:  None
* Returns:     Nothing
*/

/*
* Function:    DeleteSearchItems
* Description: Remove search items from the Search
*              dialog box depending on the user's privilege
*              level
*
* Parameters:  None
* Returns:     Nothing
*/

```

```

/*-----
* File name:      main.c
* Date:           9/26/93 1800
* Author:         rlm
* Description:    Main contains the main routine and the event-loop
*                switch statement for calling routines to do the real work.
*-----
*
* 03/25/94 - rlm - v1.2 modify to remove ORACLE calls and use RPC
*            server exclusively
*
* 04/25/94 - alj - modified to add events for new help dialog buttons
*            for help operations and searches.
* 05/31/94 - alj - modified to add help dialogs for custodian help.
* 07/26/94 - alj - modified to add help dialogs for TDOCS.
*-----
* Archive:       /home/goliath/rmarsha/v1.1/development/interface/SCCS/s.main.c
* Author:
* Revision:      1.5.0
* Date:          94/05/02-14:45:29
*-----
*
*
* encompasses:   main(), issueCmds()
*-----
*/

/*
*Func Name:  main
*Purpose:    The mainline procedure call the initialization proc,
*            provides the event loop, and calls the cleanup routine.
*
*Inputs:
*Outputs:
*/

```

```

/*-----
* File name:      maint.c
* Date:           10/18/93 1045
* Author:         rlm
* Description:    Maint performs maintenance functions.
*                The only such function at this time is
*                retirement.
*-----
*
* 04/12/94 - rlm - modify to remove ORACLE calls and use RPC
*
*-----
* Archive:        /home/goliath/rmarsha/v1.1/development/interface/SCCS/s.maint.c
* Author:
* Revision:       1.1.0
* Date:           94/03/10-15:03:43
*-----
*
* Encompasses:    loadRetire,closeRetire,Retire.
*/

/*
* Function:       loadRetirement
*
* Description:    Display the retirement dialog box
*
* Parameters:     None
* Returns:        Nothing
*/

/*
* Function:       closeRetirement
* Description:    Close the Retirement dialog and perform any cleanup necessary.
*
* Parameters:     None
* Returns:        Nothing
*/

/*
* Function:       retire
* Description:    Retires the SRA record.
*                header & file submitted
*                to server for TOPIC and WP Repository
*
* Parameters:     None
* Returns:        0 = OK, -1 = Error
*/

```

```
/*  
 * Function:  reallyRetire  
 * Description: Retire the document after verification  
 *  
 * Parameters: None  
 * Returns:   Nothing  
 */
```

```
/*  
 * Function:  dontretire  
 * Description: Do not retire the document after verification  
 *  
 * Parameters: None  
 * Returns:   Nothing  
 */
```

```

/*-----
* memory.c
*
* 7/2/93 - cjm - coded.
*-----
* $Archive$
* $Author$
* $Revision$
* $Date$
*-----
*
* The purpose of this module to handle different memory models.
* Specifically, the following are handled:
*
*      ANSI C      - malloc and free
*      GALAXY API - vmemAlloc and vmemFree
*
* Those functions are redefined as
*
*      void* MemAlloc(size_t size);
*      void MemFree();
*
* In stddev.h, set _ANSI_C to 1 and _GALAXY to 0 to compile for
* ANSI C, and vice versa for GALAXY.
*-----
*/

/*
* MemAlloc
*
* Allocates memory block of size size.
*
* Returns a void pointer to that memory block, so coerce it;
* or NULL (assume out of memory).
*
* size - size of memory block to be allocated.
*/

/*
* MemReAlloc
*
* Allocates memory block of size size.
*
* Returns a void pointer to that memory block, so coerce it;
* or NULL (assume out of memory).
*
* size - size of memory block to be allocated.
*/

```

```
/*  
 * MemFree  
 *  
 * Frees allocated memory.  
 */
```



```

/*-----
* File name:      options.c
* Date:           10/8/93 1800
* Author:         alj
* Description:    options.c is used to define additional options
*                to be performed throughout the use of PADB,
*                i.e. changing passwords.
*-----
*
* 04/12/94 - rlm - modify to remove ORACLE calls and use RPC
*
*-----
* Archive:        /home/goliath/rmarsha/v1.1/development/interface/SCCS/s.options.c
* Author:
* Revision:       1.2.0
* Date:           94/04/06-12:22:25
*-----
*
* Encompasses:    loadchangePWdialog(), changePassword(),
*                changePWRetryYes(), SetupChangePWField()
*-----
*/

/*
* Function:       loadchangePWdialog
*
* Description:    Load and Open the Password Change Dialog
*
* Parameters:     None
* Returns:        Success or failure
*/

/*
* Function:       changePassword
*
* Description:    Change Password, passing password through text filter to mask it
*
* Parameters:     None
* Returns:        Success or failure
*/

/*
* Function:       changePWRetryYes
*
* Description:    Sets the focus to the first password
*                field after confirmation of notice
*

```

```

* Parameters: notice - ptr to notice structure
*             event - ptr to current event
* Returns:    True
*/

```

```

/*
* Function:    SetupChangePWField
*
* Description: Sets up the text filters to use when changing passwords
*
* Parameters:  None
* Returns:     Nothing
*/

```

```

/*
* Function:    loadUserIDs
*
* Description: Load user ID's into dialog box
*
* Parameters:  None
* Returns:     True
*/

```

```

/*
* Function:    addUserIDs
*
* Description: Add a new user to the Oracle database
*
* Parameters:  None
* Returns:     True
*/

```

```

/*
* Function:    deleteUserIDs
*
* Description: Remove a user from the Oracle database
*
* Parameters:  None
* Returns:     True
*/

```

```

/*
* Function:    changeUserIDs
*
* Description: Change a user's info in the Oracle database
*
* Parameters:  None
* Returns:     True
*/

```

```

/*
* Function:    selectIDChange
*
* Description: Retrieve the user's info from the list view
*
* Parameters: useridlistview - ptr to list view
*              state - state of the list view
* Returns:     Nothing
*/

/*
* Function:    loadPrivilegeLevel
* Description: Loads valid Privilege Levels
* Parameters: vcontrol * to the control that will contain the valid types
* Returns:     int rc = 0 (success), -1 (failure)
*/

```

```

/*-----
* File name:      prefs.c
* Date:           04/03/94 1550
* Author:         rlm
* Description:    Prefs contains routines to display and edit
*                user preferences.
*-----
* Archive:        /home/goliath/rmarsha/v1.1/development/interface/SCCS/s.prefs.c
* Author:
* Revision:       1.2.0
* Date:           94/05/03-16:38:26
*-----
*
* Encompasses:   loadPrefsdialog()
*
*-----
*/

/*
* Function:      loadPrefsDialog
*
* Description:   Load all user preferences
*
* Parameters:    None
* Returns:       Nothing
*/

/*
* Function:      savePrefs
*
* Description:   Save user preferences
*
* Parameters:    None
* Returns:       Nothing
*/

```

```

/*-----
* File name:      printer.c
* Date:           7/10/95
* Author:         dtl
* Description:    Galaxy print driver
*                (For Macintosh only)
*-----
* Archive:        /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.printer.c
* Author:
* Revision:       2.0.6
* Date:           94/05/03-16:38:08
*-----
*
* Encompasses:   makePrinter, nukePrinter, saveThenNukePrinter,
*                setPrinterFont, sendThenNukePrinter
*-----
*/

/*
* Function:      setPrinterFont
* Description:   Set the printer's font
*
* Parameters:    None
* Returns:       vfont *
*/

/*
* Function:      doPrint
* Description:   Print the document
*
* Parameters:    printer - ptr to printer structure
* Returns:       Nothing
*/

/*****
* Handling the printer object
*/

/*
* Function:      sendThenNukePrinter
* Description:   Print the document then destroy the
*                printer object
*
* Parameters:    printer - ptr to printer structure
*                good - valid printer?
* Returns:       Nothing
*/

```

```

/*
 * Function:    saveThenNukePrinter
 * Description: Destroy the printer object
 *
 * Parameters:  printer - ptr to printer structure
 *              good - not used
 * Returns:     Nothing
 */

/*
 * Function:    makePrinter
 *
 * Description:  Create the printer object
 *
 * Parameters:  None
 * Returns:     vprint *
 */

/*
 * Function:    nukePrinter
 * Description:  Destroy the printer object
 *
 * Parameters:  printer - ptr to printer structure
 * Returns:     Nothing
 */

/*****
 * Handle doing print job confirmation
 */
/*
 * Function:    doPrintJob
 *
 * Description:  Create printer and print
 *
 * Parameters:  None
 * Returns:     True
 */

```

```

/*-----
* File name:  report.c
* Date:      10/11/93 0600
* Author:    rlm
* Description: Report calls the report functions and displays
*             the report in a dialog. The user can view the
*             report and elect to print it.
*-----
*
* 03/24/94 - rlm - v1.2 modify to remove ORACLE calls and use RPC
*             server exclusively
*
* 06/15/94 - dtl - Added code for RPD database reports
*
* 07/05/94 - alj - modified to close database reports help dialog
*
* 03/13/95 - sv - Added code to bring up the subset report dialog box
*
*-----
* Archive:    /home/goliath/rmarsha/tdocs/development/interface/SCCS/s.report.c
* Author:
* Revision:   1.5.0
* Date:      94/05/03-16:38:27
*-----
*
* Encompasses:
*
*-----
*/

/*
* Function:   MenuReportWriter
*
* Description: Open the MenuReportWriter dialog box
* Parameters: None
* Returns:    Nothing
*/

/*
* Function:   LoadMenuReportWriter
*
* Description: Load the MenuReportWriter dialog box
* Parameters: None
* Returns:    Nothing
*/

```

```
/*
* Function:   MenuReportWriterProc
*
* Description: get the file name to save as
* Parameters: button - ptr to button chosen: OK, or CANCEL
*             event - ptr to current event
* Returns:   Nothing
*/
```

```
/*
* Function:   RefReport
*
* Description: Open the RefReport dialog box
* Parameters: None
* Returns:   Nothing
*/
```

```
/*
* Function:   LoadRefReport
*
* Description: Load the RefReport dialog box
* Parameters: None
* Returns:   Nothing
*/
```

```
/*
* Function:   RefReportProc
*
* Description: get the file name to save as
* Parameters: button - ptr to button chosen: OK, or CANCEL
*             event - ptr to current event
* Returns:   Nothing
*/
```



```

/*
 * rpc16.c
 *
 *-----
 * Archive:    %P%
 * Author:
 * Revision:   %R%.%L%.%B%
 * Date:       %E%-%U%
 *-----
 *
 * Implementation of rpc16.dll for windows.
 *
 * Ifdefs are used for FTP's SDK (__FTP__) and NetManage's
 * SDK (__NM__). These are defined in makefile.win where make
 * instructions can be found.
 *
 * 08/09/94 - cjm
 * 09/08/94 - cjm - merged FTP and NM versions
 * 06/22/95 - cjm - dynamic versioning
 *-----
 */

/*
 * WEP
 *
 * Called when the DLL is unloaded. Exits FTP's RPC for Windows.
 * Frees procedure instances of xdr function callbacks - THIS SHOULD
 * BE **ONLY** ONCE - see above.
 *
 * Called by window manager.
 *
 * Calls RPCExit, xdr_freeproc.
 */

/*
 * __RPCConnect
 *
 * Saves db access string and calls clnt_create to obtain client connection and
 * clnt_control to set timeout. Returns 1 on success or 0 on failure.
 *
 * Called by _RPCConnect.
 *
 * Calls RPCInit.
 */

/*
 * __RPCRequest
 *

```

```

* Sets rpc packet and calls rpc_N (where N=version).
*
* Called by _RPCRequest.
*
* Calls __RPCReadBinaryFile, rpc_N, __RPCFreeBinary,
* __RPCWriteBinaryFile, __RPCWriteAsciiFile.
*/

/*
* __RPCResults
*
* Sets results status, error, and data length.
*
* Called by _RPCRequest.
*
* Calls nothing.
*/

/*
* __RPCData
*
* Copies data to memory pointed to by data, and frees local data.
*
* Called by _RPCRequest.
*
* Calls nothing.
*/

/*
* __RPCWriteAsciiFile
*
* Writes ascii list to file named by filename. Sets results status.
* Data is freed in __RPCData. Returns 0 on success or 1 on failure.
*
* Called by __RPCRequest.
*
* Calls nothing.
*/

/*
* __RPCWriteBinaryFile
*
* Writes binary list to file named by filename. Sets results status.
* Data is freed in __RPCData. Returns 0 on success or 1 on failure.
*
* Called by __RPCRequest.
*
* Calls nothing.
*/

```

```

/*
 * __RPCReadBinaryFile
 *
 * Reads binary file to packet binary. Sets results status.
 * Data is freed in __RPCFreeBinary. Returns 0 on success or 1 on failure.
 *
 * Called by __RPCRequest.
 *
 * Calls __RPCFreeBinary.
 */

```

```

/*
 * __RPCFreeBinary
 *
 * Frees packet binary. Emulates
 *
 * Could perhaps call:
 *
 * xdr_free(xdr_pBinaryList_p,
 * (char FAR*)(pBinaryList FAR *)&pPacket->binary);
 *
 * But how would it know about handles?
 *
 * Called by __RPCRequest, __RPCReadBinary.
 *
 * Calls nothing.
 */

```

```

/*-----
* File name:      scan.c
* Date:           11/8/94
* Author:         dtl
* Description:    Scan contains the routines to perform
*                on-demand and routine scanning.
*-----
* Archive:        /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.scan.c
* Author:
* Revision:       2.0.6
* Date:           94/05/03-16:38:08
*-----
*
* Encompasses:   rawCopyFile, CheckAddPathToComboBox, ExistsPath,
*                MakeSubmitPath, RemoveSubmitFiles, onDemandScan,
*                GetSubmitInfo, GetHostName, DelSubmitFiles,
*                GetElecFiles, GetSubmitPath, GetSource
*-----
*/

```

```

/*
* Function:       onDemandScan
* Description:    Execute the M-PRO series scanning software
*                for on-demand scanning. This function can
*                only be run on the PC machine that has the
*                Calera M-PRO Series software available.
*
* Parameters:     None
* Returns:        Nothing
*/

```

```

/*
* Function:       GetSubmitInfo
* Description:    If the user has the privileges to import
*                documents for submission into CDOCS, determine
*                the source of the document (e.g. .doc for WP
*                files, .txt for ASCII files) and the name of
*                the host machine.
*
* Parameters:     source, host, which
* Returns:        Nothing
*/

```

```

/*
* Function:       GetHostName
* Description:    Get the name of the host machine for
*                importing a document into CDOCS

```

```

*           for document submission.
*
* Parameters: Host
* Returns:   Nothing
*/

/*
* Function:   GetSource
* Description: Get the directory path if there are files to copy
*
* Parameters: None
* Returns:    0 -> Path = None; No files to copy
*             1 -> Success
*             -1 -> Error
*/

/*
* Function:   GetElecFiles
* Description: Copy files from user source directory to
*             /{topicdpath}/{which_system}/submit/<host>{<pid>}
*
* Parameters: None
* Returns:    0 -> No path given
*             1 -> Success
*             2 -> Error
*/

/*
* Function:   rawCopyFile
* Description: Perform a binary file copy
*
* Parameters: oldFilePath, newFilePath
* Returns:    Success or failure
*/

/*
* Function:   DelSubmitFiles
* Description: Delete all files in the submit directory
*
* Parameters: None
* Returns:    Nothing
*/

/*
* Function:   GetSubmitPath
* Description: Return the path where the files are to be copied
*
* Parameters: None

```

* Returns: file path

*/

/*

* Function: MakeSubmitPath

* Description: Create the path where the files are to be copied

*

* Parameters: path

* Returns: Success or failure

*/

/*

* Function: RemoveSubmitFiles

* Description: Removes the files from the submit directory

*

* Parameters: path

* Returns: Success or failure

*/

/*

* Function: CheckAddPathToComboBox

* Description: Add path name to combo box for future reference

*

* Parameters: pathStr, combo

* Returns: Nothing

*/

/*

* Function: ExistsPath

*

* Description: Determine existence of file path

*

* Parameters: path

* Returns: Success or failure

*/

```

/*-----
* search.c
*
* 09/28/93 - rlm - created.
* 10/14/93 - rlm - modified to support TOPIC privilege levels
* 10/20/93 - cjm - modified users path from guest to users
* 11/12/93 - rlm - added new operator privilege
* 11/18/93 - rlm - added conditional compilation directives for __TEST1__
*              and __PROD1__
* 12/07/93 - cjm - removed absolute paths to script file
*-----
*
* 04/11/94 - rlm - v1.2 modify to remove ORACLE calls and use RPC
*              server exclusively
*
*-----
* Archive:      /home/goliath/rmarsha/tdocs/development/interface/SCCS/s.search.c
* Author:
* Revision:     1.2.0
* Date:         94/04/14-09:01:48
*-----
*
* encompasses: starttopic.
*-----
*/

/*
* Function:     starttopic
* Description:  Launch Topic
*
* Parameters:   None
* Returns:      True
*/
/*****CONDITIONAL STARTTOPIC*****/

/*
* Function:     BuildLaunchCmd()
* Description:  Construct the docdb.lnc and hitlist.lnc files
*              which are used by TOPIC to launch WordPerfect,
*              an image viewer, and to save the hitlist to a
*              file.
*
* Parameters:   topiclist - List of TOPIC partitions
* Returns:      Nothing
*/

```

```

/*
* Function:    SetTopicPrivilege()
* Description: Set the user's privilege level for TOPIC
*
* Parameters:  None
* Returns:     Nothing
*/

/*
* Function:    LoadSearchDialog
* Description: Load the Search dialog box and establish
*             call backs for all buttons, checkboxes and the
*             query listview
*
* Parameters:  None
* Returns:     Nothing
*/

/*
* Function:    SetSearchAllOf
* Description: If the user selects a check box other than ALL,
*             set the ALL checkbox off.
*
* Parameters:  control - check box clicked
*             event - current event
* Returns:     Nothing
*/

/*
* Function:    ResetSearchToAll
* Description: If the user selects the ALL check box,
*             set all other checkboxes off.
*
* Parameters:  control - All checkboxes clicked
*             event - current event
* Returns:     Nothing
*/

/*
* Function:    GetSearchSources
* Description: If the user selects the ALL check box,
*             set all other checkboxes off.
*
* Parameters:  control - ALL checkboxclicked
*             event - current event
* Returns:     Nothing
*/

```



```

/*
* Function:   SearchListProc
* Description: Allow the user to check a document in
*             or cancel the function
* Parameters: button - ptr to button chosen: OK or CANCEL
*             event - ptr to current event
*
* Returns:    Nothing
*/

```

```

/*
* Function:   SearchProc
*
* Description: Launch TOPIC or cancel search
* Parameters: button - ptr to button chosen: OK or CANCEL
*             event - ptr to current event
* Returns:    Nothing
*/

```

```

/*
* Function:   WritePrefFile
* Description: Create the TOPIC preference file
* Parameters: searchList - The list of TOPIC partitions
*             chosen by the user
*
* Returns:    Nothing
*/

```

```

/*
* Function:   SaveSearchQuery
* Description: Retrieve the user supplied name for the
*             preference file and call WritePrefFile
*
* Parameters: searchList - The list of TOPIC partitions
*             chosen by the user
* Returns:    Nothing
*/

```

```

/*
* Function:   BuildSearchQueries
* Description: Load all user preference file names into
*             the search dialog box
*
* Parameters: None
* Returns:    Nothing
*/

```

```

/*
* Function:    closeSearch
* Description: Close the search dialog box
*
* Parameters:  None
* Returns:     Nothing
*/

/*
* Function:    Search
* Description: Begin TOPIC search
*
* Parameters:  None
* Returns:     Nothing
*/

/*
* Function:    BuildPrefFileName
* Description: Get the TOPIC executable name from
*              the user's preference dialog box
*
* Parameters:  fileroot - path and exe name
* Returns:     Nothing
*/

/*
* Function:    SearchSelectProc
* Description: Call back procedure to handle all
*              search dialog objects
*
* Parameters:  None
* Returns:     search list of partitions
*/

/*
* Function:    SetTopicSources
* Description: Construct all user selected sources
*
* Parameters:  topiclist - list of user choices
* Returns:     Nothing
*/

```

```

/*-----
* File name:  submit.c
* Date:      2/15/94
* Author:    dtl
* Description: This file contains routines that are called when SUBMIT
*             is selected from the Document Menu or the Submit button.
*-----
* Archive:   /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.submit.c
* Author:
* Revision:  2.0.6
* Date:      94/05/03-16:38:08
*-----
*
* Encompasses:
*
*-----
*/

/*
* Function:   buttonSubmit
*
* Description: Submit the current TDI record from SUBMIT button
* Parameters: submitItem - ptr to "SUBMIT" item selection
*             event - ptr to current event
* Returns:    Nothing
*/

/*
* Function:   writeTDOCSinfo
*
* Description: Write the TDOCS record
* Parameters: Title - Name of operation:
*             Submit or Update
* Returns:    Nothing
*/

/*
* Function:   submitRecord
*
* Description: Pass the record to the server, display the
*             returned document number, and turn the submit
*             button off.
* Parameters: str - Pointer to record
* Returns:    Nothing
*/

```

```

/*
* Function:    getTDOCSinfo
*
* Description: Get the TDOCS record according to the document
*              number
* Parameters:  None
* Returns:    Success or Failure
*/

/*
* Function:    LoadSubmit
* Description: Display the TDOCS submit dialog
*
* Parameters:  func - delete, update, submit
* Returns:    Nothing
*/

/*
* Function:    BuildDocSet
* Description: This function build the doc set.
*
* Parameters:  dialog name, flag
*              flag = 0 :: Will get the first item on the list
*              flag = 1 :: will get all the items on the list
* Returns:    Nothing
*/

/*
* Function:    closeSubmit
* Description: Close the TDOCS Submit dialog and perform any cleanup necessary.
*
* Parameters:  None
* Returns:    Nothing
*/

/*
* Function:    checkForLoadOrSubmit
* Description: Determine function to perform
*
* Parameters:  func - delete, update, submit
* Returns:    Nothing
*/

```

```

/*
* Function:   SetFieldEntryButtons
* Description: Set the Mandatory/Optional tag for each field
*              in the CDOCS record depending on the set
*
* Parameters: func - delete, update, submit
* Returns:   Nothing
*/

/*
* Function:   EnableFieldsValues
* Description: Enable the appropriate fields in the submit
*              dialog box dependent on which document set
*              is being processed
*
* Parameters: spec - string containing M,N,or O for each field where
*              M - mandatory
*              N - Not Applicable
*              O - Optional
* Returns:   Nothing
*/

/*
* Function:   ValidateSubmitField
* Description: Validata the entry for the given Submit field
*
* Parameters: i - index into submit field
*              newstr - value of field
* Returns:   Success or Failure
*/

/*
* Function:   ResetSubmit
* Description: Clear the document number, disable the reset button
*              and enable the submit button
*
* Parameters: btn - Not used (required for callback)
*              event -Not used (required for callback)
* Returns:   Nothing
*/

/*
* Function:   DisconnectClear
* Description: Remove the Clear button from the submit dialog box
*              when Update or Delete is called
*
* Parameters: None
* Returns:   Nothing
*/

```

```
/*  
 * Function:   ReconnectClear  
 * Description: Reconnect the Clear button from the submit dialog box  
 *             when Submit is called  
 *  
 * Parameters: None  
 * Returns:   Nothing  
 */
```

```

/*-----
* File name:      subset.c
* Date:           02/27/95 1000
* Author:         sv
* Description:    File contains routines to select subsets of
*                review plan numbers and titles or OITS ID
*                to generate one or more valid individual reports
*
* Encompasses:   loadSubsetReport,closeSubsetReport
*
*-----
* Modification log
*-----
*/

/*
* Function:       loadSubsetReport
* Description:    Display the Subset Report dialog
*
* Parameters:     report_title, doc_title
* Returns:        Nothing
*/

/*
* Function:       WindowButtonProc
* Description:    Callback functions for the buttons on the
*                dialog box
*
* Parameters:     button - ptr to button chosen: GENERATE, CLOSE, HELP
*                event - ptr to current event
* Returns:        Nothing
*/

```

```

/*-----
* File name:      svc.c
* Date:           4/18/94
* Author:         dtl
* Description:     This file contains the routines that deals with
*                  SQL return messages.
*-----
* Archive:        /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.svc.c
* Author:
* Revision:       2.0.6
* Date:          94/05/03-16:38:08
*-----
*
* Encompasses:    checkForSQLErrors
*
*-----
*/

/*
* Function:       checkForSQLErrors
* Description:    Check the return code from RPC for SQL errors
*
* Parameters:     checkResults - Structure containing return status
* Returns:        Success or Failure
*/

```



```

/*-----
* File name:      tdocs.c
* Date:           1/11/94
* Author:         dtl
* Description:     This file contains the calls to load all CDOCS (previously TDOCS)
*                  drivers
*-----
* Archive:        /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.tdocs.c
* Author:
* Revision:       2.0.6
* Date:           94/05/03-16:38:08
*-----
*
* Encompasses:    buttonSubmit, Help, dialogNoteProc, expandView,
*                  editTextView, gettextview, StatusErrorProc, buttonClear,
*                  TDOCSSStart, TDOCSSStop, shutdownMain, editDialogNoteProc
*-----
*/

/*
* Function:       TDOCSSStart()
*
* Description:    Initialize TDOCS. Originally, this and TDOCSSSTOP made
*                  up the main procedure when TDOCS was a standalone system.
*                  The main screen in the original TDOCS was the SUBMIT dialog
*                  and all TDOCS functions were called from it. Now, the main
*                  menu in RPDOCS calls the TDOCS functions. This function now
*                  sets up the TDOCS environment by loading all the TDOCS
*                  dialog boxes into memory and establishing the call backs
*                  to the numerous buttons and other resources that perform
*                  the TDOCS functions.
* Parameters:     None
* Returns:        Null
*/

/*
* Function:       TDOCSSStop()
*
* Description:    Destroy potential memory leaks from TDOCS
*                  resources when the user quits the system
* Parameters:     None
* Returns:        Null
*/

```

```

/*
* Function:    expandView
*
* Description: Open the TDOCS edit dialog box to allow
*              the user full screen editing of a field
* Parameters:  button - name of the TDOCS Submit field
*              to be edited
* Returns:    Nothing
*/

/*
* Function:    gettextview
*
* Description: Get a pointer to the container named in
*              dest
* Parameters:  dest - Container item tag name
* Returns:    Pointer to the container item tag named
*              in dest
*/

/*
* Function:    buttonClear
*
* Description: Clear the current record from CLEAR button
* Parameters:  clearItem - ptr to "CLEAR" item selection
*              event - ptr to current event
* Returns:    Nothing
*/

/*
* Function:    shutdownMain()
*
* Description: Exit the program when user double clicks on top menu
* Parameters:  None
* Returns:    Nothing
*/

/*
* Function:    dialogNoteProc()
*
* Description: The dialog notification procedure gets called whenever
*              any major event applies to the dialog as a whole.
*              We need to look for the event which tells us that the
*              dialog has closed.
* Parameters:  dialog - ptr to main dialog procedure
*              event - ptr to current event
* Returns:    Nothing
*/

```

```
/*  
* Function:    editDialogNoteProc()  
*  
* Description: The edit dialog notification procedure gets called when  
*              the user wants to expand a text view.  
*              We need to look for the event which tells us that the  
*              dialog has closed.  
* Parameters: dialog - ptr to main dialog procedure  
*              event - ptr to current event  
* Returns:    Nothing  
*  
*/
```

```

/*-----
* File name:      tdocerr.c
* Date:           7/11/94
* Author:         dtl
* Description:    This file contains the routines that check the TDOCS
*                text fields for proper format. (i.e. document number,
*                user name, dates, etc.)
*-----
* Archive:        /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.tdocerr.c
* Author:
* Revision:       2.0.6
* Date:           94/05/03-16:38:08
*-----
* Encompasses:   CheckTDOCSDocNum, CheckTDOCSUserName, CheckTDOCSDate,
*                CheckDay, CheckMonth, CheckYear,
*                getMaxDay, upper
*-----
*/

/*
* Function:      CheckTDOCSUserName
*
* Description:   Check the format of the user name
* Parameters:    username
* Returns:       Nothing
*/

/*
* Function:      CheckTDOCSDocNum
*
* Description:   Check the format of the documnet number
* Parameters:    docnum - Document Number
* Returns:       Nothing
*/

/*
* Function:      CheckTDOCSDate
*
* Description:   Check the format of the date
* Parameters:    field - calling field name
*                date - date
* Returns:       Nothing
*/

```

```
/*  
 * Function:    CheckDay  
 * Description: Ensure the day is valid for the given  
 *              month and year  
 *  
 * Parameters: day, month, year  
 * Returns:    True or false  
 */
```

```
/*  
 * Function:    getMaxDay  
 * Description: Determine the last day for a given  
 *              month in a given year  
 *  
 * Parameters: day, month, year  
 * Returns:    True or false  
 */
```

```
/*  
 * Function:    CheckMonth  
 * Description: Ensure the month is valid  
 *  
 * Parameters: month  
 * Returns:    True or false  
 */
```

```
/*  
 * Function:    CheckYear  
 * Description: Ensure the year is valid  
 *  
 * Parameters: year  
 * Returns:    True or false  
 */
```

```
/*  
 * Function:    upper  
 * Description: Convert the month to uppercase  
 *  
 * Parameters: month  
 * Returns:    Nothing  
 */
```

```

/*-----
* File name:      utils.c
* Date:           4/15/94
* Author:         dtl
* Description:     This file contains utility routines.
*-----
* Archive:        /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.utils.c
* Author:
* Revision:       2.0.6
* Date:          94/05/03-16:38:08
*-----
*
* Encompasses:    GetTextItemText
*
*-----
*/

/*
* Function:       GetTextItemText
*
* Description:    Get the string in the text item pointed to
*                by tag.
* Parameters:     dialog - ptr to dialog box
*                tag - ptr to textitem
* Returns:        Nothing
*/

```

```
/*-----  
* File name:      rpcvstr.c  
* Date:          1/3/95  
* Author:        dgm  
* Description:    interface RPC which use C strings to  
*                interface that uses Galaxy vstr and vchar arrays  
*-----  
* Archive:  vrpc.c  
* Author:   dgm  
* Revision: 1.5  
* Date:     1/3/95  
* Changed:  95/05/05 07:46:07  
*-----  
*/
```

7.2.2 Client H

```
/*-----  
*  
* checkin.h  
*  
* 10/6/93 - 2258 - rlm - created.  
*-----  
*  
* 03/31/94 - rlm - v1.2 modify to remove ORACLE calls and use RPC  
*          server exclusively  
*  
*-----  
* Archive:    /db3/rpdocs/development/interface/SCCS/s.checkin.h  
* Author:  
* Revision:   1.29.0  
* Date:      94/08/08-14:35:56  
*-----  
*/
```



```

/*
* client.h
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/rpc/SCCS/s.client.h
* Author:
* Revision:   1.21.0
* Date:       95/08/17-14:04:08
*-----
*
* Header stuff for rpc client.
*
* Code below is written for the following:
*
* platform          compiler          defined by compiler
*
* SMI sun           cc                __sparc
* MS windows        Watcom C/C++      __WATCOMC__
* Microsoft         C7                __MSC__
* IBM OS/2          IBM C/C++ Set     __OS2__
* Apple Macintosh   Semantic C        __MAC__
*
* __MSC__ used to build the rpc16.dll only -- it and __WATCOMC__
* are defined in makefile.win.
*
* 02/21/94 - cjm
* 08/26/94 - cjm - windows port
* 09/07/94 - cjm - merged with other ports
* 09/16/94 - cjm - merged in OS/2 port
* 02/07/95 - cjm - solaris port (removed gnuc, acc, borland stuff)
* 06/22/95 - cjm - dynamic versioning
*-----
*/

```

```

/*
 * clntrpc.h
 *
 * -----
 * Archive:    /home/samson/dlincoln/rpdocs/development/rpc/SCCS/s.clntrpc.h
 * Author:
 * Revision:   1.15.0
 * Date:       95/08/17-14:05:45
 * -----
 *
 * Header stuff for port-specific rpc client.
 *
 * Code below is written for the following:
 *
 * platform          compiler          defined by compiler
 *
 * SMI sun           cc                __sparc
 * MS windows        Watcom C/C++      __WATCOMC__
 * Microsoft         C7                __MSC__
 * IBM OS/2          IBM C/C++ Set     __OS2__
 * Apple Macintosh   Semantic C        __MAC__
 *
 * __MSC__ used to build the rpc16.dll only -- it and __WATCOMC__
 * are defined in makefile.win.
 *
 * 02/21/94 - cjm
 * 08/26/94 - cjm - windows port
 * 09/07/94 - cjm - merged with other ports
 * 09/16/94 - cjm - merged in OS/2 port
 * 02/08/95 - cjm - solaris port (removed gnuc, acc, borland stuff)
 * 06/22/95 - cjm - dynamic versioning
 * -----
 */

```

```
/*
 * common.h
 *-----
 * Archive:    /home/goliath/rmarsha/v1.1/development/interface/SCCS/s.common.h
 * Author:
 * Revision:   1.1.0
 * Date:       94/03/10-15:03:38
 *-----
 */
```

```

/*
* da.h
*
* -----
* Archive:    /home/samson/dlincoln/rpdocs/development/rpc/SCCS/s.da.h
* Author:
* Revision:   2.29.0
* Date:       95/07/07-08:50:02
* -----
*
* Header file for RPDOCS RPC.
*
* da=database access, be it Oracle or Topic or etc....
*
* rpcgen will generate da.h, da_clnt.c, da_svc.c, da_xdr.c, etc.
*
* Idea is this, a generic rpc database server.
*
* FUNCTIONS:
*
* A function, kill_N, to kill the server from the client (ala
* Bloomer's die).
*
* A single service function, rpc_N, handles all service requests (ala
* Bloomer's rip). It takes as argument an RPCPacket and returns
* RPCReturn.
*
* Service request packets are defined as follows:
*
* docset: long identifying document set as one of cds, cdm,
* tdi, qa, csp, etc.
*
* access: string of arbitrary length identifying how the server
* is to log in to the database to perform sql transactions,
* defined as "userid/passwd:t:host:instance" where host may be
* goliath or samson and instance may be test1 or prod1
*
* request: long identifying the service request, e.g., from
* simple file copy to sql transactions to document indexing, etc.
*
* validate: string of arbitrary length used to validate request
*
* data: a string of arbitrary length defining control
* information for processing the service request, e.g., userid,
* document identifier, document header, etc., or tdocs submit
* data (see format specification below)
*
* ascii: linked list of strings with maximum length that define

```

```

*   an ascii file, abstract, review, etc.
*
*   binary: a linked list of opaque data with maximum length that
*   define a WP document, an image file, etc. - may later need to
*   handle multiple binary files
*
* Service request returns are defined as follows:
*
*   status: a long identifying success, failure (of several
*   sorts)
*
*   error: a long identifying sql or file i/o error
*
*   data: a string of arbitrary length that contains rpd mismatch
*   data, labeled sql select data or unlabeled sql fetch data (see
*   format specification below)
*
*   ascii: linked list of strings with maximum length that contain
*   an ascii file
*
*   binary: a linked list of opaque data with maximum length that
*   contain a WP document, an image file, etc. - may later need
*   to handle multiple binary files
*
* For both service requests and service returns, the format of data
* will be as follows:
*
*       data= (L\2V(;V)*\1)*\0
*           or (L\2V\3V\1)*\0
*           or (V\3V)*\1)*\0
*       L   = label text
*       V   = value text
*
*       \2  = label delimiter
*       ;   = packet list delimiter
*       \3  = return list delimiter
*       \1  = value/list delimiter
*       ()  = syntactic grouping
*       *   = zero or more
*       \0  = data must always be a null-terminated string
*
* Thus, for example,
*
*       data = "" (no data)
*       data = "L\2V\1" (control)
*       data = "L\2V\1L\2V\1" (rpd checkin)
*       data = "L\2V\1L\2V;V;V;\1L\2V\1" (tdocs submit)
*       data = "L\2V\3V\1L\2V\3V\1L\2V\3V\1" (mismatch return)

```

```

*      data = "V3V1V3V1V3V1" (sql fetch)
*
* Normally this header is generated from da.x with rpcgen but FTP's
* rpcgen generates only DOS code and thus this file must be made to
* conform to the Windows API with special attention to declaring
* callback functions and pointers as FAR. Besides, only program and
* version numbers change over time.
*
* Process for updating RPC code:
*
* 1 Copy da.h, da_clnt.c, da_xdr.c to a backup as templates.
* 2 Next cd to ../rpc; run make to rpcgen generate new da.h,
*   da_clnt.c, da_xdr.c files; copy these files to ../rpc16.
* 3 Then use templates to cut and paste adding FAR to all callback
*   functions and pointers.
*
* Code below is written for the following:
*
* platform          compiler          defined by compiler
*
* SMI sun           cc                __sparc
* MS windows        Watcom C/C++      __WATCOMC__
* Microsoft         C7                __MSC__
* IBM OS/2          IBM C/C++ Set     __OS2__
* Apple Macintosh   Semantic C        __MAC__
*
* __MSC__ used to build the rpc16.dll only -- it and __WATCOMC__
* are defined in makefile.win.
*
* 02/18/94 - cjm - created
* 08/12/94 - cjm - moved constants to dadefine.h
* 08/26/94 - cjm - windows port
* 09/07/94 - cjm - merged with other ports
* 09/16/94 - cjm - merged in OS/2 port
* 02/07/95 - cjm - solaris port (removed gnucc, acc, borland stuff)
* 06/22/95 - cjm - dynamic versioning
*-----
*/

```

```
/*
 * dadefs.h
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/rpc/SCCS/s.dadefs.h
 * Author:
 * Revision:   2.34.0
 * Date:       95/07/10-07:52:14
 *-----
 *
 * Defines shared by client and server.
 *
 * 08/12/94 - cjm - moved constants from da.x
 * 09/07/94 - cjm - merged with other ports
 * 09/16/94 - cjm - merged in OS/2 port
 *-----
 */
```

```
/*
 * define.h
 *-----
 * Archive:    /home/goliath/rmarsha/v1.1/development/interface/SCCS/s.define.h
 * Author:
 * Revision:   1.1.0
 * Date:       94/03/10-15:03:38
 *-----
 */
```



```
/*-----  
* File name:  docdb.h  
* Date:      3/12/95  
* Author:    dtl  
*-----  
* Archive:    /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.docdb.h  
* Author:  
* Revision:   2.0.6  
* Date:      94/05/03-16:38:08  
*-----  
*/
```

```
/*-----  
* File name:  globals.h  
* Date:      2/28/94  
* Author:    dtl  
*-----  
* Archive:    /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.globals.h  
* Author:  
* Revision:   2.0.6  
* Date:      94/05/03-16:38:08  
*-----  
*/
```

```
/*-----  
* File name:  help.h  
* Date:      6/10/95  
* Author:    dtl  
*-----  
* Archive:   /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.help.h  
* Author:  
* Revision:  2.0.6  
* Date:      94/05/03-16:38:08  
*-----  
*/
```

```
/*-----  
* File name:  hitlist.h  
* Date:      6/11/95  
* Author:    dtl  
*-----  
* Archive:    /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.hitlist.h  
* Author:  
* Revision:   2.0.6  
* Date:      94/05/03-16:38:08  
*-----  
*/
```

```
/*
 * init.h
 *-----
 * Archive:    /home/goliath/rmarsha/v1.1/development/interface/SCCS/s.init.h
 * Author:
 * Revision:   1.1.0
 * Date:      94/03/10-15:03:40
 *-----
 */
```

```

/*-----
* logon.h
*
* 10/07/93 - rlm - created with defines from logon.c.
* 03/25/94 - rlm - modified to develop rpd v1.1
* 07/14/94 - rlm - added new privilege levels for TDOCS
*-----
* Archive:    /home/goliath/rmarsha/v1.1/development/interface/SCCS/s.logon.h
* Author:
* Revision:   1.2.0
* Date:       94/03/28-16:37:14
*-----
*/

/*
* Array of top level menu privileges for each user.  The -2's are place holders
* to fill out the array for looping purposes.
*/

```

```
/*
*-----
* Archive:    /home/goliath/ajohnson/v1.1/development/interface/SCCS/s.main.h
* Author:
* Revision:   1.6.0
* Date:       94/05/03-16:38:23
*-----
*/
```

```
/*
 * maint.h
 *-----
 * Archive:    /db3/rpdocs/development/interface/SCCS/s.maint.h
 * Author:
 * Revision:   1.28.0
 * Date:       94/08/08-14:35:19
 *-----
 */
```



```
/*
 * options.h
 *-----
 * Archive:    /home/goliath/rmarsha/v1.1/development/interface/SCCS/s.options.h
 * Author:
 * Revision:   1.1.0
 * Date:       94/03/10-15:03:44
 *-----
 */
```

```
/*-----  
* File name:  pref.h  
* Date:      6/12/95  
* Author:    dtl  
*-----  
* Archive:   /home/samson/dlincoln/rpdocs/development/interface/SCCS/s.pref.h  
* Author:  
* Revision:  2.0.6  
* Date:      94/05/03-16:38:08  
*-----  
*/
```

```
/*  
 * prefs.h  
 *-----  
 * Archive:    /db3/rpdocs/development/interface/SCCS/s.prefs.h  
 * Author:  
 * Revision:   1.25.0  
 * Date:      94/08/08-14:35:39  
 *-----  
 */
```

```
/*  
 * report.h  
 *-----  
 * Archive:    /db3/rpdocs/development/interface/SCCS/s.report.h  
 * Author:  
 * Revision:   1.25.0  
 * Date:      94/08/08-14:35:42  
 *-----  
 */
```

```
/*
*-----
* Archive:    /home/goliath/dlincoln/rpdocs2.0/development/interface/search.h
* Author:     dlincoln
* Revision:    1.6.0
* Date:       94/09/15
*-----
* 12/13/94 dgm  use char* for string literals
*-----
*/
```

```
/*-----  
*  
* startwp.h  
*  
* 07/14/94 - 1102 - rlm - created.  
*-----  
* Archive:    /user1/goliath/dlincoln/rpdocs/development/interface/SCCS/s.startwp.h  
* Author:  
* Revision:   2.1.0  
* Date:      94/11/11-11:19:21  
*-----  
*/
```

```
/* Submit.h
*
* History Creation Date: 2/28/94 DTL
*
* Description: Header file for the submit C file
*
*/
```

```
/* svc.h
*
* History Creation Date:    10/18/94 DTL
*
* Description:              This file contains the messages associated with
*                           the server status and error flags
*
* -----
* 12/13/94 dgm  use char* fro string literals
* -----
*/
```



```
/* Tdocs.h
*
* History Creation Date: 1/24/94 DTL
*
* Description: Header file for the main TDOCS C file
*
*/
```

```
/*-----  
* File name:  rpcvstr.h  
* Date:      1/3/95  
* Author:    dgm  
* Description: interface RPC which use C strings to  
*            interface that uses Galaxy vstr and vchar arrays  
*-----  
* Archive:   vrpc.h  
* Author:    dgm  
* Revision:   1.5  
* Date:      1/3/95  
* Changed:   95/05/05 07:46:06  
*-----  
*/
```

8 INDEXED INTERNAL MODULE COMMENTS—SERVER/BATCH

Chapter 8 contains listings of the internal comments for both the server and batch custom code modules. An index cross-referencing various functions, actions, and subjects of interest precedes the listings of the C code (both .c and .pc files). The .pc files are source code which is precompiled by the ORACLE Pro*C precompiler to produce .c files for final compiling along with other .c source file (see Chapter 3 for further discussion). The header files (.h) comments are also provided, but are not indexed.

8.1 SERVER/BATCH MODULE INDEX

Table 8-1. Server c/pc index

Description	Module
Change Working Directory	doc.c
Check Format	rpdkhk.pc
Free RPC Buffer	da_svr_proc.c
Get Help List—Get Help File	hlp.pc
Initializes, Sets and Frees Environmental Structure	env.c
Intellitag	rpdkhk.pc
Kill RPC Loop	da_svr_proc.c
Macro Styler	rpdkhk.pc
MALLOC, REALLOC and FREE Replacements	mem.c
Produce Document Lists	rpddoc.pc
Read/Write ASCII Files	doc.c
Read/Write Binary Files	doc.c
RPC Packet to Buffer Conversion	dat.c
RPC Request Dispatch Loop	da_svr_proc.c
Server CDOCS Check In	cdocsin.pc
Server CDOCS Define	cdocsin.pc
Server CDOCS Delete	cdocsin.pc
Server CDOCS Insert	cdocsin.pc
Server CDOCS Rename	cdocsin.pc

Table 8-1. Server c/pc index (cont'd)

Description	Module
Server CDOCS Update	cdocsin.pc
Server Check for Duplicate	rpd.pc
Server Commit in Database	db.pc
Server DBA Log In	db.pc
Server Get Document Sets	cdocsin.pc
Server Get Help	db.pc
Server Log In	cdocsin.pc
Server Log Out	cdocsin.pc
Server Report Writer Management Functions	rpddtd.pc
Server Retire	cdocsin.pc
Server Rollback	cdocsin.pc
Server Check In (formerly RPD)	rpdchk.pc
Server Define (formerly RPD)	rpddef.pc
Server Report Writer (formerly RPD)	rpdrw.pc
Server Reports (formerly RPD)	rpdrpt.pc
Server Retire (formerly RPD)	rpdret.pc
Server Set Alarm	cdocsin.pc
Server Set Document Set	rpd.pc
Server Set Version (formerly RPD)	rpd.pc
Server Set Title	rpd.pc
Server Singly Linked List Functions	sllist.c
Server Status Check	rpd.pc
Server Status Report Functions	rpt.c
Server Synchronization Functions	rp dout.pc
Server New Acquisitions (formerly TDOCS)	tdocsrpt.pc
Server Check In (formerly TDOCS)	tdocschk.pc
Server Check Out (formerly TDOCS)	tdocschk.pc

Table 8-1. Server c/pc index (cont'd)

Description	Module
Server Delete (formerly TDOCS)	tdocsdel.pc
Server Document Information Management (formerly TDOCS)	tdocsdat.pc
Server Label (formerly TDOCS)	tdocslbl.pc
Server Name Management (formerly TDOCS)	tdocsnms.pc
Server Output Synchronization (formerly TDOCS)	tdocsout.pc
Server Record Management Functions (formerly TDOCS)	tdocsrec.pc
Server Submission Functions (formerly TDOCS)	tdocssub.pc
Server Submit (formerly TDOCS)	tdocsdoc.pc
Server Update Functions (formerly TDOCS)	tdocsupd.pc
Server User Log In	db.pc
Server User Log Out	db.pc
Server User Management Functions	usr.pc
Start Logging—Log Results	log.c
Validate Document	rpdchk.pc

Table 8-2. Batch c/pc index

Description	Module
Batch Log In	batchdb.pc
Batch Log Out	batchdb.pc
Batch Rollback	batchdb.pc
Batch Intellitag	rpdbat.pc
Batch Pieces	rpdbat.pc
Batch Report Writer	rpdbat.pc
Batch Document Functions (formerly RPD)	rpdbat.pc
Batch Check In (formerly RPD)	rpdchk.pc
Batch Delete Archive (formerly RPD)	rpdchk.pc

Table 8-2. Batch c/pc index (cont'd)

Description	Module
Batch Move Archive (formerly RPD)	rpdchk.pc
Batch Update Check In (formerly RPD)	rpdchk.pc
Batch Define (formerly RPD)	rpddef.pc
Batch Retire (formerly RPD)	rpdrct.pc
Batch Report Writer	rpdrw.pc
Batch Process Sets (formerly TDOCS)	tdocsbat.pc
Batch Update (formerly TDOCS)	tdocsbat.pc
Batch Submit (formerly TDOCS)	tdocsbat.pc
Batch Delete (formerly TDOCS)	tdocsbat.pc
Batch Update (formerly TDOCS)	tdocsdoc.pc
Batch Submit (formerly TDOCS)	tdocsdoc.pc
Batch Delete (formerly TDOCS)	tdocsdoc.pc
Batch Remove File (formerly TDOCS)	tdocsdoc.pc
Batch Restore File (formerly TDOCS)	tdocsdoc.pc
Batch Copy File (formerly TDOCS)	tdocsdoc.pc
Batch Hydrology Functions (formerly TDOCS)	tdocshyd.pc
Batch NIST Functions	tdocsnst.pc
Batch Record Management Functions (formerly TDOCS)	tdocsrec.pc
Batch Reporting Functions (formerly TDOCS)	tdocsrpt.pc
Batch TOPIC Load (formerly TDOCS)	tdocstop.pc

8.2 SERVER/BATCH COMMENT LISTINGS

8.2.1 Server C

```

/*
 * cdocsin.pc
 *
 * This server module implements TDOCS and RPD input synchronization.
 *
```

```

* 03/27/95 - cjm
* 05/16/95 - cjm - upgrade to Pro*C 2.0
* 07/10/95 - cjm - new sync design
* 07/20/95 - cjm - split in and out
* 12/07/95 - sv - changed the file from tdocsin.pc to cdocsin.pc
*               to incorporate rpd syncin code
*-----
* Archive:      /home/samson/dlincoln/rpdocs/development/server/SCCS/s.cdocsin.pc
* Author:
* Revision:     1.2.0
* Date:        96/01/03-09:36:20
*-----
*/

```

```

/*
* TDOCSSyncInSetAlarm
*
* Sets alarm signal handler, calculates seconds to next hour, sets alarm.
* Returns 0 on success or 1 on error.
*
* Called by TDOCSSyncInAlarm.
*
* Calls nothing.
*/

```

```

/*
* SigSyncChild
*
* This acknowledges child exits so system does change them into zombies.
*
* Function call signal sets this callback.
*
* This is a carry over from SunOS and probably not needed for Solaris.
*/

```

```

/*
* TDOCSSyncInAlarm
*
* Synchronization alarm handler: sets up signal handler for sync-in done
* and forks synchronization of test and production databases and documents,
* and signals sync-in done. Returns 0 on success, 1 on failure.
*
* Called by system signal.
*
* Calls LogSyncError, TDOCSSyncIn, TDOCSSyncInSetAlarm.
*/

```

```

/*

```

```

* RPDSyncIn
*
* Processes sync in on a document set by document set basis. Returns 0 on
* success, 1 on failure.
*
* Called by TDOCSSyncInAlarm.
*
* Calls LogSynchronization, TDOCSSyncInLogIn, ChangeWorkingDir,
* RPDSyncInGetDocSets, TDOCSSyncInLogOut, RPDSyncInDocSet,
* TDOCSTreeDataList.
*/

/*
* Common messages for both TDOCS and RPD syncin :
* define variables ::
*
* TDOCS_SYNC_LOG_DATA
* TDOCS_SYNC_LOG_DONE
*
* Functions ::
*
* TDOCSSyncInLogIn, ChangeWorkingDir, TDOCSSyncInLogOut, TDOCSTreeDataList
*/

/*
* This code checks the first rpd partition and if the documents
* are not found in the first partition it will quit trying to syncin. This
* may restrict one from doing syncin for other document sets in rpd. That is
* the reason I have this code commented out and replaced it with the line
* above
*/

/*
* TDOCSSyncIn
*
* Processes sync in on a document set by document set basis. Returns 0 on
* success, 1 on failure.
*
* Called by TDOCSSyncInAlarm.
*
* Calls LogSynchronization, TDOCSSyncInLogIn, ChangeWorkingDir,
* TDOCSSyncInGetDocSets, TDOCSSyncInLogOut, TDOCSSyncInDocSet,
* TDOCSTreeDataList.
*/

/*
* The following lines of code perform tdocs syncin for each document set
* at a time. But the code seems to quit performing syncin as soon as it

```


* comes across a partion which does not have anything to syncin.

*/

/*

* TDOCSSyncInLogIn

*

* Logs in to either the production or test database for synchronization.

* Returns 0 on success, 1 on failure.

*

* Called by TDOCSSyncIn.

*

* Calls nothing.

*/

/*

* TDOCSSyncInLogOut

*

* Logs out of database.

*

* Called by TDOCSSyncIn.

*

* Calls nothing.

*/

/*

* RPDSyncInGetDocSets

*

* Fetches list of document sets, converted to lower case, that might require
synchronization. Returns 0 on success, 1 on failure.

*

* Called by RPDSyncIn.

*

* Calls SLLInitialize, LogSyncError, MemAlloc, SLLAppend, TDOCSFreeDataList.

*/

/*

* Common messages and subroutines for TDOCS and RPD syncin

* define variables ::

*

* TDOCS_SYNC_MEM_ERR

* TDOCS_SYNC_DOCS_ERR

*

* functions ::

*

* TDOCSFreeDataList

*/

/*

* TDOCSSyncInGetDocSets

```

*
* Fetches list of document sets, converted to lower case, that might require
* synchronization. Returns 0 on success, 1 on failure.
*
* Called by TDOCSSyncIn.
*
* Calls SLLInitialize, LogSyncError, MemAlloc, SLLAppend, TDOCSSyncFreeDataList.
*/

/*
* RPDSSyncInDocSet
*
* Processes sync in on a document by document basis. For each header file
* found in the remote out-queue, the header is read, document id modified,
* record inserted, files renamed to local incoming, and committed or
* rolled back. Returns 0 on success, 1 on failure.
*
* Called by RPDSSyncIn.
*
* Calls LogSynchronization, LogSyncError, RPDSSyncInHeader, RPDSSyncInDefine,
* RPDSSyncInFiles, RPDSSyncInCheckin, RPDSSyncInRetire, TDOCSSyncInRollback,
* RPDSSyncInCommit.
*/

/*
* TDOCSSyncInDocSet
*
* Processes sync in on a document by document basis. For each header file
* found in the remote out-queue, the header is read, document id modified,
* record inserted, files renamed to local incoming, and committed or
* rolled back. Returns 0 on success, 1 on failure.
*
* Called by TDOCSSyncIn.
*
* Calls LogSynchronization, LogSyncError, TDOCSSyncInHeader, TDOCSSyncInInsert,
* TDOCSSyncInFiles, TDOCSSyncInUpdate, TDOCSSyncInDelete, TDOCSSyncInRollback,
* TDOCSSyncInCommit.
*/

/*
* RPDSSyncInCommit
*
* Commits synchronization transaction.
*
* Called by RPDSSyncInDocSet.
*
* Calls nothing.
*/

```

```

/*
 * TDOCSSyncInRollback
 *
 * Rolls back synchronization transaction.
 *
 * Called by TDOCSSyncInDocSet.
 *
 * Calls nothing.
 */

/*
 * RPDSyncInHeader
 *
 * Deletes or reads header file into header host variables. Returns 0 on
 * success, 1 on failure.
 *
 * Called by TDOCSSyncInDocSet.
 *
 * Calls LogSyncError, MemAlloc, GetData, GetListData, MemFree,
 * TDOCSFreeAssociatedData.
 */

/*
 * TDOCSSyncInHeader
 *
 * Deletes or reads header file into header host variables. Returns 0 on
 * success, 1 on failure.
 *
 * Called by TDOCSSyncInDocSet.
 *
 * Calls LogSyncError, MemAlloc, GetData, GetListData, MemFree,
 * TDOCSFreeAssociatedData.
 */

/*
 * RPDSyncInRetire
 *
 * Sets new document id and inserts header for batch loading. Returns 0 on
 * success, 1 on failure.
 *
 * Called by RPDSyncInDocSet.
 *
 * Calls LogSyncError, TDOCSInsertData, TDOCSInsertAssocData.
 */

/*
 * RPDSyncInCheckin
 *
 * Sets new document id and inserts header for batch loading. Returns 0 on

```

```

* success, 1 on failure.
*
* Called by RPDSyncInDocSet.
*
* Calls LogSyncError, TDOCSInsertData, TDOCSInsertAssocData.
*/

/*
* RPDSyncInDefine
*
* Sets new document id and inserts header for batch loading. Returns 0 on
* success, 1 on failure.
*
* Called by RPDSyncInDocSet.
*
* Calls LogSyncError, TDOCSInsertData, TDOCSInsertAssocData.
*/

/*
* TDOCSSyncInInsert
*
* Sets new document id and inserts header for batch loading. Returns 0 on
* success, 1 on failure.
*
* Called by TDOCSSyncInDocSet.
*
* Calls LogSyncError, TDOCSInsertData, TDOCSInsertAssocData.
*/

/*
* TDOCSSyncInUpdate
*
* Updates and marks document-numbered record for batch update. Returns 0 on
* success, 1 on failure.
*
* Called by TDOCSSyncInDocSet.
*
* Calls TDOCSDeleteAssocData, TDOCSUpdateData, TDOCSInsertAssocData,
* LogSyncError.
*/

/*
* TDOCSSyncInDelete
*
* Marks document-numbered record for batch deletion. Returns 0 on success, 1
* on failure.
*
* Called by TDOCSSyncInDocSet.
*

```

```

* Calls TDOCSDeleteData, LogSyncError.
*/

/*
* RPDsyncInFiles
*
* Renames remote outgoing document files to local incoming files. On success
* it also deletes the document header file. Returns 0 on success, 1 on failure.
*
* Called by RPDsyncInDocSet.
*
* Calls LogSyncError.
*/

/*
* TDOCSSyncInFiles
*
* Renames remote outgoing document files to local incoming files. On success
* it also deletes the document header file. Returns 0 on success, 1 on failure.
*
* Called by TDOCSSyncInDocSet.
*
* Calls LogSyncError.
*/

/*
* RPDsyncInSetDocIDInstID
*
* For definition sets new document id and instance id for document set, in
* which case document set and id must have already been set. Otherwise sets
* current document id and current and new instance id for document set, in
* which case document set and id and document number must have already been
* set. Returns SVC_SUCCESS or SVC_SQL_ERROR.
*
* Called by
*
* Calls SQLError.
*/

/*
* RPDsyncInSetStatusAndID
*
* Sets status and status id. Returns SVC_SUCCESS or SVC_SQL_ERROR.
*
* Called by RPDDefine, RPDCheckin, RPDRetire.
*
* Calls SQLError.
*/

```

```

/*
 * da_svc_proc.c
 *
 * This server module implements RPC services.
 *
 * 02/21/94 - cjm
 * 04/01/94 - cjm - merged tdocs with rpd
 * 02/08/95 - cjm - solaris port
 *
 *-----
 * Archive:      /home/samson/dlincoln/rpdocs/development/server/SCCS/s.da_svc_proc.c
 * Author:
 * Revision:     2.23.0
 * Date:         95/11/13-10:59:26
 *-----
 */

```

```

/*
 * rpc_N
 *
 * Main service request switch maps service request ids to service functions.
 * It also frees previous results, logs user in (where request is validated and
 * logged), rolls back on failure, and logs user out. Returns pRPCReturn,
 * pointer to service results packet.
 *
 * Called by da_prog_N.
 *
 * Calls FreeReturn, LogUserIn, GetUserInfo, GetUserNames, GetUserList,
 * ChangePassword, AddUser, DropUser, GetNames, RPDGetDocSets,
 * RPDGetActiveDocList, RPDGetNonRetDocList, RPDGetDocSubsetList,
 * RPDGetCDSCDMDoc, RPDDDBReport, RPDOITSSStatusReport, RPDSHORTDBAReport,
 * RPDDDBAReport, RPDDefine, RPDCheckFormat, RPDCheckin, RPDRetire,
 * RPDGetReportList, RPDReportWriter, TDOCSGetDocSets, TDOCSGetDocSetSpec,
 * TDOCSSubmitDocument, TDOCSGetRecord, TDOCSUpdateDocument,
 * TDOCSDeleteDocument, TDOCSUndeleteDocument, TDOCSLabelDocuments,
 * TDOCSRelabelDocuments, TDOCSCheckoutDocument, TDOCSCheckinDocument,
 * TDOCSCirculationReport, TDOCSNewDocsReport, TDOCSGetRefRptList,
 * TDOCSReferenceReport, TDOCSStatisticsReport, SQLDTRDFetchReports,
 * SQLDTRDSelectReport, SQLDTRDInsertReport, SQLDTRDUpdateReport,
 * SQLDTRDDeleteReport, SQLDTRDQueryReport, SQLDTRDGetFile, SQLDTRDPutFile,
 * Rollback, LogUserOut, GetHelp, GetHelpList.
 */

```

```

/*
 * kill_N
 *
 * Unregisters service and exits.
 *
 * Called by da_prog_N

```

```
*  
* Calls nothing.  
*/  
  
/*  
* FreeReturn  
*  
* Frees any previous results. The xdr_free routine does not work error free.  
*  
* Called by rpc_N  
*  
* Calls nothing.  
*/
```

```

/*
 * dat.c
 *
 * This server module implements common RPC packet data encoding and decoding.
 *
 * 07/25/94 - cjm
 * 02/08/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.dat.c
 * Author:
 * Revision:   2.22.0
 * Date:       95/11/13-10:59:27
 *-----
 */

/*
 * PutListData
 *
 * Puts fetched data in RPCData.pReturn->data. Returns SVC_SUCCESS or
 * SVC_MEM_ERROR.
 *
 * Called by RPDGetDocSets, RPDGetActiveDocList, RPDGetNonRetDocList,
 * RPDGetDocSubsetList, SQLDTDFetchReports, SQLDTDQueryReport, RPDGetReportList,
 * TDOCSCheckinDocument, TDOCSGetDocSets, GetNames, TDOCSGetRefRptList,
 * GetUserNames, GetUserList.
 *
 * Calls MemAlloc, MemRealloc.
 */

/*
 * GetData
 *
 * Data is passed in a string as described in da.x. This function gets data for
 * a single-valued field. Returns pointer to data.
 *
 * Called by RPDCheckinGetData, RPDDefineGetData, SQLDTDGetData,
 * SQLDTDSelectReport, SQLDTDDeleteReport, SQLDTDSetFilePath,
 * RPDRetireGetData, TDOCSCheckoutDocument, TDOCSCheckinDocument,
 * TDOCSGetGeneralData, TDOCSGetDocumentData, TDOCSUndeleteDocument,
 * TDOCSGetRecord, TDOCSInitNewDocsReport, TDOCSOpenCirculationCursor,
 * TDOCSSyncInHeader, ChangePassword, AddUser, DropUser.
 *
 * Calls nothing.
 */

```



```

/*
 * GetListData
 *
 * Data is passed in a string as described in da.x. This function gets data for
 * a multi-valued field. On error, sets error flag. Returns pointer to a list
 * of data values.
 *
 * Called by TDOCSGetDocumentData, TDOCSRelabelDocuments, TDOCSSyncInHeader.
 *
 * Calls nothing.
 */

/*
 * SetDataError
 *
 * Sets data error string.
 *
 * Called by TDOCSGetDocumentData, TDOCSCheckData, TDOCSGetDeleteData,
 * TDOCSGetSubmitData, TDOCSGetUpdateData.
 *
 * Calls nothing.
 */

```

```

/*
 * db.pc
 *
 * This server module declares common database host variables, and implements
 * common database stuff like log in, log out, commit, rollback, etc.
 *
 * 02/21/94 - cjm
 * 04/01/94 - cjm - merged tdocs with rpd
 * 04/02/94 - cjm - fixed log in bug
 * 02/08/95 - cjm - solaris port
 * 02/20/95 - cjm - removed reliance on tcp for db access
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 * 10/26/95 - sv - added a new field called documentset_id
 *
 * -----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.db.pc
 * Author:
 * Revision:   2.22.0
 * Date:       95/11/13-10:59:30
 * -----
 */

/*
 * SQLSuccess
 *
 * Records success and returns it. Returns SVC_SUCCESS.
 *
 * Called by SQLDTDInsertReport, SQLDTDUpdateReport, SQLDTDSelectReport,
 * SQLDTDDeleteReport, TDOCSUndeleteDocument.
 *
 * Calls Commit.
 */

/*
 * SQLNotFound
 *
 * Records status, and returns it. Returns status.
 *
 * Called by RPDGetDocSets, RPDGetActiveDocList, RPDGetNonRetDocList,
 * RPDGetDocSubsetList, SQLDTDSelectReport, SQLDTDDeleteReport,
 * SQLDTDFetchReports, SQLDTDQueryReport, RPDFetchDBCursor,
 * FetchOITSSStatusCursor, RPDSHORTDBAREPORT, RPDDBARREPORT, RPDGetReportList,
 * RPDReportWriter, TDOCSCheckoutDocument, TDOCSCheckinDocument,
 * TDOCSCheckData, TDOCSDeleteDocument, TDOCSUndeleteDocument,
 * TDOCSGetDeleteData, TDOCSGetDocSets, TDOCSGetDocSetSpec, TDOCSLabelDocuments,
 * GetNames, TDOCSUpdateDeleteData, TDOCSDeleteData, TDOCSRetrieveData,
 * TDOCSGetUpdateData, GetUserNames, GetUserList.
 *

```

```

* Calls Commit, Rollback.
*/

/*
* SQLError
*
* Reports, records, and returns sql error. Returns SVC_SQL_ERROR.
*
* Called by LogUserIn, Commit, Rollback, RPDSetDocIDInstID, RPDCheckStatus,
* RPDSetStatusAndID, RPDSetTitle, RPDSetVersionNameIDAndDate,
* RPDSetPartitionAndFilename, RPDCreateNewInstance, RPDSetAndCheckVersion,
* RPDCheckinInsert, RPDCheckinUpdate, RPDUpdateTitle, RPDDefineInsert,
* RPDGetCDSCDMDoc, RPDGetDocSets, RPDGetActiveDocList, RPDGetNonRetDocList,
* RPDGetDocSubsetList, SQLDTDInsertReport, SQLDTDUpdateReport,
* SQLDTDSelectReport, SQLDTDDeleteReport, SQLDTDFetchReports,
* SQLDTDQueryReport, RPDRetireInsert, RPDDDBReport, RPDOITSSStatusReport,
* RPDOpenDBCursor, RPDFetchDBCursor, OpenOITSSStatusCursor,
* FetchOITSSStatusCursor, RPDShortDBAReport, RPDDBAReport, RPDGetReportList,
* RPDReportWriter, TDOCSCheckoutDocument, TDOCSCheckinDocument,
* TDOCSGetGeneralData, TDOCSCheckData, TDOCSDeleteDocument,
* TDOCSUndeleteDocument, TDOCSGetDeleteData, TDOCSDocSetFromTitle,
* TDOCSGetDocSets, TDOCSGetDocSetSpec, TDOCSHydroRefReport,
* TDOCSOpenHydroRefCursor, TDOCSFetchHydroRefCursor,
* TDOCSRelabelDocuments, TDOCSLabelDocuments, GetNames, TDOCSNISTRefReport,
* TDOCSOpenNISTRefCursor, TDOCSFetchNISTRefCursor, TDOCSInsertData,
* TDOCSInsertAssocData, TDOCSUpdateDeleteData, TDOCSDeleteData,
* TDOCSRetrieveData, TDOCSGetAssociatedData, TDOCSInitNewDocsReport,
* TDOCSCirculationReport, TDOCSOpenNewDocCursor, TDOCSFetchNewDocCursor,
* TDOCSOpenCirculationCursor, TDOCSFetchCirculationCursor,
* TDOCSGetRefRptList, TDOCSReferenceReport, TDOCSGetSubmitData,
* TDOCSGetUpdateData, GetUserInfo, ChangePassword, AddUser,
* DropUser, GetUserNames, GetUserList.
*
* Calls LogError, Rollback.
*/

/*
* LogDBAIn
*
* Logs in dba as test.
*
* Called by main.
*
* Calls LogError.
*/

```

```

/*
* LogUserIn
*
* Logs in userid/passwd@t:<host>:<instance> as found in pPacket->access. Sets
* pReturn status and error. Also changes to the working directory specified in
* the database access string.
*
* Called by main.
*
* Calls LogRequest, ChangeWorkingDir, SQLError.
*/

/*
* LogUserOut
*
* Logout userid/passwd@t:<host>:<instance>. Does NOT set pReturn status or
* error.
*
* Called by main.
*
* Calls LogRequest, LogError.
*/

/*
* Commit
*
* Commits oracle transactions. Sets pReturn status and error.
*
* Called by SQLSuccess, SQLNotFound, RPDCheckFormat, RPDCheckin, RPDDefine,
* RPDGetCDSCDMDoc, RPDRetire, TDOCSCheckoutTDIDocument,
* TDOCSCheckinTDIDocument, TDOCSSubmitCSPRecord, TDOCSUpdateCSPRecord,
* TDOCSDeleteCSPRecord, TDOCSRelabelDocuments, TDOCSRelabelDocuments, AddName,
* DropName, TDOCSSubmitQAREcord, TDOCSUpdateQAREcord, TDOCSDeleteQAREcord,
* TDOCSGetRefRptList, TDOCSSubmitTDIRecord, TDOCSUpdateTDIRecord,
* TDOCSDeleteTDIRecord, GetUserInfo, ChangePassword, AddUser, DropUser.
*
* Calls SQLError.
*/

/*
* Rollback
*
* Rolls back oracle transactions. Does NOT set pReturn->status.
*
* Called by SQLNotFound, SQLError, RPDCheckin, TDOCSLabelDocuments,
* TDOCSReferenceReport, TDOCSSyncInDocSet.
*
* Calls LogError.
*/

```

```

/*
 * doc.c
 *
 * This server module implements common working directory change and document
 * read and write.
 *
 * 02/24/94 - cjm
 * 04/01/94 - cjm - merged tdocs with rpd
 * 04/21/94 - cjm - fixed read file problem
 * 02/08/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.doc.c
 * Author:
 * Revision:   2.22.0
 * Date:      95/11/13-10:59:32
 *-----
 */

/*
 * ChangeWorkingDir
 *
 * Changes working directory. Returns SVC_SUCCESS or SVC_OP_ERROR.
 *
 * Called by LogUserIn, TDOCSSyncIn.
 *
 * Calls nothing.
 */

/*
 * ReadAsciiFile
 *
 * Reads ascii file to pReturn ascii. Returns SVC_SUCCESS, SVC_MEM_ERROR, or
 * SVC_FILE_ERROR.
 *
 * Called by RPDDbReport, RPDITSSStatusReport, RPDSHORTDBAReport, RPDDbReport,
 * TDOCSHydroRefReport, TDOCSNISTRefReport, TDOCSNewDocsReport,
 * TDOCS CirculationReport, TDOCSStatisticsReport.
 *
 * Calls MemAlloc.
 */

/*
 * WriteAsciiFile
 *
 * Writes ascii list to file named by filepath. Frees list nodes. Returns
 * SVC_SUCCESS or SVC_FILE_ERROR.
 *

```

```

* Called by nothing.
*
* Calls MemFree.
*/

/*
* ReadBinaryFile
*
* Reads binary file to pReturn binary. Returns SVC_SUCCESS, SVC_MEM_ERROR, or
* SVC_FILE_ERROR.
*
* Called by RPDGetCDSCDMDoc, SQLDTDGetFile, RPDReportWriter.
*
* Calls MemAlloc.
*/

/*
* WriteBinaryFile
*
* Writes binary list to file named by filepath. Frees list nodes. Returns
* SVC_SUCCESS or SVC_FILE_ERROR.
*
* Called by RPDWriteConvertAndParse, SQLDTDPutFile.
*
* Calls MemFree.
*/

```

```

/*
 * env.c
 *
 * This server module implements RPDOCS environment settings.
 *
 * This module is shared by server and batch.
 *
 * 03/31/95 - cjm
 *
 * -----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.env.c
 * Author:
 * Revision:   1.12.0
 * Date:       95/11/13-10:59:34
 * -----
 */

```

```

/*
 * RPDOCSVoidEnv
 *
 * NULLs environment structure.
 *
 * Called by RPDOCSInitEnv.
 *
 * Calls nothing.
 */

```

```

/*
 * RPDOCSSetEnv
 *
 * Sets section's environment setting-identified variable to setting-identified
 * value. Return 0 on success, 1 on failure.
 *
 * Called by RPDOCSInitEnv.
 *
 * Calls nothing.
 */

```

```

/*
 * RPDOCSFreeEnv
 *
 * Frees space pointed to by environment variables.
 *
 * Called by nothing.
 *
 * Calls nothing.
 */

```

```
/*  
 * RPDOCSInitEnv  
 *  
 * Sets environment variables from ini file. Returns 0 on success, 1 on failure.  
 *  
 * Called by Server::main, Batch::main.  
 *  
 * Calls RPDOCSVoidEnv, RPDCCSetEnv.  
 */
```



```

/*
 * hlp.pc
 *
 * This module implements functions for information on, and handling of help.
 *
 * 07/10/95 - cjm
 *
 * -----
 * Archive:    /home/samson/cmoehle/rpdocs/development/server/SCCS/s.hlp.pc
 * Author:
 * Revision:   1.1.0
 * Date:       95/07/10-10:28:21
 * -----
 */

/*
 * GetHelp
 *
 * Gets help header and text by privilege, button, and subkey - on error returns
 * help for button==0, subkey==0. Expects "PRIVILEGE2<privilege>\1BUTTON2
 * <key>\2SUBKEY\2<subkey>\1". Writes results to pReturn data. Sets pReturn
 * status and error.
 *
 * Called by rpc_N
 *
 * Calls MemAlloc, Commit, SQLError.
 */

/*
 * GetHelpList
 *
 * Gets list of help (button, subkey, and header) for given privilege. Writes
 * to pReturn data, and values to list. Sets pReturn status and error.
 *
 * Called by rpc_N
 *
 * Calls PutListData, SQLNotFound, SQLError.
 */

```

```

/*
 * log.c
 *
 * This server module implements service request logs.
 *
 * 09/07/94 - cjm
 * 02/08/95 - cjm - solaris port
 *
 * -----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.log.c
 * Author:
 * Revision:   2.23.0
 * Date:       95/11/13-10:59:38
 * -----
 */

/*
 * RPDOCSLogFailMail
 *
 * Mails system administrator startup failure message.
 *
 * Called by main.
 *
 * Calls nothing.
 */

/*
 * RPDOCSInitLog
 *
 * Initializes logging. Returns 0 on success, 1 on failure.
 *
 * Called by main.
 *
 * Calls nothing.
 */

/*
 * LogRequest
 *
 * Logs a service request
 *
 * Called by LogUserIn, LogUserOut.
 *
 * Calls nothing.
 */

```

```

/*
 * LogError
 *
 * Logs an error string
 *
 * Called by SQLError, LogDBAIn, LogUserOut, Rollback, LogErrorID.
 *
 * Calls nothing.
 */

/*
 * LogErrorID
 *
 * Logs an error id (int)
 *
 * Called by RPTFileOpen, RPTPrRecord.
 *
 * Calls LogError.
 */

/*
 * LogSynchronization
 *
 * Logs synchronization.
 *
 * Called by LogSyncError, TDOCSSyncIn.
 *
 * Calls nothing.
 */

/*
 * LogSyncError
 *
 * Logs synchronization error and return 1.
 *
 * Called by TDOCSSyncInAlarm, TDOCSSyncInGetDocSets, TDOCSSyncInDocSet,
 * TDOCSSyncInHeader, TDOCSSyncInInsert, TDOCSSyncInUpdate, TDOCSSyncInDelete,
 * TDOCSSyncInFiles.
 *
 * Calls LogSynchronization.
 */

```

```

/*
 * mem.c
 *
 * This server module implements common memory management.
 *
 * 07/11/94 - cjm
 * 02/08/95 - cjm - solaris port
 *
 * -----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.mem.c
 * Author:
 * Revision:   2.22.0
 * Date:       95/11/13-10:59:41
 * -----
 */

/*
 * MemAlloc
 *
 * Allocates memory. Returns 0 on success or 1 on failure.
 *
 * Called by PutListData, ReadAsciiFile, ReadBinaryFile, RPDSetAndCheckVersion,
 * RPDValidateDocument, RPDGetDocSubsetList, RPDShortDBAReport, RPDDBAReport,
 * RPTFileOpen, RPTSetRecord, TDOCSGetDocSetSpec, TDOCSLabelDocuments,
 * TDOCSFormatData, TDOCSGetAssociatedData, TDOCSWriteNewDocsReport,
 * TDOCSSubmitDocument, TDOCSSyncInGetDocSets, TDOCSSyncInHeader, GetUserInfo.
 *
 * Calls nothing.
 */

/*
 * MemRealloc
 *
 * Reallocates memory. Returns 0 on success or 1 on failure.
 *
 * Called by PutListData, RPTSetRecord, TDOCSFormat, TDOCSWriteNewDocsReport.
 *
 * Calls nothing.
 */

```

```
/*  
 * MemFree  
 *  
 * Frees memory and sets pointer to NULL. Returns 0 on success or 1 on failure.  
 *  
 * Called by WriteAsciiFile, WriteBinaryFile, RPDGetDocSubsetList,  
 * RPDShortDBAReport, RPDDBAReport, RPTFileClose, TDOCSFreeDataList,  
 * TDOCSWriteNewDocsReport, TDOCSSyncInHeader.  
 *  
 * Calls nothing.  
 */
```

```

/*
 * rpd.pc
 *
 * This module declares RPD database host variables, and implements RPD
 * functions common to define, checkin, and retire.
 *
 * 02/21/94 - cjm
 * 04/01/94 - cjm - merged tdocs with rpd
 * 05/15/94 - cjm - removed topic loading to batch
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 * -----
 * Archive:      /home/samson/dlincoln/rpdocs/development/server/SCCS/s.rpd.pc
 * Author:
 * Revision:     2.22.0
 * Date:         95/11/13-10:59:44
 * -----
 */

/*
 * RPDDocumentIsDefined
 *
 * Checks for duplicate definitions. Document set and document number must have
 * been set. Returns count of records that match document set and document
 * number, or 0 on error (not found).
 *
 * Called by RPDDefineCheckData, RPDRetireCheckData.
 *
 * Calls nothing.
 */

/*
 * RPDSetDocSet
 *
 * Sets document set from pPacket docset. Returns SVC_SUCCESS, SVC_OP_ERROR,
 * or SVC_SQL_ERROR.
 *
 * Called by RPDDefine, RPDCheckFormat, RPDCheckin, RPDRetire.
 *
 * Calls nothing.
 */

/*
 * RPDSetDocIDInstID
 *
 * For definition sets new document id and instance id for document set, in
 * which case document set and id must have already been set. Otherwise sets

```

```

* current document id and current and new instance id for document set, in
* which case document set and id and document number must have already been
* set. Returns SVC_SUCCESS or SVC_SQL_ERROR.
*
* Called by RPDDefine, RPDCheckFormat, RPDCheckin, RPDRetire.
*
* Calls SQLError.
*/

```

```

/*
* RPDCheckStatus
*
* Checks that status is not DEFINE or CHECKIN - only one per day per
* document allowed. Returns SVC_SUCCESS or SVC_SQL_ERROR.
*
* Called by RPDCheckin, RPDCheckFormat.
*
* Calls SQLError.
*/

```

```

/*
* RPDSetStatusAndID
*
* Sets status and status id. Returns SVC_SUCCESS or SVC_SQL_ERROR.
*
* Called by RPDDefine, RPDCheckin, RPDRetire.
*
* Calls SQLError.
*/

```

```

/*
* RPDSetTitle
*
* Gets title for document set, document id and instance id. Returns
* SVC_SUCCESS or SVC_SQL_ERROR.
*
* Called by RPDValidateDocument.
*
* Calls SQLError.
*/

```

```

/*
* RPDSetMajorMinorVersion
*
* Sets major and minor version.
*
* Called by RPDDefine.
*

```

```

* Calls nothing.
*/

/*
* RPDSetVersionNameIDAndDate
*
* Sets version name, id and date to selection for name or to empty strings.
* Then copies to appropriate host variable. Returns SVC_SUCCESS or
* SVC_SQL_ERROR.
*
* Called by RPDDefineGetData, RPDDefine, RPDCheckinGetData, RPDRetireGetData.
*
* Calls SQLError.
*/

/*
* RPDSetPartitionAndFilename
*
* Sets current and new partition based on document set id, and file name based
* on document id and instance id. Returns SVC_SUCCESS or SVC_SQL_ERROR.
*
* Called by RPDDefine, RPDCheckFormat, RPDCheckin, RPDRetire.
*
* Calls SQLError.
*/

/*
* RPDSetDocumentSortID
*
* Sets sort ID from document number.
*
* Called by RPDDefine.
*
* Calls nothing.
*/

/*
* RPDCreateNewInstance
*
* Creates new instance given document set id, document id and instance id.
* Assumes that represents the previous instance.
*
* Called by RPDCheckinInsert, RPDRetireInsert.
*
* Calls SQLError.
*/

```



```

/*
 * rpdchk.pc
 *
 * This server module implements RPD rpd checkin.
 *
 * 03/02/94 - cjm
 * 04/01/94 - cjm - merged tdocs with rpd
 * 05/15/94 - cjm - removed topic loading to batch
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.rpdchk.pc
 * Author:
 * Revision:   2.23.0
 * Date:       95/12/22-08:32:24
 *-----
 */

/*
 * RPDMacroStyler
 *
 * Generates for each checkin file, fn, a styled file, fn.wp. Copies file to
 * file.wp, gets its time stamp, runs the macro styler against it, gets the
 * time stamp for the output file, compares time stamps and if different
 * assumes success, and if not assumes error. Returns status SVC_SUCCESS or
 * I SVC_OP_ERROR, with error set to SVC_NO_WPFILE_ERROR, SVC_MACRO_RUN_ERROR,
 * SVC_NO_MACRO_ERROR, or SVC_MACRO_FAIL_ERROR.
 *
 * Called by RPDCheckFormat.
 *
 * Calls RPDRunWPTG.
 */

/*
 * RPDIntellitag
 *
 * Tags a styled file with intellitag. Macro styler generated for each file,
 * fn, a styled file, fn.wp. Intellitag generates for each fn.wp a fn.wp.sgm.
 * The files are not significant in the end, only the log file intellitag
 * generates. A child is forked and wp monitored when not debugging. Returns
 * status SVC_SUCCESS or SVC_OP_ERROR, with error set to SVC_NO_WPFILE_ERROR,
 * SVC_MACRO_RUN_ERROR, SVC_NO_MACRO_ERROR, or SVC_MACRO_FAIL_ERROR.
 *
 * Called by RPDCheckFormat.
 *
 * Calls RPDRunWPTG, RPDIntellitagLog.
 */

```

```

/*
 * RPDIntellitagLog
 *
 * Checks intellitag log file for errors and reports them. Returns status as
 * SUCCESS or SVC_OP_ERROR, in which case error is set to SVC_NO_LOGFILE_ERROR,
 * SVC_NO_RULEFILE_ERROR, or SVC_TG_FAIL_ERROR.
 *
 * Called by RPDIntellitag.
 *
 * Calls nothing.
 */

/*****
 * RPDCheckinGetData
 *
 * Gets checkin data from pPacket. If "MAJOR" version name is found
 * h_majorversionname is set and h_minorversionname is set to "", and vice
 * versa. For OITS, all checkins are major version changes.
 *
 * Called by RPDCheckFormat, RPDCheckin.
 *
 * Calls GetData, RPDSetVersionNameIDAndDate.
 */

/*
 * RPDCheckinCheckData
 *
 * Checks checkin data from pPacket. Returns SVC_SUCCESS or SVC_OP_ERROR.
 *
 * Called by RPDCheckin.
 *
 * Calls nothing.
 */

/*
 * RPDSetAndCheckVersion
 *
 * Checks user version against current version plus major or minor increment.
 * Also sets new version. On mismatch, writes mismatch to pReturn->data.
 * Where user version is "" a major version change is assumed and so
 * incremented. Returns SVC_SUCCESS or SVC_OP_ERROR.
 *
 * Called by RPDCheckFormat, RPDCheckin.
 *
 * Calls MemAlloc, SQLError.
 */

```

```

/*
 * RPDCheckinInsert
 *
 * Inserts a check in record by selection and modification of last archived
 * record identified by document set, document id and instance id. Returns
 * SVC_SUCCESS or SVC_SQL_ERROR.
 *
 * Called by RPDCheckin.
 *
 * Calls RPDCreateNewInstance, SQLError.
 */

/*
 * RPDCheckinUpdate
 *
 * Updates status to ARCHIVED, partition to archived partition of record
 * identified by document set, document id and instance id. Returns SVC_SUCCESS
 * or SVC_SQL_ERROR.
 *
 * Called by nothing.
 *
 * Calls SQLError.
 */

/*
 * RPDWriteConvertAndParse
 *
 * Writes and converts document. Returns SVC_SUCCESS or SVC_FILE_ERROR.
 *
 * Called by RPDCheckFormat, RPDCheckin.
 *
 * Calls WriteBinaryFile.
 */

/*
 * RPDValidateDocument
 *
 * Validates parsed document set, document number, and title against expected
 * input. If request=RPD_FORCE_CHECKIN_REQUEST then title is updated. If
 * mismatches are found then pReturn data is set to the mismatch(es) as follows:
 * (L\2E\3R\1)* where L labels mismatch as DOCUMENT_SET, DOC_NUM or TITLE, E is
 * the expected value, and R is the received value. Returns SVC_SUCCESS,
 * SVC_SQL_ERROR, SVC_FILE_ERROR, or SVC_OP_ERROR.
 *
 * Called by RPDCheckFormat, RPCCheckin.
 *
 * Calls MemAlloc, RPDUpdateTitle.
 */

```

```

/*
* RPDUpdateTitle
*
* Updates title for document record identified by document set, document id
* and instance id. Returns SVC_SUCCESS or SVC_SQL_ERROR.
*
* Called by RPDValidateDocument.
*
* Calls SQL_Error.
*/

/*
* RPDCheckFormat
*
* Checks cds, cdm or rps format.
*
* -) Moves data (document set, document number) to host variables.
* -) Converts and checks document set, document number, and title match.
* -) Runs macrostyler and intellitag against the document.
*
* Expects pPacket->docset=="CDS", "CDM", or "RPS"; and pPacket->data=
* "DOC_NUM2~~~~\1". Sets pReturn status and, possibly, error. Status might be
* SVC_SUCCESS, SVC_SQL_ERROR (error = Oracle error code), or SVC_OP_ERROR
* (SVC_DATA_SET_ERROR, SVC_MISMATCH_ERROR). And for macrostyler,
* SVC_MACROSTYLER_ERROR (SVC_CANT_FIND_WPFILE_ERROR).
*
* Called by rpc_N
*
* Calls RPDSetDocSet, RPDCheckinGetData, RPDSetDocIDInstID, RPDCheckStatus,
* RPDSetAndCheckVersion, RPDSetPartitionAndFilename, RPDWriteConvertAndParse,
* RPDValidateDocument, RPDMacroStyler, RPDIntellitag, Commit.
*/

/*
* RPDCheckin
*
* Checks in a cds, cdm, or rps.
*
* -) Moves data (document set, document number, title, major version name)
* to host variables.
* -) Checks lengths of data and version match.
* -) Updates current record status and partition.
* -) Inserts new active record into Oracle.
* -) Converts and checks document set, document number, and title match.
* -) Loads new document into Topic.
* -) Commits update and insert.
*
* Expects pPacket->docset=="CDS", "CDM", or "RPS"; and pPacket->data=

```

```

* "DOC_NUM2~~~~\1MAJOR_VERSION_NAME2~~~~\1VERSION2~~~~\1" or
* "DOC_NUM2~~~~\1MINOR_VERSION_NAME2~~~~\1VERSION2~~~~\1". Sets * pReturn
* status and, possibly, error. Status might be SVC_SUCCESS, SVC_SQL_ERROR
* (error = Oracle error code), or SVC_OP_ERROR (SVC_DATA_SET_ERROR,
* SVC_DATALEN_ERROR, SVC_DATA_DUP_ERROR, SVC_MISMATCH_ERROR,
* SVC_ARCHIVE_ERROR
* SVC_WORKINGDIR_ERROR, SVC_VANCANT_DOC_ERROR, SVC_CONVERT_ERROR,
* SVC_BUILD_ERROR, SVC_LOAD_ERROR, SVC_STORE_ERROR).
*
* Called by rpc_N
*
* Calls RPDSetDocSet, RPDCheckinGetData, RPDCheckinCheckData,
* RPDSetDocIDInstID, RPDCheckStatus, RPDSetAndCheckVersion,
* RPDSetStatusAndID, RPDSetPartitionAndFilename, RPDCheckinInsert,
* RPDWriteConvertAndParse, RPDValidateDocument, Commit.
*/

```

```

/*
 * rpddef.pc
 *
 * This server module implements RPD define.
 *
 * 03/02/94 - cjm
 * 04/01/94 - cjm - merged tdocs with rpd
 * 05/15/94 - cjm - removed topic loading to batch
 * 04/13/94 - cjm - added rps doc type
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.rpddef.pc
 * Author:
 * Revision:   2.23.0
 * Date:       95/12/22-08:32:22
 *-----
 */

/*
 * RPDDefineGetData
 *
 * Gets define data from pPacket. Returns SVC_SUCCESS or SVC_SQL_ERROR.
 *
 * Called by RPDDefine.
 *
 * Calls GetData, RPDSetVersionNameIDAndDate.
 */

/*
 * RPDDefineCheckData
 *
 * Checks define data from pPacket. Returns SVC_SUCCESS or SVC_OP_ERROR.
 *
 * Called by RPDDefine.
 *
 * Calls RPDDocumentIsDefined.
 */

/*
 * RPDDefineInsert
 *
 * Inserts define record -- all host variables must have been set. Returns
 * SVC_SUCCESS or SVC_SQL_ERROR.
 *
 * Called by RPDDefine.
 */

```

```

* Calls SQLError.
*/

/*
* RPDDefine
*
* Defines a cds, cdm, rps, oits record.
*
* -) Moves data (document set, document number, title, major version name)
*   to host variables.
* -) Checks lengths of data and duplication of definition.
* -) Completes definition record (document id, instance id, date, partition,
*   file name).
* -) Inserts record into Oracle.
* -) Loads vacant document into Topic.
* -) Commits insertion.
*
* Expects pPacket->docset=="CDS", "CDM", "RPS", "OITS"; and pPacket->data=
* "DOC_NUM2~~~~\1TITLE2~~~~\1MAJOR_VERSION_NAME2~~~~\1". For OITS,
* REVIEW_PLAN_NUM is OITS_ID. Sets pReturn status and, possibly, error. Status
* might be SVC_SUCCESS, SVC_SQL_ERROR (error = Oracle error code), or
* SVC_OP_ERROR (SVC_DATA_SET_ERROR, SVC_DATALEN_ERROR,
* SVC_DATA_DUP_ERROR,
* SVC_WORKINGDIR_ERROR, SVC_VANCANT_DOC_ERROR, SVC_CONVERT_ERROR,
* SVC_HEADER_ERROR, SVC_BUILD_ERROR, SVC_LOAD_ERROR, SVC_STORE_ERROR).
*
* Called by rpc_N
*
* Calls RPDSetDocSet, RPDDefineGetData, RPDDefineCheckData,
* RPDSetDocIDInstID, RPDSetStatusAndID, RPDSetMajorMinorVersion,
* RPDSetVersionNameIDAndDate, RPDSetPartitionAndFilename,
* RPDSetDocumentSortID, RPDDefineInsert, Commit.
*/

```

```

/*
* rpddoc.pc
*
* This server module implements RPD document information and handling.
*
* 02/24/94 - cjm
* 04/01/94 - cjm - merged tdocs with rpd
* 05/15/94 - cjm - removed topic loading to batch
* 02/08/95 - cjm - solaris port
* 05/16/95 - cjm - upgrade to Pro*C 2.0
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.rpddoc.pc
* Author:
* Revision:   2.22.0
* Date:      95/11/13-11:02:27
*-----
*/

/*
* RPDGetCDSCDMDoc
*
* Gets WORKINGDIR directory from pPacket access instance. Gets PARTITION and
* FILENAME for document identified by document number in pPacket data (expects
* DOC_NUM2docnum\1") from database. Assumes /db1 and archive. From that
* constructs path to file as /db1/<WORKINGDIR>/archive/<PARTITION>/<FILENAME>.
* Then reads file to pReturn binary list. Also get RPS docs. Sets pReturn
* status and error.
*
* Called by rpc_N
*
* Calls Commit, ReadBinaryFile, SQLError.
*/

/*
* RPDGetDocSets
*
* Gets list of document sets. Writes label to pReturn data, and values to list.
* Sets pReturn status and error.
*
* Called by rpc_N
*
* Calls PutListData, SQLNotFound, SQLError.
*/

/*
* RPDGetActiveDocList
*

```



```

* Gets list of active documents' document numbers, titles and versions. Writes
* labels to pReturn data, and values to list. Sets pReturn status and error.
*
* Called by rpc_N
*
* Calls PutListData, SQLNotFound, SQLError.
*/

```

```

/*
* RPDGetNonRetDocList
*
* Gets list of non-retired documents' document numbers, titles and versions.
* Writes labels to pReturn data, and values to list. Sets pReturn status and
* error.
*
* Called by rpc_N
*
* Calls PutListData, SQLNotFound, SQLError.
*/

```

```

/*
* RPDGetDocSubsetList
*
* Gets list of document numbers and titles for whose pieces are consistent with
* the intended report query. Writes labels to pReturn data, and values to list.
* Sets pReturn status and error.
*
* Called by rpc_N
*
* Calls MemAlloc, MemFree, PutListData, SQLNotFound, SQLError.
*/

```

```

/*
 * rpddtd.pc
 *
 * This server module implements RPD report writer SQL*DTD.
 *
 * 01/17/95 - cjm
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.rpddtd.pc
 * Author:
 * Revision:   1.18.0
 * Date:       95/11/13-10:59:47
 *-----
 */

/*
 * SQLDTDInsertReport
 *
 * Inserts a SQL*DTD report record.
 *
 * Called by rpc_N
 *
 * Calls SQLDTDGetData, SQLSuccess, SQLError.
 */

/*
 * SQLDTDUpdateReport
 *
 * Updates a SQL*DTD record by record id.
 *
 * Called by rpc_N
 *
 * Calls SQLSuccess, SQLError.
 */

/*
 * SQLDTDGetData
 *
 * Get SQL*DTD data for insert and update. Returns SVC_SUCCESS or SVC_OP_ERROR.
 *
 * Called by SQLDTDInsertReport, SQLDTDUpdateReport.
 *
 * Calls GetData.
 */

```

```

/*
 * SQLDTDSelectReport
 *
 * Selects report data given report id.
 *
 * Called by rpc_N
 *
 * Calls GetData, SQLSuccess, SQLNotFound, SQLError.
 */

/*
 * SQLDTDDeleteReport
 *
 * Delete a SQL*DTD report record by report id.
 *
 * Called by rpc_N
 *
 * Calls GetData, SQLSuccess, SQLNotFound, SQLError.
 */

/*
 * SQLDTDFetchReports
 *
 * Fetches report id, titles, help.
 *
 * Called by rpc_N
 *
 * Calls PutListData, SQLNotFound, SQLError.
 */

/*
 * SQLDTDQueryReport
 *
 * Runs a SQL*DTD query and returns tag, search text results.
 *
 * Called by rpc_N
 *
 * Calls PutListData, SQLNotFound, SQLError.
 */

/*
 * SQLDTDGetFile
 *
 * Retrieves WP macro (<report>.wpm) or Intellitag DTD (<report>.dtd) file.
 *
 * Called by rpc_N.
 *
 * Calls SQLDTDSetFilePath, ReadBinaryFile.
 */

```

```

/*
 * SQLDTPutFile
 *
 * Stores WP macro (<report>.wpm) or Intellitag DTD
 * (<report>.dtd) file.
 *
 * Called by rpc_N
 *
 * Calls SQLDTPutFilePath, WriteBinaryFile.
 */

/*
 * SQLDTPutFilePath
 *
 * Sets filepath for getting and putting SQLDTP files. Returns SVC_SUCCESS or
 * SVC_OP_ERROR.
 *
 * Called by SQLDTPutFile, SQLDTPGetFile.
 *
 * Calls GetData.
 */

```

```

/*
 * rpdout.pc
 *
 * This server module implements RPD output synchronization.
 *
 *-----
 * Archive:    /home/samson/sanjeev/rpdocs/development/server/SCCS/s.tdocsout.pc
 * Author:
 * Revision:   1.5.0
 * Date:       95/11/13-11:01:45
 *-----
 */

/*
 * RPDsyncOut
 *
 * Returns SVC_SUCCESS or SVC_OP_ERROR (SVC_SYNC_OUT_ERROR).
 *
 * Called by RPDSyncOutDocument
 *
 * Calls RPDsyncOutCheckin.
 */

/*
 * RPDsyncOutNameExt
 *
 * Computes sync out file name and transaction extension appended to filenames.
 * Must be done to avoid overwriting transactions before the remote site grabs
 * them. Returns SVC_SUCCESS or SVC_OP_ERROR.
 *
 * Called by RPDsyncOutDefine.
 *
 * Calls nothing.
 */

/*
 * RPDsyncOutCheckin
 *
 * Writes header info to submit control file and copies files to document set's
 * outgoing synchronization directory. Returns SVC_SUCCESS or SVC_OP_ERROR
 * (SVC_SYNC_OUT_ERROR).
 *
 * Called by RPDsyncOut.
 *
 * Calls RPDsyncOutFiles, RPDsyncOutHeader.
 */

```

```

/*
 * RPDsyncOutDefine
 *
 * Writes header info to delete control file and copies files to document set's
 * outgoing synchronization directory. Returns SVC_SUCCESS or SVC_OP_ERROR
 * (SVC_SYNC_OUT_ERROR).
 *
 * Called by RPDsyncOut.
 *
 * Calls RPDsyncOutExt, RPDsyncOutHeader.
 */

```

```

/*
 * RPDsyncOutRetire
 *
 * Writes header info to delete control file and copies files to document set's
 * outgoing synchronization directory. Returns SVC_SUCCESS or SVC_OP_ERROR
 * (SVC_SYNC_OUT_ERROR).
 *
 * Called by RPDsyncOut.
 *
 * Calls RPDsyncOutExt, RPDsyncOutHeader.
 */

```

```

/*
 * RPDsyncOutFiles
 *
 * Copies document files from incoming or archive/images to document set's
 * outgoing synchronization directory. Returns SVC_SUCCESS or SVC_OP_ERROR
 * (SVC_SYNC_OUT_ERROR).
 *
 * Called by RPDsyncOutCheckin
 *
 * Calls nothing.
 */

```

```

/*
 * RPDsyncOutIncomingFiles
 *
 * Copies document files from incoming to document set's outgoing
 * synchronization directory. Returns SVC_SUCCESS or SVC_OP_ERROR
 * (SVC_SYNC_OUT_ERROR).
 *
 * Called by RPDsyncOutFiles.
 *
 * Calls nothing.
 */

```

```

/*
 * RPDSThread
 *
 * Writes header info based on request to temporary file and then moves to a
 * synchronization control file. Move is done to signal data is ready for
 * synchronization. Returns SVC_SUCCESS or SVC_OP_ERROR (SVC_SYNC_OUT_ERROR).
 *
 * Called by RPDSThreadCheckin.
 *
 * Calls SLLHead, SLLNext.
 */

```

```

/*
 * Now move it to signal ready
 * if the request is DEFINE then get it ready for the server to
 * process the file
 * else for retire and checkin get it ready for the batch to complete
 * and change the name of the file
 */

```

```

/*
 * rpdret.pc
 *
 * This server module implements RPD retire.
 *
 * 03/02/94 - cjm
 * 04/01/94 - cjm - merged tdocs with rpd
 * 05/15/94 - cjm - moved topic loading to batch
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.rpdret.pc
 * Author:
 * Revision:   2.23.0
 * Date:      95/12/22-08:32:21
 *-----
 */

/*
 * RPDRetireGetData
 *
 * Gets retire data from pPacket.
 *
 * Called by RPDRetire.
 *
 * Calls GetData, RPDSetVersionNameIDAndDate.
 */

/*
 * RPDRetireCheckData
 *
 * Checks retire data from pPacket. Returns SVC_SUCCESS or SVC_OP_ERROR.
 *
 * Called by RPDRetire.
 *
 * Calls RPDDocumentIsDefined.
 */

/*
 * RPDRetireInsert
 *
 * Inserts a retired record by selection and modification of last archived
 * record identified by document set, document id and instance id. Returns
 * SVC_SUCCESS or SVC_SQL_ERROR.
 *
 * Called by RPDRetire.
 */

```



```

* Calls RPDCreateNewInstance, SQLError.
*/

/*
* RPDRetire
*
* Retires a cds, cdm, or rps.
*
* -) Moves data (document set, document number, major version name) to host
*   variables.
* -) Checks lengths of data and existence of definition.
* -) Gets primary key on most recent record.
* -) Inserts a retire record.
* -) Commits Oracle transactions.
*
* Expects pPacket->docset=="CDS", "CDM", or "RPS";and pPacket->data=
* "DOC_NUM2~~~~\1MAJOR_VERSION_NAME2~~~~\1". Sets pReturn status and,
* possibly, error.
*
* Called by rpd_N
*
* Calls RPDSetDocSet, RPDRetireGetData, RPDRetireCheckData, RPDSetDocIDInstID,
* RPDSetStatusAndID, RPDSetPartitionAndFilename, RPDRetireInsert, Commit.
*/

```

```

/*
 * rpd rpt.pc
 *
 * This server module implements RPD reports.
 *
 * 02/26/94 - cjm
 * 04/01/94 - cjm - merged tdocs with rpd
 * 06/15/94 - cjm - added report writer stuff
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.rpd rpt.pc
 * Author:
 * Revision:   2.22.0
 * Date:       95/11/13-11:00:58
 *-----
 */

/*
 * RPDDDBReport
 *
 * Generates and returns status or content report. Report includes document
 * number, title, document type, version, date, and status. Status report
 * includes only the most recent version of a document, content report all.
 *
 * Called by rpc_N
 *
 * Calls RPDOpenDBCursor, RPDFetchDBCursor, RPTFileOpen, RPTSetRecord,
 * RPTPrntRecord, RPDDDBCounts, RPTFileClose, ReadAsciiFile, SQLError.
 */

/*
 * RPDOITSStatusReport
 *
 * Generates and returns OITS status. Report includes OITS ID, topic, last
 * update date, and resolution status.
 *
 * Called by rpc_N
 *
 * Calls OpenOITSStatusCursor, FetchOITSStatusCursor, RPTFileOpen, RPTSetRecord,
 * RPTPrntRecord, RPDDDBCounts, RPTFileClose, ReadAsciiFile, SQLError.
 */

/*
 * RPDDDBCounts
 *
 * Prints record counts for db report. Returns 0 on success, 1 on failure.

```

```

*
* Called by RPDDDBReport.
*
* Calls RPTPrtLine.
*/

/*
* RPDOpenDBCursor
*
* Opens cursor on rpd for document report. Returns SVC_SUCCESS or
* SVC_SQL_ERROR.
*
* Called by RPDDDBReport.
*
* Calls SQLError.
*/

/*
* RPDFetchDBCursor
*
* Fetches data from cursor on rpd for document report. Returns SVC_SUCCESS,
* SVC_NO_MORE_RECORDS (end of data) or SVC_SQL_ERROR.
*
* Called by RPDDDBReport.
*
* Calls SQLNotFound, SQLError.
*/

/*
* OpenOITSStatusCursor
*
* Opens cursor OITS status cursor for document report. Returns SVC_SUCCESS or
* SVC_SQL_ERROR.
*
* Called by RPDOITSSStatusReport.
*
* Calls SQLError.
*/

/*
* FetchOITSSStatusCursor
*
* Fetches data from cursor on rpd for document report. Returns SVC_SUCCESS,
* SVC_NO_MORE_RECORDS (end of data) or SVC_SQL_ERROR.
*
* Called by RPDOITSSStatusReport.
*
* Calls SQLNotFound, SQLError.
*/

```

```

/*
 * RPDShortDBAReport
 *
 * Dumps document type, document id, instance id, document number, and status
 * record data for document type.
 *
 * Called by rpc_N
 *
 * Calls MemAlloc, RPTFileOpen, RPTPrtLine, RPTFileClose, ReadAsciiFile,
 * SQLNotFound, SQLError, MemFree, SQLNotFound, SQLError.
 */

```

```

/*
 * RPDDBAReport
 *
 * Dumps all record data for document type. Reference to checkout removed to
 * comments.
 *
 * Called by rpc_N
 *
 * Calls MemAlloc, RPTFileOpen, RPTPrtLine, RPTFileClose, ReadAsciiFile,
 * SQLNotFound, SQLError, MemFree.
 */

```

```

/*
 * RPDGetReportList
 *
 * Returns list of report writer reports, titles, types, and helps. Sets
 * pReturn status and error.
 *
 * Called by rpc_N
 *
 * Calls PutListData, SQLNotFound, SQLError.
 */

```

```

/*
 * rpdw.pc
 *
 * This server module implements RPD report writer.
 *
 * 10/31/94 - cjm
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 * -----
 * Archive:      /home/samson/dlincoln/rpdocs/development/server/SCCS/s.rpdw.pc
 * Author:
 * Revision:     1.25.0
 * Date:         95/11/13-11:01:04
 * -----
 */

/*
 * RPDReportWriter
 *
 * Gets report name from report title, reads and returns it. Sets return status
 * as SVC_SUCCESS, SVC_SQL_ERROR (error as sql error), or SVC_RPT_ERROR.
 *
 * Called by rpc_N
 *
 * Calls RPDGenerateReport, ReadBinaryFile, SQLNotFound, SQLError.
 */

/*
 * RPDRWWriter
 *
 * Runs the RPD report writer writer utility to generate an SGML report.
 *
 * Called by RPDGenerateReport.
 *
 * Calls nothing.
 */

/*
 * RPDRWWPChar
 *
 * Run the RPD report writer wpchar utility to convert some SGML to WP
 * characters.
 *
 * Called by RPDGenerateReport.
 *
 * Calls nothing.
 */

```

```

/*
 * RPDRWBackendMacro
 *
 * Runs the RPD report writer macro in WP to convert the SGML report to WP
 * format.
 *
 * Called by RPDGenerateReport.
 *
 * Calls RPDRunWPTG.
 */

/*
 * RPDRunWPTG
 *
 * Runs wp or tg command line with system call and monitors its progress,
 * killing it if it terminates. IMPORTANT: the command line must background the
 * process! Returns 0 on success, 1 on failure.
 *
 * Called by RPDRWBackendMacro.
 *
 * Calls nothing.
 */

/*
 * RPDGenerateReport
 *
 * Runs report writer on current report (h_rwreport):
 *
 * -) calls writer
 * -) calls wpchar
 * -) calls backend macro processor
 *
 * Returns SVC_SUCCESS or error status.
 *
 * Called by RPDReportWriter.
 *
 * Calls RPDRWWriter, RPDRWWPChar, RPDRWBackendMacro.
 */

```

```

/*
 * rpt.c
 *
 * This server module implements common report writing tasks.
 *
 * 04/14/94 - cjm
 * 02/08/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.rpt.c
 * Author:
 * Revision:   2.22.0
 * Date:       95/11/13-11:01:06
 *-----
 */

/*
 * RPTFileOpen
 *
 * Initializes report. Opens report file. For headers specify in a format
 * string all spacing, title, labels including a \f as first char and a %3d
 * for page numbering. For margin and maxlines consider a page of courier 12
 * point prints 80 chars by 60 lines. Include header lines. For colwidths don't
 * exceed page width. This must be an array, the last cell a flag == 0.
 * Returns SVC_SUCCESS or SVC_FILE_ERROR.
 *
 * Called by LogErrorID, RPDDDBReport, RPDOITSSStatusReport, RPDSHORTDBAREport,
 * RPDDDBAREport, TDOCSHydroRefReport, TDOCSNISTRefReport,
 * TDOCSInitNewDocsReport, TDOCS CirculationReport.
 *
 * Calls LogErrorID, MemAlloc.
 */

/*
 * RPTFileClose
 *
 * Closes report and file.
 *
 * Called by LogErrorID, RPDDDBReport, RPDOITSSStatusReport, RPDSHORTDBAREport,
 * RPDDDBAREport, TDOCSHydroRefReport, TDOCSNISTRefReport,
 * TDOCSInitNewDocsReport, TDOCS CirculationReport.
 *
 * Calls MemFree.
 */

```

```

/*
* RPTPrtLine
*
* Prints special case report line(s). All paging, spacing, etc must be
* specified.
*
* Called by RPDDDBCounts, RPDSHORTDBAREport, RPDDDBAREport, TDOCSHydroRefReport,
* TDOCSNISTRefReport, TDOCSWriteNewDocsReport, TDOCSEndNewDocsReport,
* TDOCS CirculationReport.
*
* Calls LogErrorID.
*/

```

```

/*
* RPTSetRecord
*
* Formats and collects record data - no preformatting needed. The record must
* be an array of pointers to null-terminated strings (no NULL pointers) though
* they may be zero length.
*
* Called by RPDDDBReport, RPDOITSStatusReport, TDOCSWriteNewDocsReport,
* TDOCS CirculationReport.
*
* Calls MemAlloc, MemRealloc.
*/

```

```

/*
* RPTPrtRecord
*
* Prints collected record - paging if needed.
*
* Called by RPDDDBReport, RPDOITSStatusReport, TDOCSWriteNewDocsReport,
* TDOCS CirculationReport.
*
* Calls LogErrorID.
*/

```

```

/*
* RPTFormatLine
*
* Formats text into line with margins and width.
*
* Called by TDOCSHydroRefReport, TDOCSNISTRefReport.
*
* Calls nothing.
*/

```



```
/*  
 * RPTStuffLine  
 *  
 * Stuffs line with prefixed and suffixed text. For efficiency, resets line to  
 * end of line. Returns 0 if there's text to stuff otherwise 1.  
 *  
 * Called by TDOCSNISTRefReport.  
 *  
 * Calls nothing.  
 */
```

```

/*
 * sllist.c
 *
 * This server module implements common singly linked lists. Note that nodes
 * use void* to point to any data, thus code that uses sllists must allocate
 * and free data.
 *
 * It is shared by server and batch.
 *
 * 01/20/94 - cjm - created
 * 04/01/94 - cjm - merged tdocs with rpd
 * 02/08/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.sllist.c
 * Author:
 * Revision:   2.22.0
 * Date:       95/11/13-11:01:12
 *-----
 */

/*
 * SLLInitialize
 *
 * Allocates and initializes pointer to a singly linked list. Returns pointer
 * to list on success, NULL on failure (out of memory).
 *
 * Called by Server::TDOCSGetAssociatedData, Server::TDOCSSyncInGetDocSets,
 * Batch::AppendBatchReport, Batch::RPDGetBatchRecords,
 * Batch::TDOCSAppendCleanUp, Batch::TDOCSGetAssociatedData.
 *
 * Calls nothing.
 */

/*
 * SLLReset
 *
 * Resets pointer to a singly linked list. Free first!
 *
 * Called by nothing.
 *
 * Calls nothing.
 */

/*
 * SLLAppend
 *
 * Appends data to tail of a singly linked list. Use SLLInitialize to

```

```

* initialize list. Returns pointer to node on success, NULL on failure
* (out of memory).
*
* Called by Server::TDOCSGetAssociatedData, Server::TDOCSSyncInGetDocSets,
* Batch::AppendBatchReport, Batch::RPDGetBatchRecords,
* Batch::TDOCSAppendCleanUp, , Batch::TDOCSGetAssociatedData.
*
* Calls nothing.
*/

/*
* SLLHead
*
* Returns singly linked list's head node's data and initializes for
* traversal. Use SLLNext to access remaining nodes' data. Returns data on
* success, NULL on empty list.
*
* Called by Server::TDOCSRelabelDocuments, Server::TDOCSRelabelDocuments,
* Server::TDOCSInsertAssocData, Server::TDOCSFormatData,
* Server::TDOCSSyncOutHeader, Batch::DumpBatchReport,
* Batch::RPDSummaryReport, Batch::RPDBatchCheckinDocuments,
* Batch::RPDUpdateCheckinRecords, Batch::RPDBatchDefineDocuments,
* Batch::RPDUpdateDefineRecords, Batch::RPDBatchRetireDocuments,
* Batch::RPDUpdateRetireRecords, Batch::RPDDeleteRetirePieces,
* Batch::RPDPieces, Batch::RPDMacroStyler, Batch::RPDIntellitag,
* Batch::RPDIntellitagLog, Batch::TDOCSCleanUpIncoming,
* Batch::TDOCSWriteAssociatedData.
*
* Calls nothing.
*/

/*
* SLLNext
*
* Returns next node's data. Call SLLHead to initialize traversal and get
* head node's data. Returns data on success, NULL on end of list.
*
* Called by Server::TDOCSRelabelDocuments, Server::TDOCSRelabelDocuments,
* Server::TDOCSInsertAssocData, Server::TDOCSFormatData,
* Server::TDOCSSyncOutHeader, Batch::DumpBatchReport,
* Batch::RPDSummaryReport, Batch::RPDBatchCheckinDocuments,
* Batch::RPDUpdateCheckinRecords, Batch::RPDBatchDefineDocuments,
* Batch::RPDUpdateDefineRecords, Batch::RPDBatchRetireDocuments,
* Batch::RPDUpdateRetireRecords, Batch::RPDDeleteRetirePieces,
* Batch::RPDPieces, Batch::RPDMacroStyler, Batch::RPDIntellitag,
* Batch::RPDIntellitagLog, Batch::TDOCSCleanUpIncoming,
* Batch::TDOCSWriteAssociatedData.
*

```

```

* Calls nothing.
*/

/*
* SLLTail
*
* Returns singly linked list's tail node's data. Returns data on success, NULL
* on empty list.
*
* Called by nothing.
*
* Calls nothing.
*/

/*
* SLLDelete
*
* Deletes head node of singly linked list. Returns data on success, NULL on
* end of list.
*
* Called by Server::TDOCSFreeDataList, Batch::FreeDataList,
* Batch::RPDFreeBatchRecords.
*
* Calls nothing.
*/

/*
* PSLFree
*
* Frees dynamically allocated singly linked list. Use SLLDelete to delete
* nodes first.
*
* Called by nothing.
*
* Calls nothing.
*/

```

```

/*
 * tdocschk.pc
 *
 * This server module implements TDOCS checkout and checkin.
 *
 * 04/07/94 - cjm
 * 12/12/94 - cjm - redid header fields & implemented scanning
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:      /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocschk.pc
 * Author:
 * Revision:     2.24.0
 * Date:         95/11/13-11:01:19
 *-----
 */

/*
 * TDOCSCheckoutDocument
 *
 * Checks a document identified by document number out to a name or location by
 * adding a record to the tdocs circulation table. If request is for check out
 * and document is already out, an error is returned; but if the request is to
 * force check out, then a new record is created. Returns status as SVC_SUCCESS,
 * SVC_SQL_ERROR or SVC_OP_ERROR (with error as SVC_DATA_FIND_ERROR (if the
 * document cannot be found) or SVC_CHECKED_OUT_ERROR (if the document is
 * already checked out)).
 *
 * Called by rpc_N
 *
 * Calls TDOCSDocSetFromTitle, GetData, Commit, SQLNotFound, SQLError.
 */

/*
 * TDOCSCheckinDocument
 *
 * Checks a document identified by document number in by deleting it from the
 * tdocs circulation table. Returns status as SVC_SUCCESS, SVC_SQL_ERROR or
 * SVC_OP_ERROR (with error as SVC_DATA_FIND_ERROR (if the document cannot be
 * found) or SVC_CHECKED_IN_ERROR (if the document is already checked in)).
 *
 * Called by rpc_N
 *
 * Calls GetData, PutListData, Commit, SQLNotFound, SQLError.
 */

```

```

/*
 * tdocsdat.pc
 *
 * This server module implements TDOCS data handling.
 *
 * 04/01/94 - cjm
 * 12/12/94 - cjm - redid header fields & implemented scanning
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocsdat.pc
 * Author:
 * Revision:   2.24.0
 * Date:       95/11/13-11:02:45
 *-----
 */

```

```

/*
 * TDOCSGetGeneralData
 *
 * Gets document set data - document set name, sharable set, host name, and
 * source type.
 *
 * Called by TDOCSGetSubmitData, TDOCSGetUpdateData, TDOCSGetDeleteData.
 *
 * Calls TDOCSDocSetFromTitle, GetData, SQLError.
 */

```

```

/*
 * TDOCSGetDocumentData
 *
 * Gets tdocs document data from pPacket. Returns SVC_SUCCESS or SVC_MEM_ERROR.
 *
 * Called by TDOCSGetSubmitData, TDOCSGetUpdateData, TDOCSGetDeleteData.
 *
 * Calls GetData, GetListData, SetDataError, TDOCSFreeAssociatedData.
 */

```

```

/*
 * TDOCSFreeAssociatedData
 *
 * Frees authors, addressees, numbers, and codes lists.
 *
 * Called by TDOCSGetDocumentData, TDOCSRelabelDocuments, TDOCSGetRecord,
 * TDOCSInsertAssocData, TDOCSGetAssociatedData, TDOCSNewDocsReport,
 * TDOCSFetchNewDocCursor, TDOCSSyncInHeader.
 */

```

* Calls TDOCSFreeDataList.

*/

/*

* TDOCSFreeDataList

*

* Frees associated data list.

*

* Called by TDOCSFreeAssociatedData.

*

* Calls SLLDelete, MemFree, SLLFree.

*/

/*

* TDOCSCheckData

*

* Checks for required, optional and not applicable fields.

*

* Called by TDOCSGetSubmitData, TDOCSGetUpdateData, TDOCSGetDeleteData.

*

* Calls SetDataError, SQLNotFound, SQLError.

*/

```

/*
 * tdocsdel.pc
 *
 * This server module implements TDOCS document deletion.
 *
 * 12/09/94 - cjm
 * 12/12/94 - cjm - redid header fields & implemented scanning
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocsdel.pc
 * Author:
 * Revision:   1.9.0
 * Date:       95/11/13-11:01:24
 *-----
 */

/*
 * TDOCSDeleteDocument
 *
 * Marks tdi document for deletion - actual deletion done in batch.
 *
 * -) Gets client supplied data (document set and id).
 * -) Marks record for deletion or deletes record.
 *
 * Note the following cases when a delete request occurs:
 *
 *      oldstatus   oldsource   newstatus   newsource   recordaction   fileaction
 *
 *      S           0/0        -           -           delete        -
 *      S           0/N        -           -           delete        remove
 *
 *      U or L      0/0        D           0/0        update        -
 *      U or L      0/N        D           N/0        update        -
 *
 * Called by rpc_N
 *
 * Calls TDOCSGetDeleteData, TDOCSDeleteData, TDOCSSyncOut, Commit,
 * SQLNotFound, SQLError.
 */

/*
 * TDOCSUndeleteDocument
 *
 * Undeletes a document marked for deletion.
 *
 * Called by rpd_N

```



```

*
* Calls TDOCSDocSetFromTitle, GetData, SQLSuccess, SQLNotFound, SQLError.
*/

/*
* TDOCSTGetDeleteData
*
* Sets, gets, and checks tdocs delete data. If current record status is
* input/submit/update and there's old files in incoming, they are deleted.
* Returns SVC_SUCCESS,
*
* Called by TDOCSTDeleteDocument.
*
* Calls TDOCSTGetGeneralData, TDOCSTGetDocumentData, TDOCSTCheckData,
* SetDataError, SQLNotFound, SQLError.
*/

```

```

/*
 * tdocsdoc.pc
 *
 * This server module implements TDOCS file submission via ftp.
 *
 * 11/16/94 - cjm
 * 12/12/94 - cjm - redid header fields & implemented scanning
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 * 10/26/95 - sv - modified TDOCSDocSetFromTitle function to get documentset_id
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocsdoc.pc
 * Author:
 * Revision:   1.23.0
 * Date:      95/11/13-11:01:28
 *-----
 */

/*
 * TDOCSDocSetFromTitle
 *
 * Sets h_documentset, h_docuemntset_id from given h_documentsettitle.
 * Return SVC_SUCCESS or error.
 *
 * Called by TDOCSCheckoutDocument, TDOCSGetGeneralData, TDOCSGetDocSetSpec,
 * TDOCSGetRecord, TDOCSNewDocsReport, TDOCSCirculationReport,
 * TDOCSOpenCirculationCursor.
 *
 * Calls SQLError.
 */

/*
 * TDOCSGetDocSets
 *
 * Retrieves documents sets that are loadable. Sets pReturn status and error.
 *
 * Called by rpc_N
 *
 * Calls PutListData, SQLNotFound, SQLError.
 */

/*
 * TDOCSGetDocSetSpec
 *
 * Gets specification and docset sharable flag for document set.
 *
 * Called by rpc_N

```

```

*
* Calls TDOCSDocSetFromTitle, MemAlloc, Commit, SQLNotFound, SQLError.
*/

/*
* TDOCSToIncomingFiles
*
* Gets files from local submit directory that client created for submit
* or update, and renames them by document id to their incoming document
* set directory.
*
* It is expected that the ascii document will have a ".txt" extension,
* the image and figure file(s) will be ordered with three-digit
* indexes (plus a character for figures) and have a ".tif"
* or ".pcx" extension. No assumption is made about the binary.
*
* The ".txt" is mapped to
*
*    ../incoming/<docset>/<docid>.txt,
*
* the images and figures to
*
*    ../incoming/<docset>/<docid>.<index>.<extension>,
*
* and the binary to
*
*    ../incoming/<docset>/<docid>.bin
*
* subdir - submit directory
*
* Also, to handle DOS ^M, text files are copied byte by byte to remove it.
*
* Called by TDOCSSubmitDocument, TDOCSUpdateDocument.
*
* Calls TDOCSIncStatus.
*/

/*
* TDOCSIncStatus
*
* Sets status and error, removes incoming host files. Returns status.
*
* Called by TDOCSToIncomingFiles.
*
* Calls nothing.
*/

```

```
/*  
 * TDOCSCleanSubmit  
 *  
 * Cleans out submit directory on start up.  
 *  
 * Called by main.  
 *  
 * Calls nothing.  
 */
```

```

/*
* tdocslbl.pc
*
* This server module TDOCS label generation.
*
* 03/14/94 - cjm
* 12/12/94 - cjm - redid header fields & implemented scanning
* 02/08/95 - cjm - solaris port
* 05/16/95 - cjm - upgrade to Pro*C 2.0
*
* -----
* Archive:      /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocslbl.pc
* Author:
* Revision:     2.24.0
* Date:         95/11/13-11:01:37
* -----
*/

/*
* TDOCSRelabelDocuments
*
* Regenerates labels for documents in data list.
*
* Called by rpc_N
*
* Calls GetListData, SLLHead, SLLNext, Commit, TDOCSFreeAssociatedData,
* TDOCSLabelDocuments, SQLError.
*/

/*
* TDOCSLabelDocuments
*
* Generates labels for any tdi, qa, and/or csp document not yet labeled.
* Stuffs labels into the pReturn ascii linked list. For each record in tdi,
* qa, csp whose status is defined but not labeled or loaded, subject code,
* tdi/qa/csp number, and title is written to file in label format.
*
* Called by rpc_N
*
* Calls MemAlloc, TDOCSLabelTitle, SQLNotFound, Rollback, SQLError.
*/

```

```
/*  
 * TDOCSLabelTitle  
 *  
 * Formats title for label.  
 *  
 * Called by TDOCSLabelDocuments.  
 *  
 * Calls nothing.  
 */
```

```

/*
 * tdocsnms.pc
 *
 * This server module implements TDOCS names.
 *
 * 07/21/94 - cjm
 * 12/12/94 - cjm - redid header fields & implemented scanning
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocsnms.pc
 * Author:
 * Revision:   2.24.0
 * Date:       95/11/13-11:01:40
 *-----
 */

/*
 * GetNames
 *
 * Gets list of CNWRA names. Writes label to pReturn data, and values to list.
 * Sets pReturn status and error.
 *
 * Called by rpc_N
 *
 * Calls PutListData, SQLNotFound, SQLError.
 */

```

```

/*
* tdocsout.pc
*
* This server module implements TDOCS output synchronization.
*
* 03/27/95 - cjm
* 05/16/95 - cjm - upgrade to Pro*C 2.0
* 07/10/95 - cjm - new sync design
* 07/20/95 - cjm - split in and out
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocsout.pc
* Author:
* Revision:   1.5.0
* Date:       95/11/13-11:01:45
*-----
*/

/*
* TDOCSSyncOut
*
* Controls logic of synchronization output of individual documents of
* sharable documents sets:
*
*
*      local      | local | remote | remote
*      input      | batch | input  | batch
*      -----+-----+-----+-----
*      S          | L     | -       | -
*      Ss         | Ls    | S       | L
*      L/U        | L     | -       | -
*      L/Us       | Ls    | S       | L
*      Ls/U       | -     | -       | -
*      Ls/Us      | Ls    | U       | L
*      L/D        | X     | -       | -
*      L/Ds       | X     | -       | -
*      Ls/D       | X     | -       | -
*      Ls/Ds      | X     | D       | X
*
*
* Returns SVC_SUCCESS or SVC_OP_ERROR (SVC_SYNC_OUT_ERROR).
*
* Called by TDOCSSubmitDocument, TDOCSSyncUpdateDocument, TDOCSSyncDeleteDocument.
*
* Calls TDOCSSyncOutSubmit, TDOCSSyncOutUpdate, TDOCSSyncOutDelete.
*/

```



```

/*
 * TDOCSSyncOutNameExt
 *
 * Computes sync out file name and transaction extension appended to filenames.
 * Must be done to avoid overwriting transactions before the remote site grabs
 * them. Returns SVC_SUCCESS or SVC_OP_ERROR.
 *
 * Called by TDOCSSyncOutSubmit, TDOCSSyncOutUpdate, TDOCSSyncOutDelete.
 *
 * Calls nothing.
 */

/*
 * TDOCSSyncOutSubmit
 *
 * Writes header info to submit control file and copies files to document set's
 * outgoing synchronization directory. Returns SVC_SUCCESS or SVC_OP_ERROR
 * (SVC_SYNC_OUT_ERROR).
 *
 * Called by TDOCSSyncOut.
 *
 * Calls TDOCSSyncOutFiles, TDOCSSyncOutHeader.
 */

/*
 * TDOCSSyncOutUpdate
 *
 * Writes header info to update control file and copies files to document set's
 * outgoing synchronization directory. Returns SVC_SUCCESS or SVC_OP_ERROR
 * (SVC_SYNC_OUT_ERROR).
 *
 * Called by TDOCSSyncOut.
 *
 * Calls TDOCSSyncOutExt, TDOCSSyncOutFiles, TDOCSSyncOutHeader.
 */

/*
 * TDOCSSyncOutDelete
 *
 * Writes header info to delete control file and copies files to document set's
 * outgoing synchronization directory. Returns SVC_SUCCESS or SVC_OP_ERROR
 * (SVC_SYNC_OUT_ERROR).
 *
 * Called by TDOCSSyncOut.
 *
 * Calls TDOCSSyncOutExt, TDOCSSyncOutHeader.
 */

```

```

/*
 * TDOCSSyncOutFiles
 *
 * Copies document files from incoming or archive/images to document set's
 * outgoing synchronization directory. Returns SVC_SUCCESS or SVC_OP_ERROR
 * (SVC_SYNC_OUT_ERROR).
 *
 * Called by TDOCSSyncOutSubmit, TDOCSSyncOutUpdate.
 *
 * Calls nothing.
 */

/*
 * TDOCSSyncOutIncomingFiles
 *
 * Copies document files from incoming to document set's outgoing
 * synchronization directory. Returns SVC_SUCCESS or SVC_OP_ERROR
 * (SVC_SYNC_OUT_ERROR).
 *
 * Called by TDOCSSyncOutFiles.
 *
 * Calls nothing.
 */

/*
 * TDOCSSyncOutPreviousFiles
 *
 * Copies document files from archive/images to document set's
 * outgoing synchronization directory. Returns SVC_SUCCESS or SVC_OP_ERROR
 * (SVC_SYNC_OUT_ERROR).
 *
 * Called by TDOCSSyncOutFiles.
 *
 * Calls nothing.
 */

/*
 * TDOCSSyncOutHeader
 *
 * Writes header info based on request to temporary file and then moves to a
 * synchronization control file. Move is done to signal data is ready for
 * synchronization. Returns SVC_SUCCESS or SVC_OP_ERROR (SVC_SYNC_OUT_ERROR).
 *
 * Called by TDOCSSyncOutSubmit, TDOCSSyncOutUpdate, TDOCSSyncOutUpdate.
 *
 * Calls SLLHead, SLLNext.
 */

```

```

/*
 * tdocsrec.pc
 *
 * This server module implements TDOCS record handling.
 *
 * 12/09/94 - cjm
 * 12/12/94 - cjm - redid header fields & implemented scanning
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 * -----
 * Archive:      /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocsrec.pc
 * Author:
 * Revision:     1.23.0
 * Date:         95/11/13-11:01:47
 * -----
 */

/*
 * TDOCSGetRecord
 *
 * Gets a tdocs record by document number and formats it for return as
 * [<L>?<V>[;<V>]*]+ where L is a label and V is a value and <V>[;<V>] is a
 * list of values separated by ;s.
 *
 * Called by rpc_N
 *
 * Calls TDOCSDocSetFromTitle, GetData, TDOCSRetrieveData, TDOCSFormatData,
 * TDOCSFreeAssociatedData.
 */

/*
 * TDOCSInsertData
 *
 * Inserts document data into the tdocs table. Returns SVC_SUCCESS or
 * SVC_SQL_ERROR.
 *
 * Called by TDOCSSubmitDocument, TDOCSUpdateDocument, TDOCSSyncInInsert,
 * TDOCSSyncInUpdate.
 *
 * Calls SQLError.
 */

/*
 * TDOCSInsertAssocData
 *
 * Inserts associated into associated tables. Returns SVC_SUCCESS or
 * SVC_SQL_ERROR.

```

```

*
* Called by TDOCSSubmitDocument, TDOCSUpdateDocument, TDOCSSyncInInsert,
* TDOCSSyncInUpdate.
*
* Calls SLLHead, SLLNext, TDOCSFreeAssociatedData, SQLError.
*/

/*
* TDOCSDeleteAssocData
*
* Deletes associated tdocs data for an update. Returns SVC_SUCCESS,
* SVC_OP_ERROR, or SVC_SQL_ERROR.
*
* Called by TDOCSUpdateDocument.
*
* Calls SQLNotFound, SQLError.
*/

/*
* TDOCSUpdateData
*
* Updates tdocs record by document set and document id. Returns SVC_SUCCESS or
* SVC_SQL_ERROR.
*
* Called by TDOCSUpdateDocument, TDOCSSyncInUpdate.
*
* Calls nothing.
*/

/*
* TDOCSDeleteData
*
* "Deletes" record by marking for batch deletion. Sets pReturn status. Returns
* SVC_SUCCESS, SVC_OP_ERROR, or SVC_SQL_ERROR.
*
* Called by TDOCSDeleteDocument.
*
* Calls SQLNotFound, SQLError.
*/

/*
* TDOCSRetrieveData
*
* Gets record for document number from tdocs and associated tables. Returns
* SVC_SUCCESS, SVC_SQL_ERR, SVC_OP_ERROR (SVC_DATA_FIND_ERROR), or
* SVC_MEM_ERROR.
*
* Called by TDOCSGetRecord.

```

```

*
* Calls TDOCSGetAssociatedData, SQLNotFound, SQLError.
*/

/*
* TDOCSFormatData
*
* Formats tdocs data for return.
*
* Called by TDOCSGetRecord.
*
* Calls MemAlloc, TDOCSFormat, SLLHead, SLLNext.
*/

/*
* TDOCSFormat
*
* Formats a retrieved record for return. Returns 0 on success, 1 on failure.
*
* Called by TDOCSFormatData.
*
* Calls MemRealloc.
*/

/*
* TDOCSGetAssociatedData
*
* Fetches associated data for current document set and documentid into list.
* Returns SVC_SUCCESS, SVC_MEM_ERROR (end of data) or SVC_SQL_ERROR.
*
* Called by TDOCSRetrieveData, TDOCSFetchNewDocCursor.
*
* Calls SLLInitialize, MemAlloc, SLLAppend, TDOCSFreeAssociatedData,
* SQLError.
*/

```

```

/*
 * tdocsrpt.pc
 *
 * This server module implements TDOCS report generation.
 *
 * 04/03/94 - cjm
 * 12/12/94 - cjm - redid header fields & implemented scanning
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocsrpt.pc
 * Author:
 * Revision:   2.25.0
 * Date:       95/11/13-11:01:52
 *-----
 */

/*
 * TDOCSNewDocsReport
 *
 * Generates a report on new tdi documents entered from given date.
 *
 * Called by rpc_N
 *
 * Calls TDOCSDocSetFromTitle, TDOCSInitNewDocsReport, TDOCSOpenNewDocCursor,
 * TDOCSFetchNewDocCursor, TDOCSWriteNewDocsReport, TDOCSEndNewDocsReport,
 * ReadAsciiFile, TDOCSFreeAssociatedData.
 */

/*
 * TDOCSInitNewDocsReport
 *
 * Initializes new documents report. Returns SVC_SUCCESS, SVC_SQL_ERROR, or
 * SVC_MEM_ERROR.
 *
 * Called by TDOCSNewDocsReport.
 *
 * Calls GetData, RPTFileOpen, SQLError.
 */

/*
 * TDOCSWriteNewDocsReport
 *
 * Writes labeled data. Returns SVC_SUCCESS or SVC_MEM_ERROR.
 *
 * Called by TDOCSNewDocsReport.
 */

```

```

* Calls RPTPrtLine, RPTSetRecord, SLLHead, SLLNext, MemAlloc, MemRealloc,
* MemFree, RPTPrtRecord.
*/

/*
* TDOCSEndNewDocsReport
*
* Ends report with count of documents. Returns SVC_SUCCESS or SVC_MEM_ERROR.
*
* Called by TDOCSNewDocsReport.
*
* Calls RPTPrtLine, RPTFileClose.
*/

/*
* TDOCS CirculationReport
*
* Generates and returns a circulation report for the specified document set
* and possibly individual checkout name. Report includes document number,
* checkout date, and checkout name.
*
* Called by rpc_N
*
* Calls TDOCSDocSetFromTitle, RPTFileOpen, TDOCSOpenCirculationCursor,
* TDOCSFetchCirculationCursor, RPTPrtLine, RPTSetRecord, RPTPrtRecord,
* RPTFileClose, ReadAsciiFile, SQLError.
*/

/*
* TDOCSOpenNewDocCursor
*
* Opens cursor on tdocs for new document report. Returns SVC_SUCCESS or
* SVC_SQL_ERROR.
*
* Called by TDOCSNewDocsReport.
*
* Calls SQLError.
*/

/*
* TDOCSFetchNewDocCursor
*
* Fetches data from cursor on tdocs along with associated data for new
* document report. Returns SVC_SUCCESS, SVC_NO_MORE_RECORDS (end of data) or
* SVC_SQL_ERROR.
*
* Called by TDOCSNewDocsReport.
*

```

```

* Calls TDOCSFreeAssociatedData, TDOCSGetAssociatedData, SQLError.
*/

/*
* TDOCSOpenCirculationCursor
*
* Opens cursor on tdi or csp circulation for circulation report. Returns
* SVC_SUCCESS or SVC_SQL_ERROR.
*
* Called by TDOCS_CirculationReport.
*
* Calls TDOCSDocSetFromTitle, GetData, SQLError.
*/

/*
* TDOCSFetchCirculationCursor
*
* Fetches data from cursor on tdi or csp circulation for circulation report.
* Returns SVC_SUCCESS, SVC_NO_MORE_RECORDS (end of data) or SVC_SQL_ERROR.
*
* Called by TDOCS_CirculationReport.
*
* Calls SQLError.
*/

/*
* TDOCSGetRefRptList
*
* Returns list of tdocs reference reports. Sets pReturn status and error.
*
* Called by rpc_N
*
* Calls PutListData, Commit, SQLError.
*/

/*
* TDOCSReferenceReport
*
* Generates a reference report.
*
* Called by rpc_N
*
* Calls TDOCSHydroRefReport, TDOCSNISTRefReport, Rollback, SQLError.
*/

/*
* TDOCSStatisticsReport
*

```


- * Retrieves a statistics report.
- *
- * Called by rpc_N
- *
- * Calls ReadAsciiFile.
- */

```

/*
* tdocssub.pc
*
* This server module implements TDOCS document submission.
*
* 12/09/94 - cjm
* 12/12/94 - cjm - redid header fields & implemented scanning
* 02/08/95 - cjm - solaris port
* 05/16/95 - cjm - upgrade to Pro*C 2.0
* 10/26/95 - sv - modified TDOCSGetSubmitData to get unique letter from
*                  documentset_id
*-----
* Archive: /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocssub.pc
* Author:
* Revision: 1.9.0
* Date: 95/11/13-11:01:55
*-----
*/

```

```

/*
* TDOCSSubmitDocument
*
* Submits a tdocs document as follows:
*
* -) Sets internal data like document id, etc., and gets client supplied
*    data like document date, author, title, etc., and performs error
*    checking, returning bad data.
* -) Gets any incoming files.
* -) Inserts the main record.
* -) Inserts associated data.
* -) Returns status in pReturn.
*
* Note that the document is not loaded into Topic - done in batch.
*
* Called by rpc_N
*
* Calls TDOCSGetSubmitData, TDOCSInsertData, TDOCSInsertAssocData,
* TDOCSSyncOut, Commit, MemAlloc.
*/

```

```
/*  
* TDOCSGetSubmitData  
*  
* Sets, gets, and checks tdocs submit data. Returns SVC_SUCCESS,  
*  
* Called by TDOCSSubmitDocument.  
*  
* Calls TDOCSGetGeneralData, TDOCSGetDocumentData, TDOCSCheckData,  
* TDOCSGetIncomingFiles, SetDataError, SQLError.  
*/
```

```

/*
* tdocsupd.pc
*
* This server module implements TDOCS document updates.
*
* 12/09/94 - cjm
* 12/12/94 - cjm - redid header fields & implemented scanning
* 02/08/95 - cjm - solaris port
* 05/16/95 - cjm - upgrade to Pro*C 2.0
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocsupd.pc
* Author:
* Revision:   1.9.0
* Date:       95/11/13-11:02:07
*-----
*/

```

```

/*
* TDOCSUpdateDocument
*
* Updates a tdocs document by replacement as follows:
*
* -) Gets client supplied data like document date, author, title, etc., and
*    sets internal data like document id, etc., and performs error checking,
*    returning bad data. And gets any incoming files.
* -) Deletes old associated data.
* -) Updates old record.
* -) Inserts new associated data.
* -) Places transaction in output queue for synchronization.
* -) Returns status in pReturn.
*
* Note the following cases when an update request occurs:
*
*
*      oldstat  oldsrc  newfil  newstat  newsrc  filact
*
*      SorU     0/0     N       SorU     0/0     -
*      SorU     0/N     N       SorU     0/N     -
*      SorU     0/0     Y       SorU     0/N     copy
*      SorU     0/N     Y       SorU     0/N     replace
*
*      L        0/0     N       U        0/0     -
*      L        0/N     N       U        N/0     -
*      L        0/0     Y       U        0/N     copy
*      L        0/N     Y       U        N/N     copy
*
* Called by rpc_N
*

```

```

* Calls TDOCSGetUpdateData, TDOCSDeleteAssocData, TDOCSUpdateData,
* TDOCSInsertAssocData, TDOCSSyncOut, Commit.
*/

/*
* TDOCSGetUpdateData
*
* Sets, gets, and checks tdocs update data. Returns SVC_SUCCESS,
*
* Called by TDOCSUpdateDocument.
*
* Calls TDOCSGetDocumentData, TDOCSGetGeneralData, TDOCSCheckData,
* TDOCSCheckData, TDOCSGetIncomingFiles, SetDataError, SQLNotFound, SQLError.
*/

```

```

/*
 * usr.pc
 *
 * This module implements functions for information on and handling of users.
 *
 * 02/24/94 - cjm
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:      %P%
 * Author:
 * Revision:     %R%.%L%.%B%
 * Date:         %E%- %U%
 *-----
 */

/*
 * GetUserInfo
 *
 * Gets privileges and name for userid found in pPacket->data. Also checks
 * version number. Expects "<label>\2<userid>\1<label>\2<version>\2" (without
 * brackets). Writes results to pReturn data. Sets pReturn status and error.
 *
 * Called by rpc_N
 *
 * Calls MemAlloc, Commit, SQLError.
 */

/*
 * ChangePassword
 *
 * Changes user password. Expects pPacket->data=
 * "USER_ID\2userid\1PASSWD\2passwd\1". Sets pReturn status and error.
 *
 * Called by rpc_N
 *
 * Calls GetData, Commit, SQLError.
 */

/*
 * AddUser
 *
 * Adds user as defined by userid, password, privilege, and name. Expects
 * pPacket->data="USER_ID\2userid\1PASSWD\2passwd\1...
 * PRIVILEGE\2privilege\1NAME\2name\1". Sets pReturn status and error.
 *
 * Called by rpc_N

```

```

*
* Calls GetData, Commit, SQLError.
*/

/*
* DropUser
*
* Drops user as defined by userid. Expects pPacket->data="USER_ID\2userid\1".
* Sets pReturn status and error.
*
* Called by rpc_N
*
* Calls GetData, Commit, SQLError.
*/

/*
* GetUserNames
*
* Gets list of user names. Writes label to pReturn data, and values to list.
* Sets pReturn status and error.
*
* Called by rpc_N
*
* Calls PutListData, SQLNotFound, SQLError.
*/

/*
* GetUserList
*
* Gets list of userid, privilege, and name. Writes labels to pReturn data, and
* values to list. Sets pReturn status and error.
*
* Called by rpc_N
*
* Calls PutListData, SQLNotFound, SQLError.
*/

```

8.2.2 Batch C

```
/*
* batchdb.pc
*
* This module defines common database host variables, handles parsing data
* passed to the server and general database stuff like log in, log out,
* commit, rollback. Also declares all shared oracle host variables.
*
* 05/05/94 - cjm
* 02/08/95 - cjm - solaris port
* 02/20/95 - cjm - removed reliance on tcp for db access
* 05/16/95 - cjm - upgrade to Pro*C 2.0
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.batchdb.pc
* Author:
* Revision:   2.17.0
* Date:       96/01/03-13:33:07
*-----
*/

/*
* LogIn
*
* Logs in dba using uid/pwd@instance for access.
*
* Return SUCCESS or FAILURE.
*
* Called by main.
*
* Calls nothing.
*/

/*
* LogIn
*
* Logs in dba using uid/pwd@instance for access.
*
* Return SUCCESS or FAILURE.
*
* Called by main.
*
* Calls nothing.
*/

/*
```



```

* LogOut
*
* Logout dba.
*
* Called by main.
*
* Calls nothing.
*/

/*
* Commit
*
* Commits oracle transactions. Returns SUCCESS or FAILURE.
*
* Called by RPDGetBatchRecords, RPDUpdateCheckinRecords,
* RPDUpdateDefineRecords, RPDUpdateRetireRecords, RPDeleteRetirePieces,
* RPDBatchReportWriter, TDOCSUpdateDocuments, TDOCSSubmitDocuments,
* TDOCSDeleteDocuments.
*
* Calls WriteBatchReport2.
*/

/*
* Rollback
*
* Rolls back oracle transactions. Returns FAILURE.
*
* Called by RPDGetBatchRecords, RPDUpdateCheckinRecords,
* RPDUpdateDefineRecords, RPDUpdateRetireRecords, RPDeleteRetirePieces,
* RPDBatchReportWriter, TDOCSOpenBatchSets, TDOCSFetchBatchSet,
* TDOCSUpdateDocuments, TDOCSSubmitDocuments, TDOCSDeleteDocuments,
* TDOCSUpdateSource, TDOCSBatchUpdateStatus, TDOCSOpenBatchCursor,
* TDOCSFetchBatchRecord, TDOCSGetAssociatedData, TDOCSDeleteRecord.
*
* Calls WriteBatchReport2.
*/

/*
* GetDateExtension
*
* Gets current date from oracle and point date to it.
*
* Return SUCCESS or FAILURE.
*
* Called by main.
*
* Calls nothing.
*/

```

```

/*
 * rpdbat.pc
 *
 * Implementation of rpd batch main function, shared functions.
 *
 * 05/05/94 - cjm
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.rpdbat.pc
 * Author:
 * Revision:   1.16.0
 * Date:       96/01/03-13:33:15
 *-----
 */

/*
 * RPDBatch
 *
 * Batch processes rpd document defines and checkins and generates batch report.
 * If any records are successfully processed, doReports id returned nonzero,
 * otherwise zero.
 *
 * Called by main.
 *
 * Calls OpenBatchReport, WriteBatchReport3, WriteBatchReportEnv,
 * RPDBatchDefineDocuments, FlushBatchReport, RPDBatchCheckinDocuments,
 * RPDBatchRetireDocuments, RPDCleanUp, RPDBatchReportWriter, CloseBatchReport.
 */

/*
 * RPDRunWPTG
 *
 * Runs wp or tg command line with system call and monitors its progress,
 * killing it if it terminates. IMPORTANT: the command line must background the
 * process! Returns 0 on success, 1 on failure.
 *
 * Called by RPDMacroStyler, RPDIntellitag.
 *
 * Calls nothing.
 */

```

```

/*
 * RPDPieceDocument
 *
 * Call piece to piece document into piece table. Returns SUCCESS or FAILURE.
 *
 * Called by RPDPieces.
 *
 * Calls WriteBatchReport2, WriteBatchReport1, WriteBatchReportRPDRecord.
 */

/*
 * RPDWriteHeader
 *
 * Writes header for document with all control info. Returns SUCCESS or FAILURE.
 *
 * Called by RPDBatchCheckinDocuments, RPDBatchDefineDocuments.
 *
 * Calls WriteBatchReport2, WriteBatchReportRPDRecord.
 */

/*
 * RPDBuildLoadDoc
 *
 * Concatenates load document as header + ascii. Returns SUCCESS or FAILURE.
 *
 * Called by RPDBatchCheckinDocuments, RPDBatchDefineDocuments.
 *
 * Calls WriteBatchReport2, WriteBatchReportRPDRecord.
 */

/*
 * RPDWriteFileList
 *
 * Writes file name to file list. Always appends since InitLoad deleted.
 * Returns SUCCESS or FAILURE.
 *
 * Called by RPDBatchCheckinDocuments, RPDBatchDefineDocuments,
 * RPDBatchRetireDocuments.
 *
 * Calls WriteBatchReport2, WriteBatchReportRPDRecord.
 */

```

```

/*
 * RPDStoreDocument
 *
 * Writes header for document with all control info. Returns SUCCESS or FAILURE.
 *
 * Called by RPDBatchCheckinDocuments, RPDBatchDefineDocuments.
 *
 * Calls WriteBatchReport2, WriteBatchReportRPDRecord.
 */

```

```

/*
 * RPDLoadDocuments
 *
 * Loads partition by creation or insertion. Then adds links to launch WP and
 * secures documents. Returns SUCCESS or FAILURE.
 *
 * Called by RPDBatchCheckinDocuments, RPDBatchDefineDocuments,
 * RPDBatchRetireDocuments.
 *
 * Calls WriteBatchReport1, RPDIndexDocuments, RPDLinkDocuments,
 * RPDSecureDocuments.
 */

```

```

/*
 * RPDIndexDocuments
 *
 * Indexes documents in partition. Returns SUCCESS or FAILURE.
 *
 * Called by RPDLoadDocuments.
 *
 * Calls WriteBatchReport3.
 */

```

```

/*
 * RPDLinkDocuments
 *
 * Links indexed documents with WP documents.
 *
 * Called by RPDLoadDocuments.
 *
 * Calls WriteBatchReport3.
 */

```

```

/*
 * RPDSecureDocuments
 *
 * Secures indexed documents. Returns SUCCESS or FAILURE.
 *
 * Called by RPDLoadDocuments.
 *
 * Calls WriteBatchReport3.
 */

/*
 * RPDCleanUp
 *
 * Cleans up by removing temporary files to backup. Called CleanUp 'cause
 * that's what it used to do, simply delete.
 *
 * Called by RPDBatch.
 *
 * Calls nothing.
 */

/*
 * RPDInitLoad
 *
 * Set firstDoc, a flag that determines how files are loaded into the partition.
 * Returns SUCCESS or FAILURE.
 *
 * Called by RPDBatchCheckinDocuments, RPDBatchDefineDocuments,
 * RPDBatchRetireDocuments.
 *
 * Calls WriteBatchReport2.
 */

/*
 * RPDGetBatchRecords
 *
 * Retrieves list of report writer batch records whose status is status
 * (DEFINE, CHECKIN, or RETIRE). Returns SUCCESS or FAILURE.
 *
 * Called by RPDBatchCheckinDocuments, RPDBatchDefineDocuments,
 * RPDBatchRetireDocuments.
 *
 * Calls SLLInitialize, SLLAppend, Commit, Rollback, RPDFreeBatchRecords,
 * WriteBatchReport1, WriteBatchReport2.
 */

```

```
/*  
 * RPDSummaryReport  
 *  
 * Lists good and bad records.  
 *  
 * Called by RPDBatchCheckinDocuments, RPDBatchDefineDocuments,  
 * RPDBatchRetireDocuments.  
 *  
 * Calls WriteBatchReport2, WriteBatchReport1, SLLHead, SLLNext.  
 */
```

```
/*  
 * RPDFreeBatchRecords  
 *  
 * Frees report writer records.  
 *  
 * Called by RPDBatchCheckinDocuments, RPDBatchDefineDocuments,  
 * RPDBatchRetireDocuments, RPDGetBatchRecords.  
 *  
 * Calls SLLDelete.  
 */
```

```

/*
* rpdchk.pc
*
* Implementation of rpd checkin batch.
*
* 05/18/94 - cjm
* 02/08/95 - cjm - solaris port
* 05/16/95 - cjm - upgrade to Pro*C 2.0
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.rpdchk.pc
* Author:
* Revision:   1.16.0
* Date:      96/01/03-13:33:17
*-----
*/

```

```

/*
* RPDBatchCheckinDocuments
*
* Batch processes rpd document checkins and generates batch report.
*
* -) First, fetches all records in docset marked for checkin into a list.
* -) Second, performs report writer functions:
*   -) For each record in list, runs copy of file against the macro styler to
*       generate styled wp documents.
*   -) Runs intellitag on styled wp documents to generate sgml instances.
*   -) For each record in list, runs pieces on sgml instances to put pieces in
*       database.
* -) Third, performs topic loading functions:
*   -) For each record in list
*     -) writes header information to an info file
*     -) concatenates info and ascii files as a load document
*     -) appends file name to checkin load file list
*     -) stores checkin WP document in archive
*     -) appends file name to archive load file list
*     -) stores archive WP document in archive
*     -) cleans up
*     -) fetches next record
*   -) Batch loads, links and secures checkin files into topic.
*   -) Batch loads, links and secures archive files into topic.
* -) Updates document status from checkin to active.
*
* If any records are successfully processed, reports are flagged. Returns
* SUCCESS or FAILURE.
*
* Called by RPDBatch.
*

```

- * Calls WriteBatchReport2, WriteBatchReport1, RPDGetBatchRecords,
- * RPDMacroStyler, FlushBatchReport, RPDIntellitag, RPDUpdateCheckinRecords,
- * RPDFreeBatchRecords, RPDPieces, RPDInitLoad, SLLHead, SLLNext,
- * RPDWriteHeader, RPDBuildLoadDoc, RPDWriteFileList, RPDStoreDocument,
- * RPDDeleteArchive, RPDMoveArchive, RPDLoadDocuments, RPDUpdateCheckinRecords,
- * RPDSummaryReport, RPDFreeBatchRecords.

*/

/*

- * RPDDeleteArchive

*

- * Deletes previous version from active topic partition. Returns SUCCESS or
- * FAILURE.

*

- * Called by RPDBatchCheckinDocuments, RPDBatchRetireDocuments.

*

- * Calls WriteBatchReport2, WriteBatchReportRPDRecord.

*/

/*

- * RPDMoveArchive

*

- * Moves previous version from active to archive topic partition. Returns
- * SUCCESS or FAILURE.

*

- * Called by RPDBatchCheckinDocuments, RPDBatchRetireDocuments.

*

- * Calls WriteBatchReport2, WriteBatchReportRPDRecord.

*/

/*

- * RPDUpdateCheckinRecords

*

- * Updates checkin records: if ok, sets status to ACTIVE and previous record's
- * status to archived, if not, it deletes the record. Returns SUCCESS or
- * FAILURE.

*

- * Called by RPDBatchCheckinDocuments.

*

- * Calls SLLHead, SLLNext, Commit, WriteBatchReport2, Rollback.

*/

/*

- * This code will move all the *.bat.* file to *.hdr.* in the outgoing
- * directory for the remote server to pick it up for syncin.

*/


```

/*
 * rpddef.pc
 *
 * Implementation of rpd define batch.
 *
 * 05/10/94 - cjm
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpd docs/development/batch/SCCS/s.rpddef.pc
 * Author:
 * Revision:   1.15.0
 * Date:      96/01/03-13:33:18
 *-----
 */

/*
 * RPDBatchDefineDocuments
 *
 * Batch processes rpd document defines and generates batch report.
 *
 * -) First, fetches all records in docset marked for define into a list.
 * -) While there is a record:
 *   -) writes vacant ascii, wp and sgml files to incoming
 *   -) pieces the sgml file into the piece table
 *   -) writes header information to an info file
 *   -) concatenates info and ascii files as a load document
 *   -) appends file name to load file list
 *   -) stores WP document in archive
 *   -) cleans up
 *   -) fetches next record
 * -) Batch loads, links and secures files into topic.
 * -) Updates document status from define to vacant.
 *
 * If any records are successfully processed, reports are flagged. Returns
 * SUCCESS or FAILURE.
 *
 * Called by RPDBatch.
 *
 * Calls WriteBatchReport2, RPDGetBatchRecords, WriteBatchReport1,
 * RPDInitLoad, SLLHead, RPDWriteVacant, RPDPieceDocument, RPDWriteHeader,
 * RPDBuildLoadDoc, RPDWriteFileList, RPDStoreDocument, SLLNext,
 * RPDLoadDocuments, RPDUpdateDefineRecords, RPDSummaryReport,
 * RPDFreeBatchRecords.
 */

```

```

/*
 * RPDWriteVacant
 *
 * Creates vacant ascii, wp and sgml document with document
 * number and title. Returns SUCCESS or FAILURE.
 *
 * Called by RPDBatchDefineDocuments.
 *
 * Calls WriteBatchReport2, WriteBatchReportRPDRecord.
 */

/*
 * RPDUpdateDefineRecords
 *
 * Updates define records: if ok, sets status to VACANT, if not, deletes record.
 * Returns SUCCESS or FAILURE.
 *
 * Called by RPDBatchDefineDocuments.
 *
 * Calls SLLHead, SLLNext, Commit, WriteBatchReport2, Rollback.
 */

```

```

/*
 * rpdret.pc
 *
 * Implementation of batch rpd retire.
 *
 * 06/26/94 - cjm
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.rpdret.pc
 * Author:
 * Revision:   1.16.0
 * Date:       96/01/03-13:33:19
 *-----
 */

/*
 * RPDRetire
 *
 * Batch processes rpd document retires and generates batch report.
 *
 * -) Fetches all records in docset marked for retire into a list.
 * -) Then, performs topic loading functions:
 *   -) For each record in list
 *     -) appends file name to archive load file list
 *     -) stores archive WP document in archive
 *     -) cleans up
 *     -) fetches next record
 *   -) Batch loads, links and secures archive files into topic.
 * -) Finally, updates most recent record's status and partition and delete
 *    pieces.
 *
 * If any records are successfully process, reports are flagged.
 *
 * Called by RPDBatch.
 *
 * Calls WriteBatchReport2, RPDGetBatchRecords, RPDGetBatchRecords,
 * RPDInitLoad, SLLHead, RPDDelateArchive, RPDWriteFileList, RPDMoveArchive,
 * SLLNext, WriteBatchReport1, RPDLoadDocuments, RPDUpdateRetireRecords,
 * RPDDelateRetirePieces, FlushBatchReport, RPDSummaryReport,
 * RPDFreeBatchRecords.
 */

```

```

/*
 * RPDUpdateRetireRecords
 *
 * Updates retire records: if ok, sets status to RETIRED and previous record's
 * status to ARCHIVED, if not, it deletes the record. Returns SUCCESS or
 * FAILURE.
 *
 * Called by RPDBatchRetireDocuments.
 *
 * Calls SLLHead, SLLNext, WriteBatchReport2, Commit, Rollback.
 */

```

```

/*
 * RPDDeleteRetirePieces
 *
 * For each okay record, deletes it associated pieces in the piece table.
 * Returns SUCCESS or FAILURE.
 *
 * Called by RPDRetireDocuments.
 *
 * Calls WriteBatchReport2, SLLHead, SLLNext, Commit, WriteBatchReport1,
 * Rollback.
 */

```

```
/*
 * rpdrw.pc
 *
 * Implementation of rpd batch report writer.
 *
 * 06/26/94 - cjm
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.rpdrw.pc
 * Author:
 * Revision:   1.15.0
 * Date:       96/01/03-13:33:20
 *-----
 */
```

```

/*
* RPDPieces
*
* For each file in report writer record list, calls pieces. Macro styler
* generated for each file, fn, a styled file, fn.wp. Intellitag generated for
* each fn.wp a fn.wp.sgm. These are fed to pieces to put document pieces into
* the database.
*
* Called by RPDBatchCheckinDocuments.
*
* Calls WriteBatchReport1, SLLHead, SLLNext, RPDPieceDocument.
*/

/*
* RPDMacroStyler
*
* For each file in report writer record list, copies file to file.wp, gets its
* time stamp, runs the macro styler against it, gets the time stamp for the
* output file, compares time stamps and if different assumes success, and if
* not assumes error. This process generates for each checkin file, fn, a
* styled file, fn.wp. Note, too, that there are different sets of macros for
* test1 and prod1 databases. Test1 macros are prefixed with 't'.
*
* Called by RPDBatchCheckinDocuments.
*
* Calls WriteBatchReport1, SLLHead, SLLNext, WriteBatchReportRPDRecord,
* RPDRunWPTG.
*/

/*
* RPDIntellitag
*
* For all files in the incoming directory, batch them through intellitag.
* Macro styler generated for each file, fn, a styled file, fn.wp. Intellitag
* generates for each fn.wp a fn.wp.sgm. These are fed to pieces. Returns
* SUCCESS or FAILURE.
*
* Called by RPDBatchCheckinDocuments.
*
* Calls WriteBatchReport1, SLLHead, RPDRunWPTG, SLLNext, RPDIntellitagLog.
*/

```

```

/*
 * RPDIntellitagLog
 *
 * Checks intellitag log file for errors and reports them. Returns SUCCESS or
 * FAILURE.
 *
 * Called by RPDIntellitag.
 *
 * Calls SLLHead, WriteBatchReport1, SLLNext, WriteBatchReport2,
 * WriteBatchReportRPDRecord.
 */

/*
 * Report Writer Backend
 */

/*
 * RPDRWWriter
 *
 * Runs the RPD report writer utility to generate an SGML report. Returns
 * SUCCESS or FAILURE.
 *
 * Called by RPDBatchReportWriter.
 *
 * Calls WriteBatchReport2, WriteBatchReport3.
 */

/*
 * RPDRWWPChar
 *
 * Run the RPD report writer wpchar utility to convert some SGML to WP
 * characters. Returns SUCCESS or FAILURE.
 *
 * Called by RPDBatchReportWriter.
 *
 * Calls WriteBatchReport2.
 */

/*
 * RPDRWBackendMacro
 *
 * Runs the RPD report writer macro in WP to convert the SGML report to WP
 * format. Returns SUCCESS or FAILURE.
 *
 * Called by RPDBatchReportWriter.
 *
 * Calls WriteBatchReport1, WriteBatchReport2.
 */

```

```

/*
 * RPDRWStore
 *
 * Stores report to reports/writer/<report>.wp. Returns SUCCESS or FAILURE.
 *
 * Called by RPDBatchReportWriter.
 *
 * WriteBatchReport2.
 */

/*
 * RPDBatchReportWriter
 *
 * Retrieves and batches list of report writer reports and runs report writer
 * on current report (h_rwreport):
 *
 * -) calls writer
 * -) calls wpchar
 * -) calls backend macro processor
 * -) moves file to reports/writer/<report>.wp.
 *
 * Called by RPDBatch.
 *
 * Calls WriteBatchReport1, WriteBatchReport2, RPDRWWriter, RPDRWWPChar,
 * RPDRWBackendMacro, RPDRWStore, Commit, Rollback.
 */

/*
 * RPDBatchDocuments
 *
 * Generates all documents in dataSet from pieces.
 */

/*
 * RPDGenerateDocument
 *
 * Generates a document in a document set.
 */

```



```

/*
 * tdocsbat.pc
 *
 * Implementation of tdocs batch control.
 *
 * 05/05/94 - cjm
 * 12/14/94 - cjm - generalized header fields and document sets
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.tdocsbat.pc
 * Author:
 * Revision:   1.18.0
 * Date:       96/01/03-13:33:28
 *-----
 */

```

```

/*
 * TDOCSBatch
 *
 * Batch processes tdocs document sets. For each loadable document set
 *
 * -) fetches set name and specification
 * -) updates records marked for update
 * -) submits records marked for submission
 * -) deletes records marked for deletion
 *
 * Order is important.
 *
 * Called by main.
 *
 * Calls OpenBatchReport, WriteBatchReport3, WriteBatchReportEnv,
 * CloseBatchReport, TDOCSOpenBatchSets, TDOCSFetchBatchSets,
 * TDOCSUpdateDocuments, TDOCSSubmitDocuments, TDOCSDeleteDocuments,
 * TDOCSBatchReports.
 */

```

```

/*
 * TDOCSOpenBatchSets
 *
 * Opens cursor on document sets to be batched and fetches them into a
 * list the first time. Each time it sets h_documentset. Returns SUCCESS or
 * FAILURE.
 *
 * Called by TDOCSBatch.
 *
 * Calls WriteBatchReport2, Rollback.
 */

```

```

/*
 * TDOCSFetchBatchSet
 *
 * For each doc set fetched above, get specification and other data. Returns
 * SUCCESS or ENDLOOP.
 *
 * Called by TDOCSBatch.
 *
 * Calls SLLHead, SLLNext, SLLDelete, SLLFree, WriteBatchReport2, Rollback.
 */

```

```

/*
 * TDOCSSetBatchControl
 *
 * Controls logic of batch update, submit, delete.
 *
 * Whether it's time to load a partition (at DOCS_PER_PARTITON
 * docs waitonload is set to 0).
 *
 * Whether load should insert documents (previous partition with
 * less than DOCS_PER_PARTITION docs) or create a new partition
 * (newPartition set to 1).
 *
 * The name of the partiton to insert into or create (in
 * subpartnumber).
 *
 * Whether the document is the first of a load request
 * (tdocsFirstDocument set to 1).
 *
 * Document id of the first document of a partition (in
 * h_firstpartdoc).
 *
 * Creates partition as needed.
 *
 * Returns SUCCESS or FAILURE.
 *
 * Called by TDOCSUpdateDocuments, TDOCSSubmitDocuments,
 * TDOCSDeleteDocuments.
 *
 * Calls nothing.
 */

```

```

/*
 * TDOCSCheckLoadBoundary
 *
 * Checks whether current docid <+ mult 1024 < next docid is true. Call after
 * fetch to get next document id.
 *
 * Called by TDOCSSubmitDocuments.
 *
 * Calls nothing.
 */

/*
 * TDOCSUpdateDocuments
 *
 * Updates tdocs document set records marked for update:
 *
 * -) opens cursor on tdocs document set records marked for update
 *    and for each fetched record
 * -) unloads its document from topic
 * -) restores files to incoming for submit
 * -) removes update bit from source
 * -) updates status of all update records to submit status
 * -) dumps batch report
 *
 * In effect, the submit function is called next to put updates into topic.
 * RPC results are set in pReturn.
 *
 * Called by TDOCSBatch.
 *
 * Calls WriteBatchReport2, TDOCSOpenBatchCursor, TDOCSFetchBatchRecord,
 * TDOCSSetBatchControl, TDOCSUnloadDocument, TDOCSUpdateFiles,
 * TDOCSUpdateSource, TDOCSBatchUpdateStatus, Commit, Rollback, DumpBatchReport.
 */

/*
 * TDOCSSubmitDocuments
 *
 * Submits tdocs records from Oracle to Topic.
 *
 * -) sets all records marked input to submit (labeling does
 *    this but some document sets are never labeled)
 * -) opens cursor on all tdocs records marked for submission
 *    loaded and fetches first record.
 * -) for each record fetched:
 * -) sets partition and document information, possibly
 *    creating the partition.
 * -) copies files from incoming to partition subdirectory
 * -) writes load and link files and appends file name to file list.

```

```

* -) if partition is full, loads prepared documents into
*   Topic and updates those records' status.
* -) when done generating load files and file lists, loads Topic
*   if any prepared documents remain and updates those
*   records' status
* -) if successfull, commits transactions and cleans up incoming
*   files, otherwise rolls them back
*
* Called by TDOCSBatch.
*
* Calls WriteBatchReport2, TDOCSBatchUpdateStatus, TDOCSOpenBatchCursor,
* TDOCSFetchBatchRecord, TDOCSSetBatchControl, TDOCSSubmitFiles,
* TDOCSWriteLoadLinkFiles, TDOCSWriteLoadFileList, TDOCSCheckLoadBoundary,
* TDOCSLoadPartition, TDOCSBatchUpdateStatus, TDOCSCleanUpIncoming,
* Commit, Rollback, DumpBatchReport.
*/

/*
* TDOCSDeleteDocuments
*
* Deletes records marked for deletion from tdocs.
*
* -) for each record marked for deletion
*   -) unloads it from topic
*   -) deletes its files
*   -) updates delete records to deleted
*
* Called by TDOCSBatch.
*
* Calls WriteBatchReport2, TDOCSOpenBatchCursor, TDOCSFetchBatchRecord,
* TDOCSSetBatchControl, TDOCSUnloadDocument, TDOCSDeleteFiles,
* TDOCSDeleteRecord, Commit, Rollback, DumpBatchReport.
*/

```

```

/*
* tdocsdoc.pc
*
* Implementation of document handling for tdocs batch.
*
* 05/09/94 - cjm
* 12/14/94 - cjm - generalized header fields and document sets
* 02/08/95 - cjm - solaris port
* 05/16/95 - cjm - upgrade to Pro*C 2.0
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.tdocsdoc.pc
* Author:
* Revision:   1.18.0
* Date:       96/01/03-13:33:53
*-----
*/

/*
* TDOCSUpdateFiles
*
* For updating record, if it has associated files in incoming then do nothing;
* otherwise move archive and image files back to incoming so submit can find
* them. Returns SUCCESS because files may already have been updated but the
* process failed later.
*
* Called by TDOCSUpdateDocuments.
*
* Calls TDOCSRemoveLoadLinkFiles, TDOCSRestoreTxtFile, TDOCSRestoreBinFile,
* TDOCSRestoreImgFiles, TDOCSRemoveTxtFile, TDOCSRemoveBinFile,
* TDOCSRemoveImgFiles.
*/

/*
* TDOCSSubmitFiles
*
* For submitting record, if it has associated files, store them. Returns
* SUCCESS or FAILURE.
*
* Called by TDOCSSubmitDocuments.
*
* Calls TDOCSCopyTxtFile, TDOCSCopyBinFile, TDOCSCopyImgFilese
*/

```

```

/*
 * TDOCSDeleteFiles
 *
 * For deleting record, if it has associated files, delete them. Returns
 * SUCCESS because files may already have been updated but the process failed
 * later.
 *
 * Called by TDOCSDeleteDocuments.
 *
 * Calls TDOCSRemoveLoadLinkFiles, TDOCSRemoveTxtFile, TDOCSRemoveBinFile,
 * TDOCSRemoveImgFiles.
 */

/*
 * TDOCSRemoveTxtFile
 *
 * Removes a document's text file. Returns SUCCESS.
 *
 * Called by TDOCSUpdateFiles, TDOCSDeleteFiles.
 *
 * Calls WriteBatchReport1.
 */

/*
 * TDOCSRemoveBinFile
 *
 * Removes a document's binary file. Returns SUCCESS.
 *
 * Called by TDOCSUpdateFiles, TDOCSDeleteFiles.
 *
 * Calls WriteBatchReport1.
 */

/*
 * TDOCSRemoveImgFiles
 *
 * Removes a document's image files. Returns SUCCESS.
 *
 * Called by TDOCSUpdateFiles, TDOCSDeleteFiles.
 *
 * Calls WriteBatchReport1.
 */

```

```

/*
 * TDOCSRemoveLoadLinkFiles
 *
 * Removes a document's load and link files. Returns SUCCESS.
 *
 * Called by TDOCSUpdateFiles, TDOCSDeleteFiles.
 *
 * Calls WriteBatchReport1.
 */

```

```

/*
 * TDOCSRestoreTxtFile
 *
 * Restores a document's text file. Returns SUCCESS.
 *
 * Called by TDOCSUpdateFiles.
 *
 * Calls WriteBatchReport1.
 */

```

```

/*
 * TDOCSRestoreBinFile
 *
 * Restores a document's binary file. Returns SUCCESS.
 *
 * Called by TDOCSUpdateFiles.
 *
 * Calls WriteBatchReport1.
 */

```

```

/*
 * TDOCSRestoreImgFiles
 *
 * Restores a document's image file(s). Returns SUCCESS.
 *
 * Called by TDOCSUpdateFiles.
 *
 * Calls nothing.
 */

```

```

/*
 * TDOCSCopyTxtFile
 *
 * Copies text file to archive. Returns SUCCESS or FAILURE.
 *
 * Called by TDOCSSubmitFiles.
 *
 * Calls WriteBatchReport1.
 */

```

```

/*
 * TDOCSCopyBinFile
 *
 * Copies binary file to archive. Returns SUCCESS or FAILURE.
 *
 * Called by TDOCSSubmitFiles.
 *
 * Calls WriteBatchReport1.
 */

```

```

/*
 * TDOCSCopyImgFiles
 *
 * Copies image file(s) to image archive. Returns SUCCESS or FAILURE.
 *
 * Called by TDOCSSubmitFiles.
 *
 * Calls WriteBatchReport1.
 */

```

```

/*
 * TDOCSCreatePartDirs
 *
 * Create subpartition directories. Returns SUCCESS because directories may
 * already exist.
 *
 * Called by TDOCSSetBatchControl.
 *
 * Calls nothing.
 */

```

```

/*
 * TDOCSAppendCleanUp
 *
 * Collects document ids for cleanup of files in incoming.
 *
 * Called by None.
 *
 * Calls SLLInitialize, SLLAppend, WriteBatchReport1.
 */

```



```
/*  
 * TDOCSCleanUpIncoming  
 *  
 * Cleans out incoming document set files after successful loading.  
 *  
 * Called by TDOCSSubmitDocuments.  
 *  
 * Calls SLLHead, SLLNext, FreeDataList.  
 */
```

```

/*
* tdocshyd.pc
*
* This batch module implements the TDOCS hydrology db.
*
* 07/15/94 - cjm
* 12/12/94 - cjm - redid header fields & implemented scanning
* 02/08/95 - cjm - solaris port
* 05/16/95 - cjm - upgrade to Pro*C 2.0
* 06/01/95 - cjm - moved from server to batch
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.tdocshyd.pc
* Author:
* Revision:   1.8.0
* Date:       96/01/03-13:33:32
*-----
*/

```

```

/*
* TDOCSHydroRefReport
*
* Generates a report on hydrology references.
*
* Called by TDOCSBatchReports.
*
* Calls RPTFileOpen, TDOCSOpenHydroRefCursor, TDOCSFetchHydroRefCursor,
* RPTFormatLine, RPTPrLine, RPTFileClose, WriteBatchReport1,
* WriteBatchReport2, Commit, Rollback.
*/

```

```

/*
* TDOCSOpenHydroRefCursor
*
* Opens cursor on hydrology data for report. Returns SUCCESS or FAILURE.
*
* Called by TDOCSHydroRefReport.
*
* Calls WriteBatchReport2, Rollback.
*/

```

```
/*  
* TDOCSFetchHydroRefCursor  
*  
* Fetches data from cursor on hydrology database report. Returns SUCCESS  
* FAILURE, or ENDLOOP.  
*  
* Called by TDOCSHydroRefReport.  
*  
* Calls WriteBatchReport2, Rollback.  
*/
```

```

/*
* tdocsnst.pc
*
* This batch module implements the TDOCS nist db.
*
* 07/30/94 - cjm
* 12/12/94 - cjm - redid header fields & implemented scanning
* 02/08/95 - cjm - solaris port
* 05/16/95 - cjm - upgrade to Pro*C 2.0
* 06/01/95 - cjm - moved from server to batch
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.tdocsnst.pc
* Author:
* Revision:   1.8.0
* Date:       96/01/03-13:33:38
*-----
*/

```

```

/*
* TDOCSNISTRefReport
*
* Generates a report on nist citations.
*
* Called by TDOCSBatchReports.
*
* Calls RPTFileOpen, TDOCSOpenNISTRefCursor, TDOCSFetchNISTRefCursor,
* RPTStuffLine, RPTFormatLine, RPTPrLine, RPTFileClose,
* WriteBatchReport1, WriteBatchReport2, Commit, Rollback.
*/

```

```

/*
* TDOCSOpenNISTRefCursor
*
* Opens cursor on NIST data for report. Returns SUCCESS or FAILURE.
*
* Called by TDOCSNISTRefReport.
*
* Calls WriteBatchReport2, Rollback.
*/

```

```
/*  
* TDOCSFetchNISTRefCursor  
*  
* Fetches data from cursor on nist database report. Returns SUCCESS,  
* ENDLOOP, or FAILURE.  
*  
* Called by TDOCSNISTRefReport.  
*  
* Calls WriteBatchReport2, Rollback.  
*/
```

```

/*
* tdocsrec.pc
*
* Implementation of record handling for tdocs batch.
*
* 05/09/94 - cjm
* 12/14/94 - cjm - generalized header fields and document sets
* 02/08/95 - cjm - solaris port
* 05/16/95 - cjm - upgrade to Pro*C 2.0
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.tdocsrec.pc
* Author:
* Revision:   1.18.0
* Date:       96/01/03-13:33:40
*-----
*/

/*
* TDOCSUpdateSource
*
* For update, updates source (removes update bits). Returns SUCCESS or FAILURE.
*
* Called by TDOCSUpdateDocuments.
*
* Calls WriteBatchReport2, Rollback.
*/

/*
* TDOCSBatchUpdateStatus
*
* Updates status of batched tdocs records. Returns SUCCESS or FAILURE.
*
* Called by TDOCSUpdateDocuments, TDOCSSubmitDocuments.
*
* Calls WriteBatchReport2, Rollback.
*/

/*
* TDOCSOpenBatchCursor
*
* Opens cursor on tdocs for data and document batch. Returns SUCCESS or
* FAILURE.
*
* Called by TDOCSUpdateDocuments, TDOCSSubmitDocuments, TDOCSDeleteDocuments.
*
* Calls WriteBatchReport2, Rollback.
*/

```

```

/*
 * TDOCSFetchBatchRecord
 *
 * Fetches batch record for document update, submit, delete. Returns SUCCESS,
 * ENDLOOP (end of data) or FAILURE.
 *
 * Called by TDOCSUpdateDocuments, TDOCSSubmitDocuments, TDOCSDeleteDocuments.
 *
 * Calls AppendBatchReport, TDOCSFreeAssociatedData, TDOCSGetAssociatedData,
 * WriteBatchReport1, WriteBatchReport2, Rollback.
 */

/*
 * TDOCSGetAssociatedData
 *
 * Fetches associated data for current documentid into list. Returns SUCCESS or
 * FAILURE.
 *
 * Called by TDOCSLoadCSPDocuments, TDOCSLoadQADDocuments, TDOCSLoadTDIDDocuments.
 *
 * Calls SLLInitialize, SLLAppend, TDOCSFreeAssociatedData, WriteBatchReport1,
 * WriteBatchReport2, Rollback.
 */

/*
 * TDOCSFreeAssociatedData
 *
 * Frees authors, addressees, numbers, codes lists.
 *
 * Called by TDOCSFetchBatchRecord.
 *
 * Calls FreeDataList.
 */

/*
 * TDOCSDeleteRecord
 *
 * Deletes tdocs and associated data but not circulation records for
 * document id and set. Returns SUCCESS, ENDLOOP (end of data) or FAILURE.
 *
 * Called by TDOCSDeleteDocuments.
 *
 * Calls WriteBatchReport2, Rollback.
 */

```

```

/*
* tdocsrpt.pc
*
* Implementation of tdocs batch report generation.
*
* 04/03/94 - cjm
* 01/20/95 - cjm - moved statistics report to batch
* 02/08/95 - cjm - solaris port
* 05/16/95 - cjm - upgrade to Pro*C 2.0
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.tdocsrpt.pc
* Author:
* Revision:   1.15.0
* Date:      96/01/03-13:33:43
*-----
*/

```

```

/*
* TDOCSBatchReports
*
* Runs TDOCS batch reports.
*
* Called by TDOCSBatch.
*
* Calls TDOCSStatisticsReport, TDOCSHydroRefReport, TDOCSNISTRefReport.
*/

```

```

/*
* TDOCSStatisticsReport
*
* Generates a statistics report.
*
* Called by TDOCSBatchReports.
*
* Calls RPTFileOpen, RPTPrLine, TDOCSDocSetStatistics, RPTFileClose,
* ReadAsciiFile, SQLError.
*/

```

```

/*
* TDOCSDocSetStatistics
*
* Generates a yearly and monthly statistics on document set. Returns
* SVC_SUCCESS or SVC_SQL_ERROR.
*
* Called by TDOCSStatisticsReport.
*
* Calls RPTPrLine, SQLError.
*/

```



```

/*
 * tdocstop.pc
 *
 * Implementation of topic loading for tdocs batch.
 *
 * 05/09/94 - cjm
 * 12/14/94 - cjm - generalized header fields and document sets
 * 02/08/95 - cjm - solaris port
 * 05/16/95 - cjm - upgrade to Pro*C 2.0
 *
 *-----
 * Archive:      /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.tdocstop.pc
 * Author:
 * Revision:     1.18.0
 * Date:         96/01/03-13:33:45
 *-----
 */

```

```

/*
 * TDOCSUnloadDocument
 *
 * Deletes document from topic. Returns SUCCESS or FAILURE.
 *
 * Called by TDOCSUpdateDocuments, TDOCSDeleteDocuments.
 *
 * Calls nothing.
 */

```

```

/*
 * TDOCSWriteLoadLinkFiles
 *
 * Writes tdocs load file. Returns SUCCESS or FAILURE.
 *
 * Called by TDOCSSubmitDocuments.
 *
 * Calls WriteBatchReport1, TDOCSWriteHeaderFields, TDOCSWriteDocumentLabel,
 * TDOCSWriteDocumentPattern, TDOCSGetImageListFile,
 * TDOCSWriteImageLabelsPatterns, TDOCSAppendAsciiText.
 */

```

```

/*
 * TDOCSWriteHeaderFields
 *
 * Writes header fields. Returns SUCCESS or FAILURE.
 *
 * Called by TDOCSWriterInternalHeader, TDOCSWriteExternalHeader.
 *
 * Calls TDOCSWriteAssociatedData.
 */

```

```

/*
 * TDOCSWriteAssociatedData
 *
 * Writes associated data to load file.
 *
 * Called by TDOCSWriteCSPLoadFile, TDOCSWriteTDILoadFile, TDOCSWriteQALoadFile.
 *
 * Calls nothing.
 */

```

```

/*
 * TDOCSWriteDocumentLabel
 *
 * If there is a document to launch, writes launch pad label. Returns SUCCESS
 * or FAILURE.
 *
 * Called by TDOCSWriteLoadFile.
 *
 * Calls nothing.
 */

```

```

/*
 * TDOCSWriteDocumentPattern
 *
 * If there is a document to launch, writes launch pad pattern. Returns SUCCESS
 * or FAILURE.
 *
 * Called by TDOCSWriteLinkFile.
 *
 * Calls nothing.
 */

```

```

/*
 * TDOCSGetImageListFile
 *
 * Creates and opens image list file. Returns image list file pointer.
 *
 * Called by TDOCSWriteLoadLinkFiles.
 *
 * Calls WriteBatchReport1.
 */

```

```
/*
 * TDOCSWriteImageLabelsPatterns
 *
 * If there are images to launch, writes launch pad labels. Returns SUCCESS or
 * FAILURE.
 *
 * Called by TDOCSWriteLoadFile.
 *
 * Calls nothing.
 */
```

```
/*
 * TDOCSAppendAsciiText
 *
 * Appends any ascii text to the header file for loading/linking. Returns
 * SUCCESS or FAILURE.
 *
 * Called by TDOCSWriteLoadLinkFiles.
 *
 * Calls nothing.
 */
```

```
/*
 * TDOCSLoadPartition
 *
 * Loads partition by creation or insertion. Note that for hyperlinking, it is
 * the link files with link patterns that must be indexed and linked but load
 * files that must be in place for loading into the topic viewer. Returns
 * SUCCESS or FAILURE.
 *
 * Called by TDOCSSubmitDocuments.
 *
 * Calls nothing.
 */
```

8.2.3 Server H

```
/*
 * dat.h
 *
 * This server header contains data handling defines, macros, and prototypes.
 *
 * 03/03/94 - cjm
 * 04/01/94 - cjm - merged tdocs with rpd
 * 02/09/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.dat.h
 * Author:
 * Revision:   1.15.0
 * Date:       95/11/13-10:59:28
 *-----
 */
```

```
/*
* db.h
*
* This server header contains general database defines, macros, and
* prototypes.
*
* 02/21/94 - cjm
* 04/01/94 - cjm - merged tdocs with rpd
* 05/15/94 - cjm - removed topic loading to batch
* 12/12/94 - cjm - redid header fields & implemented scanning
* 02/09/95 - cjm - solaris port
* 02/20/95 - cjm - removed reliance on tcp for db access
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.db.h
* Author:
* Revision:   2.22.0
* Date:      95/11/13-10:59:29
*-----
*/
```

```
/*
* doc.h
*
* This server header contains document read/write defines, macros,
* and prototypes.
*
* 03/03/94 - cjm
* 04/01/94 - cjm - merged tdocs with rpd
* 02/09/95 - cjm - solaris port
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.doc.h
* Author:
* Revision:   2.22.0
* Date:       95/11/13-10:59:33
*-----
*/
```

```
/*
 * env.h
 *
 * This server header contains environment defines and prototypes.
 *
 * 03/31/95 - cjm
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.env.h
 * Author:
 * Revision:   1.12.0
 * Date:       95/11/13-10:59:36
 *-----
 */
```

```
/*
 * log.h
 *
 * This server header contains logging defines, macros, and
 * prototypes.
 *
 * 09/07/94 - cjm
 * 02/09/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.log.h
 * Author:
 * Revision:   2.23.0
 * Date:       95/11/13-10:59:39
 *-----
 */
```



```
/*
 * max.h
 *
 * This server header contains RPDOCS common defines for column widths.
 *
 * Values represent maximums across tables - e.g., RPD document numbers
 * are 20 chars whereas in TDOCS its 13 and in NIST 6, but the max
 * herein defined is 20+1.
 *
 * 02/09/95 - cjm
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.max.h
 * Author:
 * Revision:   1.15.0
 * Date:      95/11/13-10:59:40
 *-----
 */
```

```
/*
 * mem.h
 *
 * This server header defines memory management prototypes.
 *
 * 07/11/94 - cjm
 * 02/09/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.mem.h
 * Author:
 * Revision:   2.22.0
 * Date:       95/11/13-10:59:42
 *-----
 */
```

```
/*
 * rpd.h
 *
 * This server header contains RPD defines, macros, and prototypes.
 *
 * 02/21/94 - cjm
 * 04/01/94 - cjm - merged tdocs with rpd
 * 02/09/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.rpd.h
 * Author:
 * Revision:   2.23.0
 * Date:       95/11/13-11:02:40
 *-----
 */
```

```
/*
 * rpddoc.h
 *
 * Implementation of da server.
 *
 * Header file for RPD portion of RPDOCS server.
 *
 *-----
 * Archive:    /home/samson/sanjeev/rpdocs/development/server/SCCS/s.tdocsdoc.h
 * Author:
 * Revision:   1.17.0
 * Date:       95/11/13-11:01:26
 *-----
 */
```

```
/*
 * rpddtd.h
 *
 * This server header contains RPD SQL*DTD defines, macros, and prototypes.
 *
 * 02/21/94 - cjm
 * 04/01/94 - cjm - merged tdocs with rpd
 * 02/09/95 - cjm - solaris port
 *
 * -----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.rpddtd.h
 * Author:
 * Revision:   1.11.0
 * Date:       95/11/13-10:59:46
 * -----
 */
```

```

/*
 * rpdocs.h
 *
 * Implementation of da server.
 *
 * This header include common headers. The order of includes is
 * IMPORTANT!
 *
 * It also handles defines, macros, and prototypes common to rpd
 * and tdocs.
 *
 * 04/06/94 - cjm
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.rpdocs.h
 * Author:
 * Revision:   2.7.0
 * Date:       95/11/13-11:00:50
 *-----
 */

```

```
/*
 * rpd rpt.h
 *
 * This server header contains RPD report defines, macros, and prototypes.
 *
 * 02/21/94 - cjm
 * 04/01/94 - cjm - merged tdocs with rpd
 * 02/09/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.rpd rpt.h
 * Author:
 * Revision:   1.11.0
 * Date:       95/11/13-11:00:56
 *-----
 */
```

```
/*
 * rpdrw.h
 *
 * This server header contains RPD report writer defines, macros, & prototypes.
 *
 * 02/21/94 - cjm
 * 04/01/94 - cjm - merged tdocs with rpd
 * 02/09/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.rpdrw.h
 * Author:
 * Revision:   1.11.0
 * Date:       95/11/13-11:01:02
 *-----
 */
```



```
/*
 * tdocssyn.h
 *
 * This server header contains RPD synchronization defines, macros, and
 * prototypes.
 *
 * 04/03/95 - cjm
 *
 *-----
 * Archive:    /home/samson/sanjeev/rpdocs/development/server/SCCS/s.tdocssyn.h
 * Author:
 * Revision:   1.13.0
 * Date:       95/11/13-11:01:59
 *-----
 */
```

```
/*
 * rpt.h
 *
 * This server header contains common report generation defines,
 * macros, and prototypes.
 *
 * 04/14/94 - cjm
 * 02/09/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.rpt.h
 * Author:
 * Revision:   2.22.0
 * Date:       95/11/13-11:01:10
 *-----
 */
```

```
/*
 * sllist.h
 *
 * This server header contains single linked list defines, macros,
 * and prototypes.
 *
 * 01/20/94 - cjm - created
 * 02/09/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.sllist.h
 * Author:
 * Revision:   2.22.0
 * Date:       95/11/13-11:01:14
 *-----
 */
```

```
/*
* tdocs.h
*
* This server header contains TDOCS defines, macros, and prototypes.
*
* 03/24/94 - cjm
* 04/01/94 - cjm - merged tdocs with rpd
* 05/15/94 - cjm - removed topic loading to batch
* 12/12/94 - cjm - redid header fields & implemented scanning
* 02/09/95 - cjm - solaris port
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocs.h
* Author:
* Revision:   2.23.0
* Date:       95/11/13-11:01:16
*-----
*/
```

```
/*
 * tdocsdoc.h
 *
 * Implementation of da server.
 *
 * Header file for TDOCS portion of RPDOCS server.
 *
 * 03/24/94 - cjm
 * 04/01/94 - cjm - merged tdocs with rpdocs server
 * 02/08/95 - cjm - solaris port
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocsdoc.h
 * Author:
 * Revision:   1.17.0
 * Date:       95/11/13-11:01:26
 *-----
 */
```

```
/*
 * tdocshyd.h
 *
 * This server header contains Hydrology defines, macros, and
 * prototypes.
 *
 * 07/15/94 - cjm
 * 02/09/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocshyd.h
 * Author:
 * Revision:   2.18.0
 * Date:       95/11/13-11:01:30
 *-----
 */
```

```
/*
* tdocslbl.h
*
* This server header file contains TDOCS labelling defines, macros,
* and prototypes.
*
* 03/24/94 - cjm
* 04/01/94 - cjm - merged tdocs with rpd
* 12/12/94 - cjm - redid header fields & implemented scanning
* 02/09/95 - cjm - solaris port
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocslbl.h
* Author:
* Revision:   1.23.0
* Date:       95/11/13-11:01:35
*-----
*/
```

```
/*
 * tdocsnst.h
 *
 * This server header contains NIST defines, macros, and prototypes.
 *
 * 07/30/94 - cjm
 * 02/09/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocsnst.h
 * Author:
 * Revision:   2.18.0
 * Date:       95/11/13-11:01:42
 *-----
 */
```



```
/*
 * tdocsrpt.h
 *
 * This server header contains TDOCS report defines, macros, and
 * prototypes.
 *
 * 03/24/94 - cjm
 * 04/01/94 - cjm - merged tdocs with rpdocs server
 * 12/12/94 - cjm - redid header fields & implemented scanning
 * 02/09/95 - cjm - solaris port
 *
 * -----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocsrpt.h
 * Author:
 * Revision:   1.24.0
 * Date:       95/11/13-11:01:49
 * -----
 */
```

```
/*
 * tdocssyn.h
 *
 * This server header contains TDOCS synchronization defines, macros, and
 * prototypes.
 *
 * 04/03/95 - cjm
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/server/SCCS/s.tdocssyn.h
 * Author:
 * Revision:   1.13.0
 * Date:       95/11/13-11:01:59
 *-----
 */
```

8.2.4 Batch H

```
/*
 * batch.h
 *
 * Header file for batch.
 *
 * 05/05/94 - cjm
 * 02/08/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.batch.h
 * Author:
 * Revision:   2.17.0
 * Date:       96/01/03-13:33:05
 *-----
 */
```

```
/*
 * batchdb.h
 *
 * This header contains general database defines and macros.
 *
 * 02/21/94 - cjm
 * 02/08/95 - cjm - solaris port
 * 02/20/95 - cjm - removed reliance on tcp for db access
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.batchdb.h
 * Author:
 * Revision:   2.17.0
 * Date:       96/01/03-13:33:06
 *-----
 */
```

```
/*
 * batchrpt.h
 *
 * Header file for rpdocs_batch reports.
 *
 * 05/06/94 - cjm
 * 02/08/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.batchrpt.h
 * Author:
 * Revision:   2.18.0
 * Date:       96/01/03-13:33:09
 *-----
 */
```

```
/*
 * env.h
 *
 * This server header contains environment defines and prototypes.
 *
 * 03/31/95 - cjm
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.env.h
 * Author:
 * Revision:   1.1.0
 * Date:       95/04/19-13:51:12
 *-----
 */
```

```
/*
 * max.h
 *
 * This server header contains RPDOCS common defines for column widths.
 *
 * Values represent maximums across tables - e.g., RPD document numbers
 * are 20 chars whereas in TDOCS its 13 and in NIST 6, but the max
 * herein defined is 20+1.
 *
 * 02/09/95 - cjm
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.max.h
 * Author:
 * Revision:   1.3.0
 * Date:       96/01/03-13:33:51
 *-----
 */
```

```
/*
 * pieces.h
 *
 * Header file for rpdocs_batch-pieces interface.
 *
 * 05/19/94 - cjm
 * 02/08/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.pieces.h
 * Author:
 * Revision:   2.17.0
 * Date:       96/01/03-13:33:13
 *-----
 */
```



```
/*
 * rpd.h
 *
 * Header file for rpd batch.
 *
 * 05/05/94 - cjm
 * 02/08/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.rpd.h
 * Author:
 * Revision:   1.16.0
 * Date:      96/01/03-13:33:14
 *-----
 */
```

```

/*
 * sllist.h
 *
 * Implementation of da server.
 *
 * Header file for singly linked list where nodes use void* to point
 * to any data, thus applications that use sllists must allocate and
 * free data.
 *
 * 01/20/94 - cjm
 * 02/08/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.sllist.h
 * Author:
 * Revision:   1.8.0
 * Date:       96/01/03-13:33:12
 *-----
 */

```

```
/*
 * tdocs.h
 *
 * Header file for tdocs batch.
 *
 * 05/05/94 - cjm
 * 12/14/94 - cjm - generalized header fields and document sets
 * 02/08/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.tdocs.h
 * Author:
 * Revision:   1.18.0
 * Date:       96/01/03-13:33:26
 *-----
 */
```

```
/*
 * tdocshyd.h
 *
 * This server header contains Hydrology defines, macros, and
 * prototypes.
 *
 * 07/15/94 - cjm
 * 02/09/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.tdocshyd.h
 * Author:
 * Revision:   1.8.0
 * Date:       96/01/03-13:33:30
 *-----
 */
```

```
/*
 * tdocsnst.h
 *
 * This server header contains NIST defines, macros, and prototypes.
 *
 * 07/30/94 - cjm
 * 02/09/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.tdocsnst.h
 * Author:
 * Revision:   1.8.0
 * Date:       96/01/03-13:33:34
 *-----
 */
```

```
/*
* tdocsrpt.h
*
* Header file for TDOCS batch report generation.
*
* 03/24/94 - cjm
* 04/01/94 - cjm - merged tdocs with rpdocs server
* 01/20/95 - cjm - moved statistics report to batch
* 02/08/95 - cjm - solaris port
*
*-----
* Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.tdocsrpt.h
* Author:
* Revision:   1.15.0
* Date:       96/01/03-13:33:41
*-----
*/
```

```
/*
 * writer.h
 *
 * Header file for rpdocs_batch-pieces interface.
 *
 * 05/19/94 - cjm
 * 02/08/95 - cjm - solaris port
 *
 *-----
 * Archive:    /home/samson/dlincoln/rpdocs/development/batch/SCCS/s.writer.h
 * Author:
 * Revision:   2.17.0
 * Date:       96/01/03-13:33:47
 *-----
 */
```